HW5: Vector-based similarity search!

Summary

In this final HW, you will use Weaviate [https://weaviate.io/], which is a vector DB (stores data as vectors, and computes a search query by vectorizing it and doing similarity search with existing vectors).

Description

The (three) steps we need are really simple:

- install Weaviate plus vectorizer via Docker as images, run them as containers
- specify a schema for data, upload data (in .json format) to have it be vectorized
- run a query (which gets vectorized and sim-searched), get back results (as JSON)

The following sections describe the above steps. The entire HW will only take you 2 to 3 hours to complete, pinky swear :)

1. Installing Weaviate and a vectorizer module

After installing Docker, bring it up (eg. on Windows, run Docker Desktop). Then, in your (ana)conda shell, run this docker-compose command that uses this yaml 'docker-compose.yml' config file to pull in two images: the 'weaviate' one, and a text2vec transformer called 't2v-transformers':

```
docker-compose up -d
```

These screenshots show the progress, completion, and subsequently, two containers automatically being started (one for weaviate, one for t2v-transformers):

```
(base) C:\Users\satyc>
(base) C:\Users\satyc>
(base) C:\Users\satyc>
(base) C:\Users\satyc>docker-compose up -d
[+] Running 0/19
 - t2v-transformers Pulling
   - 26c5c85e47da Waiting
   - 9e79879be9c7 Waiting
   - 9ad47fcd2c0c Waiting
   - 9da6498f32c0 Waiting
   - 756350766a45 Waiting
   - a64e61a28d1e Waiting
   - 39a8c791c8b5 Waiting
   - bfccbc963b3f Waiting
   - d808540300c6 Waiting
   - 188a0f8b4cb2 Waiting
   - 8ad63110c7d9 Waiting
   - 66c95f4520ed Waiting
   - 1df5bed45a5d Waiting
 - weaviate Pulling
   - f56be85fc22e Extracting [>                        ]  65.54kB/3.375MB
   - fc5ceff4c76f Downloading [==>                      ]  734.1kB/13.94MB
   - c9d28bc41971 Downloading [==============>          ]  750.5kB/2.674MB
   - 10cc92ce0a30 Waiting
```

```
(base) C:\Users\satyc>
(base) C:\Users\satyc>docker-compose up -d
[+] Running 7/19
 - t2v-transformers Pulling
   - 26c5c85e47da Pull complete
   - 9e79879be9c7 Pull complete
   - 9ad47fcd2c0c Extracting  [==============================================>  ]  10.09MB/11.53MB
   - 9da6498f32c0 Download complete
   - 756350766a45 Download complete
   - a64e61a28d1e Download complete
   - 39a8c791c8b5 Downloading [=============================================>   ]  12.08MB/13.93MB
   - bfccbc963b3f Download complete
   - d808540300c6 Download complete
   - 188a0f8b4cb2 Downloading [>                        ]  5.947MB/4.72GB
   - 8ad63110c7d9 Download complete
   - 66c95f4520ed Downloading [=>                       ]  4.31MB/195.1MB
   - 1df5bed45a5d Waiting
 - weaviate Pulled
   - f56be85fc22e Pull complete
   - fc5ceff4c76f Pull complete
   - c9d28bc41971 Pull complete
   - 10cc92ce0a30 Pull complete
```

```
(base) C:\Users\satyc>docker-compose up -d
[+] Running 19/19
 - t2v-transformers Pulled                                                    1450.6s
   - 26c5c85e47da Pull complete                                                 39.2s
   - 9e79879be9c7 Pull complete                                                 39.7s
   - 9ad47fcd2c0c Pull complete                                                 41.5s
   - 9da6498f32c0 Pull complete                                                 41.7s
   - 756350766a45 Pull complete                                                 42.6s
   - a64e61a28d1e Pull complete                                                 43.3s
   - 39a8c791c8b5 Pull complete                                                 46.0s
   - bfccbc963b3f Pull complete                                                 47.9s
   - d808540300c6 Pull complete                                                 48.0s
   - 188a0f8b4cb2 Pull complete                                                1423.4s
   - 8ad63110c7d9 Pull complete                                                1424.9s
   - 66c95f4520ed Pull complete                                                1442.9s
   - 1df5bed45a5d Pull complete                                                1443.5s
 - weaviate Pulled                                                             28.5s
   - f56be85fc22e Pull complete                                                 3.5s
   - fc5ceff4c76f Pull complete                                                 23.0s
   - c9d28bc41971 Pull complete                                                 23.6s
   - 10cc92ce0a30 Pull complete                                                 23.8s
[+] Running 3/3
 - Network satyc_default            Created                                     1.2s
 - Container satyc-weaviate-1       Started                                    10.7s
 - Container satyc-t2v-transformers-1 Started                                  10.7s
```

Containers Give feedback 🗨

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

⚪ Only show running containers   🔍 Search   ⋮

		Name	Image	Status	Port(s)	Last started	Actions
☐	⌄ ▨	**satyc**	-	Running (2/2)		3 minutes ago	▪ ⋮ 🗑
☐	▨	**weaviate-1** 4996ceb41ebe 📋	semitechnologies/weaviate:1	Running	8080:8080 ↗	3 minutes ago	▪ ⋮ 🗑
☐	▨	**t2v-transformers-1** 6276609054ef 📋	semitechnologies/transform:	Running		3 minutes ago	▪ ⋮ 🗑

Yeay! Now we have the vectorizer transformer (to convert sentences to vectors), and weaviate (our vector DB search engine) running! On to data handling :)

2. Loading data to search for

This is the data that we'd like searched, part of which will get returned to us as results. The data is conveniently represented as an array of JSON documents, similar to Solr/Lunr. Here is our data file, conveniently named data.json (you can rename it if you like) - place it in the 'root' directory of your webserver (see below). As you can see, each datum/'row'/JSON contains three k:v pairs, with 'Category', 'Question', 'Answer' as keys - as you might guess, it seems to be in Jeopardy(TM) answer-question (reversed) format :) The file is actually called jeopardy-tiny.json, I simply made a local copy called data.json.

The overall idea is this: we'd get the 10 documents vectorized, then specify a query word, eg. 'biology', and automagically have that pull up related docs, eg. the 'DNA' one! This is a really useful semantic search feature where we don't need to specify exact keywords to search for.

Start by installing the weaviate Python client:

```
pip install weaviate-client
```

So, how to submit our JSON data, to get it vectorized? Simply use this Python script, do:

```
python weave-loadData.py
```

You will see this:


```
Anaconda Prompt                                                                                          —   □   ×
(base) C:\Users\satyc>python weave-loadData.py
C:\Users\satyc\Miniconda3\lib\site-packages\requests\__init__.py:104: RequestsDependencyWarning: urllib3 (1.26.8) or chardet (5.0.0)/charset_normalizer (2.0.4) doesn't mat
ch a supported version!
  RequestsDependencyWarning)

importing datum: 0
properties: {'answer': 'Liver', 'question': 'This organ removes excess glucose from the blood & stores it as glycogen', 'category': 'SCIENCE'}

importing datum: 1
properties: {'answer': 'Elephant', 'question': "It's the only living mammal in the order Proboseidea", 'category': 'ANIMALS'}

importing datum: 2
properties: {'answer': 'the nose or snout', 'question': 'The gavial looks very much like a crocodile except for this bodily feature', 'category': 'ANIMALS'}

importing datum: 3
properties: {'answer': 'Antelope', 'question': 'Weighing around a ton, the eland is the largest species of this animal in Africa', 'category': 'ANIMALS'}

importing datum: 4
properties: {'answer': 'the diamondback rattler', 'question': 'Heaviest of all poisonous snakes is this North American rattlesnake', 'category': 'ANIMALS'}

importing datum: 5
properties: {'answer': 'species', 'question': "2000 news: the Gunnison sage grouse isn't just another northern sage grouse, but a new one of this classification", 'categor
y': 'SCIENCE'}

importing datum: 6
properties: {'answer': 'wire', 'question': 'A metal that is ductile can be pulled into this while cold & under pressure', 'category': 'SCIENCE'}

importing datum: 7
properties: {'answer': 'DNA', 'question': 'In 1953 Watson & Crick built a model of the molecular structure of this, the gene-carrying substance', 'category': 'SCIENCE'}

importing datum: 8
properties: {'answer': 'the atmosphere', 'question': 'Changes in the tropospheric layer of this are what gives us weather', 'category': 'SCIENCE'}

importing datum: 9
properties: {'answer': 'Sound barrier', 'question': 'In 70-degree air, a plane traveling at about 1,130 feet per second breaks it', 'category': 'SCIENCE'}

(base) C:\Users\satyc>
```

If you look in the script, you'll see that we are creating a schema - we create a class called 'SimSearch' (you can call it something else if you like). The data we load into the DB, will be associated with this class (the last line in the script does this via add_data_object()).

NOTE - **you NEED to run a local webserver** [in a separate ana/conda (or other) shell], eg. via 'python serveit.py' like you did for HW4 - it's what will 'serve' data.json to weaviate :)

Great! Now we have specified our searchable data, which has been first vectorized (by 't2v-transformers'), then stored as vectors (in weaviate).

Only one thing left: querying!

3. Querying our vectorized data

To query, use this simple shell script called weave-doQuery.sh, and run this:

```
sh weave-doQuery.sh
```

As you can see in the script, we search for 'physics'-related docs, and sure enough, that's what we get:


```
(base) C:\Users\satyc>
(base) C:\Users\satyc>cat weave-doQuery.sh
echo '{
  "query": "{
    Get{
      SimSearch (
        limit: 3
        nearText: {
          concepts: [\"physics\"],
        }
      ){
        question
        answer
        category
      }
    }
}"
}' | curl \
  -X POST \
  -H 'Content-Type: application/json' \
  -d @- \
  localhost:8080/v1/graphql  # Replace this with

(base) C:\Users\satyc>
(base) C:\Users\satyc>
(base) C:\Users\satyc>sh weave-doQuery.sh
{"data":{"Get":{"SimSearch":[{"answer":"Sound barrier","category":"SCIENCE","question":"In 70-degree air, a plane traveling at about 1,130 feet per second breaks it"},{"an
swer":"the atmosphere","category":"SCIENCE","question":"Changes in the tropospheric layer of this are what gives us weather"},{"answer":"wire","category":"SCIENCE","questi
on":"A metal that is ductile can be pulled into this while cold & under pressure"}]}}}
```

Why is this exciting? Because the word 'physics' isn't in any of our results!

Now it's your turn:

• first, MODIFY the contents of data.json, to replace the 10 docs in it, with your own data, where you'd replace ("Category","Question","Answer") with ANYTHING you like, eg. ("Author","Book","Summary"), ("MusicGenre","SongTitle","Artist"), ("School","CourseName","CourseDesc"), etc, etc - HAVE fun coming up with this! You can certainly add more docs, eg. have 20 of them instead of 10

• next, MODIFY the query keyword(s) in the query .sh file - eg. you can query for 'computer science' courses, 'female' singer, 'American' books, ['Indian','Chinese'] food dishes (the query list can contain multiple items), etc. Like in the above screenshot, 'cat' the query, then run it, and get a screenshot to submit. BE SURE to also modify the data loader .py script, to put in your keys (instead of ("Category","Question","Answer"))

That's it, you're done w/ the HW :) In RL you will have a .json or .csv file (or data in other formats) with BILLIONS of items! Later, do feel free to play with bigger JSON files, eg. this 200K Jeopardy JSON file :)

FYI/'extras'
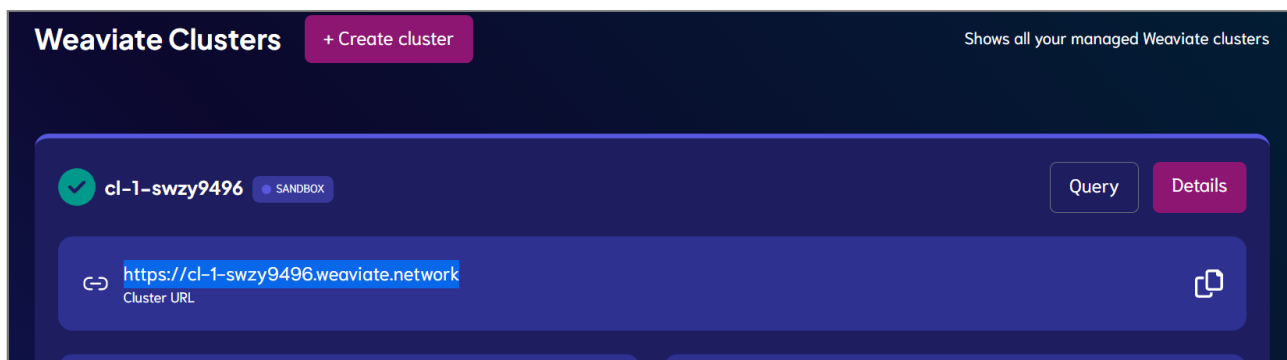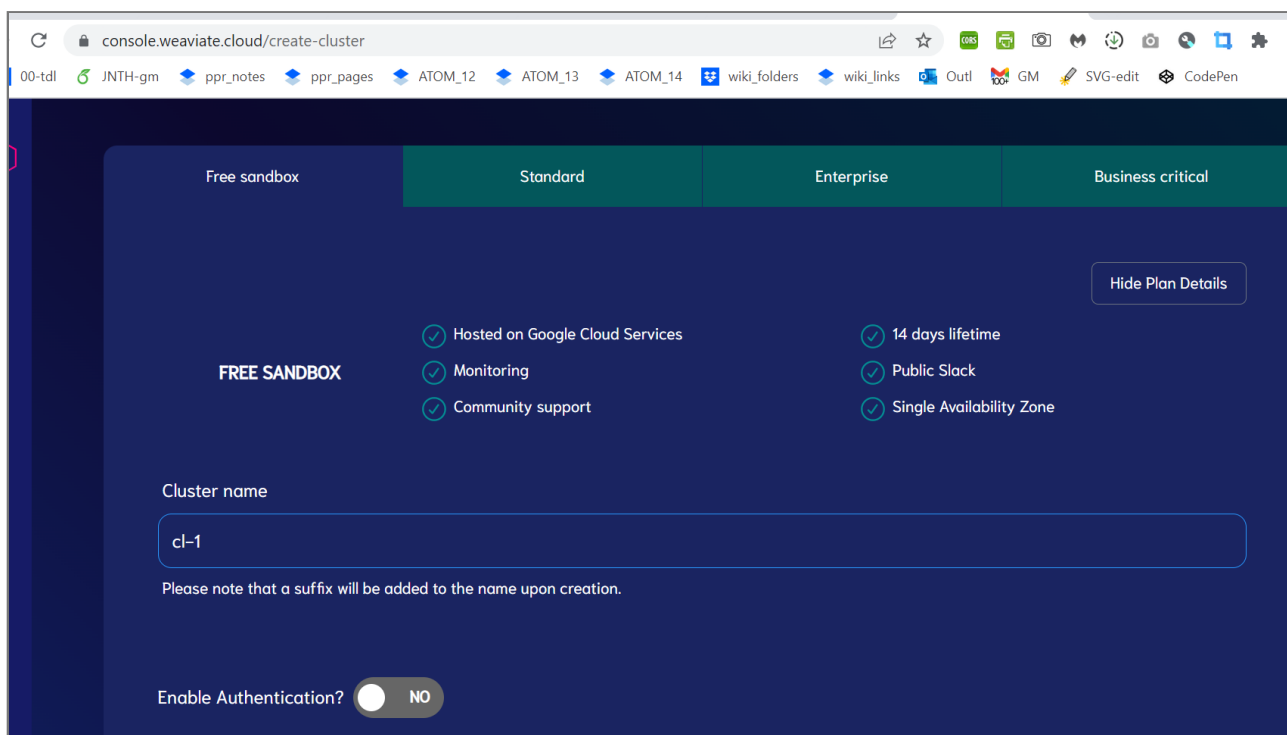
Here are two more things you can do, via 'curl':

```
(base) C:\Users\satyc>curl localhost:8080/v1/meta
{"hostname":"http://[::]:8080","modules":{"text2vec-transformers":{"model":{"_name_or_path":"./models/model","add_cross_attention":false,"architectures":["BertModel"],"attention_probs_dropout_prob":0.1,"bad_words_ids":null,"begin_suppress_tokens":null,"bos_token_id":null,"chunk_size_feed_forward":0,"classifier_dropout":null,"cross_attention_hidden_size":null,"decoder_start_token_id":null,"diversity_penalty":0,"do_sample":false,"early_stopping":false,"encoder_no_repeat_ngram_size":0,"eos_token_id":null,"exponential_decay_length_penalty":null,"finetuning_task":null,"forced_bos_token_id":null,"forced_eos_token_id":null,"gradient_checkpointing":false,"hidden_act":"gelu","hidden_dropout_prob":0.1,"hidden_size":384,"id2label":{"0":"LABEL_0","1":"LABEL_1"},"initializer_range":0.02,"intermediate_size":1536,"is_decoder":false,"is_encoder_decoder":false,"label2id":{"LABEL_0":0,"LABEL_1":1},"layer_norm_eps":1e-12,"length_penalty":1,"max_length":20,"max_position_embeddings":512,"min_length":0,"model_type":"bert","no_repeat_ngram_size":0,"num_attention_heads":12,"num_beam_groups":1,"num_beams":1,"num_hidden_layers":6,"num_return_sequences":1,"output_attentions":false,"output_hidden_states":false,"output_scores":false,"pad_token_id":0,"position_embedding_type":"absolute","prefix":null,"problem_type":null,"pruned_heads":{},"remove_invalid_values":false,"repetition_penalty":1,"return_dict":true,"return_dict_in_generate":false,"sep_token_id":null,"suppress_tokens":null,"task_specific_params":null,"temperature":1,"tf_legacy_loss":false,"tie_encoder_decoder":false,"tie_word_embeddings":true,"tokenizer_class":null,"top_k":50,"top_p":1,"torch_dtype":"float32","torchscript":false,"transformers_version":"4.27.2","type_vocab_size":2,"typical_p":1,"use_bfloat16":false,"use_cache":true,"vocab_size":30522}}},"version":"1.18.4"}
```

[you can also do 'http://localhost:8080/v1/meta' in your browser]

```
(base) C:\Users\satyc>curl localhost:8080/v1/schema
{"classes":[{"class":"Question","invertedIndexConfig":{"bm25":{"b":0.75,"k1":1.2},"cleanupIntervalSeconds":60,"stopwords":{"additions":null,"preset":"en","removals":null}},"moduleConfig":{"text2vec-transformers":{"poolingStrategy":"masked_mean","vectorizeClassName":true}},"properties":[{"dataType":["text"],"description":"This property was generated by Weaviate's auto-schema feature on Thu Apr 27 20:25:22 2023","moduleConfig":{"text2vec-transformers":{"skip":false,"vectorizePropertyName":false}},"name":"answer","tokenization":"word"},{"dataType":["text"],"description":"This property was generated by Weaviate's auto-schema feature on Thu Apr 27 20:25:22 2023","moduleConfig":{"text2vec-transformers":{"skip":false,"vectorizePropertyName":false}},"name":"question","tokenization":"word"},{"dataType":["text"],"description":"This property was generated by Weaviate's auto-schema feature on Thu Apr 27 20:25:22 2023","moduleConfig":{"text2vec-transformers":{"skip":false,"vectorizePropertyName":false}},"name":"category","tokenization":"word"}],"replicationConfig":{"factor":1},"shardingConfig":{"virtualPerPhysical":128,"desiredCount":1,"actualCount":1,"desiredVirtualCount":128,"actualVirtualCount":128,"key":"_id","strategy":"hash","function":"murmur3"},"vectorIndexConfig":{"skip":false,"cleanupIntervalSeconds":300,"maxConnections":64,"efConstruction":128,"ef":-1,"dynamicEfMin":100,"dynamicEfMax":500,"dynamicEfFactor":8,"vectorCacheMaxObjects":1000000000000,"flatSearchCutoff":40000,"distance":"cosine","pq":{"enabled":false,"bitCompression":false,"segments":0,"centroids":256,"encoder":{"type":"kmeans","distribution":"log-normal"}}},"vectorIndexType":"hnsw","vectorizer":"text2vec-transformers"}]}
```

[you can also do 'http://localhost:8080/v1/schema' in your browser]

Weaviate has a cloud version too, called WCS - you can try that as an alternative to using the Dockerized version:

Run this :)

Also, for fun, see if you can print the raw vectors for the data (the 10 docs)...

More info:

• https://weaviate.io/developers/weaviate/quickstart/end-to-end

• https://weaviate.io/developers/weaviate/installation/docker-compose

• https://medium.com/semi-technologies/what-weaviate-users-should-know-about-docker-containers-1601c6afa079

• https://weaviate.io/developers/weaviate/modules/retriever-vectorizer-modules/text2vec-transformers

---

What to submit

• your data.json that contains the data (10 docs) you put in

• a screenshot of the 'cat' of your query and the results

Alternative (!) submission [omg]

You can just submit a README.txt file that notes why you didn't/couldn't do the HW.

"Wait, WHAT?" Turns out **THIS HW IS OPTIONAL**!!! You will get the full 10 points for this HW, regardless of what you submit - but you DO need to submit something - either the .json+screenshot combo, or a README. **Such a deal!** For your own benefit, it's worth doing the HW of course - you'll get first-hand experience using a vector DB (Weaviate is a worthy alternative to Pinecone btw); but if you aren't able, we understand (hope you'll do it after the course!) :)

Getting help

There is a hw5 'forum' on Piazza, for you to post questions/answers. You can also meet w/ the TAs, CPs, or me.

Have fun! This is a really useful piece of tech to know. **Vector DBs are sure be used more and more in the near future, as a way to provide 'infinite external runtime memory' for pretrained LLMs.**