Last updated 2024-12-25T21:20:17.000Z

Feature Vectors and Feature Sets

Feature Vectors

Feature Vectors are numerical representations of entities, such as items or annotations. These vectors can vary in length and capture essential characteristics of the entities they represent.

Key Concepts

- Representation: Feature vectors can represent various entities, including items and annotations.
- Flexibility: The length of a feature vector can vary based on the specific use case or model.
- Origin: Feature vectors can be derived from raw data or generated by machine learning models.

Examples

1. **Image Embedding** - a pre-trained model like ResNet can generate a 1024-dimensional feature vector for an image, capturing its visual characteristics.
2. **Natural Language Processing (NLP)** - Large Language Models (LLMs) like BERT and RoBERTa use embeddings to represent textual elements (words, phrases, sentences) as dense vectors. These embeddings encapsulate semantic meaning and linguistic relationships.
3. **Custom Segmentation** - in marketing and business analytics, for example, customer data can be represented as feature vectors. For example, a customer's feature vector might include:

```
Age: 35
Annual Income: 75000
Years as Customer: 5
Number of Purchases: 20
Average Purchase Value: 150
```

This feature vector [35, 75000, 5, 20, 150] represents key characteristics of the customer, which can be used for segmentation, personalized marketing, or predictive modeling without involving neural networks.

Feature Set

Feature vectors are grouped under a `Feature Set`. A Feature Set contains the metadata for describing a set of feature vectors, and includes general information about the features, e.g. size, type, etc.

Create, Get, Delete

First we need to create a feature set on a project. Creating a set using the SDK:

```
import dtlpy as dl

project = dl.projects.get(project_id='60c8561b374417847ff59fba')
feature_set = project.feature_sets.create(name='rag_embeddings_BAAI_bge-large-en'
                                          size=1024,
                                          set_type='embeddings',
                                          entity_type=dl.FeatureEntityType.ITEM)
```

A Dataloop model entity can be connected directly to the feature set:

```
feature_set = project.feature_sets.create(name='text-embedding-3',
                                          set_type='embeddings',
                                          entity_type=dl.FeatureEntityType.ITEM,
                                          model_id=project.models.get('my-embedde
                                          size=1536)
```

Now we can add feature vectors for items using HuggingFace

```
from langchain_community.embeddings import HuggingFaceBgeEmbeddings
import torch
import dtlpy as dl
import tqdm

model_name = "BAAI/bge-large-en"
model_kwargs = {'device': 'cuda' if torch.cuda.is_available() else 'cpu'}
encode_kwargs = {'normalize_embeddings': True}
embeddings = HuggingFaceBgeEmbeddings(cache_folder='.cache',
                                      model_name=model_name,
                                      model_kwargs=model_kwargs,
                                      encode_kwargs=encode_kwargs
                                      )

dataset = dl.datasets.get(dataset_id='<dataset id>')
items = dataset.items.list()
pbar = tqdm.tqdm(total=items.items_count)

for item in items.all():
    text = item.download(save_locally=False).read().decode('utf8')
    embeddings = embeddings.embed_documents([text])[0]
    feature = feature_set.features.create(value=embeddings,
```

```
                                        entity=item)
    pbar.update()
```

After that, *GET* and *DELETE* are easy, same as all other SDK entities:

```
import dtlpy as dl

feature_set = dl.feature_sets.get(feature_set_name='rag_embeddings_BAAI_bge-large
feature_set.delete()
```

Getting the feature vector value from of a single item

```
item = dl.items.get(item_id='618126e38f1fa2b52ae96d05')
items_features = list(item.features.list().all())
print(f'This item has {len(items_features)} feature vectors')
```

Querying and Nearest Neighbours

We can query over feature vectors distance.

First we will define the vector to query on. Since the filter returns sorted results by the distance, we can use the `page_size` as the number of nearest neighbours we want to get (`k`).

Note: The vector should be the same size as the feature set.

```
vector = [3, 1, 4, 1, 5, ..., 9]
k = 100
```

Then we will query the feature set for the nearest neighbours:

```
custom_filter = {
    'filter': {'$and': [{'hidden': False}, {'type': 'file'}]},
    'page': 0,
    'pageSize': k,
    'resource': 'items',
    'join': {
        'on': {
            'resource': 'feature_vectors',
            'local': 'entityId',
            'forigen': 'id'
        },
        'filter': {
            'value': {
                '$euclid': {
```

```
                    'input': vector,
                    '$euclidSort': {'eu_dist': 'ascending'}
                }
            },
            'featureSetId': feature_set.id
        },
    }
}

# Filter items by feature vector distance
filters = dl.Filters(custom_filter=custom_filter,
                     resource=dl.FiltersResource.ITEM)

res = dataset.items.list(filters=filters)
print(res.items_count)

for i_item, item in enumerate(res.items):
    print(item.name)
    # get the feature vector value for the item
    filt = dl.Filters(resource=dl.FiltersResource.FEATURE, field='entityId', valu
    vector = list(feature_set.features.list(filters=filt).all())
    print(vector[0].value)
    if i_item == 10:
        break
```

Query Using Distance Threshold

We can also query using a distance threshold. This will return all items that are within the threshold distance from the query vector.

```
custom_filter = {
    'filter': {'$and': [{'hidden': False}, {'type': 'file'}]},
    'page': 0,
    'pageSize': 1000,
    'resource': 'items',
    'join': {
        'on': {
            'resource': 'feature_vectors',
            'local': 'entityId',
            'forigen': 'id'
        },
        'filter': {
            'value': {
                '$euclid': {
                    'input': "string || number[], // feature vector ID || actual
```

```
                    '$euclidFilter': {
                        "optional - $eq || $lte || otherSupportedOperators": "num
                    },
                    '$euclidSort': {'eu_dist': 'ascending'}
                }
            },
            'featureSetId': feature_set.id
        },
    }
}
```