## Introduction:

Looking for datasets and features within datasets to connect to models for updating the model can be error prone and can take a lot of time.

Our goal with this model/dataset matching feature is to create automatic recommendations for datasets and features within those datasets for a model or models that the user has already selected (starting first with the case of 1 model, many datasets). Our aim is to reduce the amount of time the user takes to create workflows by reducing errors and reducing time to find and connect datasets.

We will do so by creating matching functionality which takes in knowledge graph, grounding, object(model, dataset) information and perhaps a user search query which will rank datasets and dataset features for each feature in a model. For example, suppose we had an SIR model. For each of susceptible, infected and recovered we would create a ranked list of dataset features which could be suggested to the user or automatically connected to the model. The object information for datasets could include the entire set of dataset or only the datasets currently present in the workflow (for the use case of connecting models to datasets in a workflow).

It is our intent to expose this feature via a few endpoints on the knowledge middleware library. Since the goal of this functionality is to reduce the amount of time needed by a user to create a workflow, this functionality will need to be speedy, no longer than a few seconds depending on performance and exactly functionality performed.

Initially we intend to create ranked matching suggestions (dataset features) for all of the features of a single model (v1). Additional functionality could be implemented in future versions which includes adding a user search term to assist with matching (v2). And finally we could assist the user in matching all models in their workflow in another version of the functionality (v3). We could think of also providing in a future version reasoning for each of the matches to help the user understanding the matching recommendations and hopefully trust or distrust the system recommendations as appropriate.

The version listed above will not cover cases in which a grounding is missing for a model feature. That functionality could be attempted in future versions.

We intend to do this matching initially by either semantic similarity search (embedding the concatenated information about each feature using a text model such as BERT, sentence-transformers, etc..) and comparing the similarity by something like cosine similarity. Other potential solutions include using graph location information directly independently or with semantic embeddings (potentially taking the grounding of each model feature and hopping outward to find groundings that are assigned to dataset features which could then be semantically compared or just ranked on number of hops) or including additional semantic meaning into the task that we are attempting to perform by querying an LLM using information about the task and the retrieved semantically similar text chunks.

## System Overview:

The code for this functionality will be self-contained service, in the form of a docker container which will be surfaced as a few endpoints in the knowledge middleware library. The endpoints will include an endpoint for the different use cases as described in v1,v2,v3 descriptions in the overview, as well as an endpoint for creating embeddings or bulk embeddings of datasets/models.

For a seamless and rapid user experience we should at least store the semantic embeddings of each feature in the backend (except in the case that we are merely linking datasets and models that are already in the workflow, this could be done quickly at the time). These embeddings will be stored in a vector database and will be accessed by the api, which will then compare their similarity. Optionally, these embeddings could be stored in the dkg and could be returned by an api endpoint using a get request, perhaps with certain optional inputs to request the embedding from the knowledge graph.

This api is intended to be called by the frontend from the workflow screen when the user is looking for datasets to add to their workflow. We could imagine this could be triggered upon loading of the workflow screen, placement of the model or indication that the user is looking for a dataset. In v2, the user will be able to interact with the api via a search bar, a request would be then sent to the api upon completion of the search query. In v3, we may also expose this functionality for the purpose of matching all models in a workflow, not just a single model, which could be used similarly as v1 but for all models together in a workflow (not each model independently).

## Functional Requirements:

This functionality will take as input the groundings of each feature, the knowledge graph, object information and perhaps a user query. The quality of the results of the model/dataset matching endpoint will depend on the quality of this information. We will not use a user query until v2.

The endpoint will then

The endpoint will output ranked lists of dataset features for each of the

## Non-Functional Requirements:

To be determined but to include :

Performance specs, security, usability and scalability.

## System Design:

### API Design:

API endpoints will be created as detailed below:

**TO DO**

### Database Design:

The database will contain semantic embeddings as vectors to be stored in a vector database, such as elasticsearch or others. This database will live in a location to be determined by Matt Printz and will be updated by calling one of the endpoints of the api.

### Frontend Design:

To be determined by uncharted, but some ideas are below:

v1:

Model to dataset connections to be represented by translucent or dotted lines between datasets and models. This would allow the user to confirm matching recommendations but could be perhaps more time consuming since the user would have to click each of the lines to confirm and may spend time inspecting the recommendations.

When connecting models to datasets we could give a check, red x or nothing based on whether the model ranked those two features highly in its matching

We could upon creation of the model, loading of the workflow screen, run this endpoint in the background to have embeddings and rankings ready for future use.

We could autopopulate the connections using information from the model dataset linking functionality.

v2:

We could add a search bar that only optionally needs to be filled out somewhere where the user is adding or linking datasets. Pressing enter on the search bar could trigger the endpoint for the model and return a ranked list of datasets with features.

## Implementation:

### Technology Stack: TO DO

## Testing and Evaluation:

### Unit Tests:

Test that semantic similarity of two very similar texts are the same.

Test data ingest and processing (converting api inputs to text chunks)

Test input validation for inputs to the api using positive and negative cases and include cases where data is missing (like no columns)

### Integration Testing:

To be determined by uncharted.

### Performance Testing:

v1 - The speed and matching performance of this functionality will be tested using a several test cases for both use cases (one model, many datasets) (one model, few datasets) taken from the current staging data.

v2 - The speed and matching performance of this functionality will be tested using a several test cases for all use cases (one model, many datasets) (one model, few datasets) with and without search queries.

v3 - The speed and matching performance of this functionality will be tested using a several test cases for all use cases (one model, many datasets) (one model, few datasets) (few models, few datasets) (few models, many datasets) with and without search queries.

## Deployment:

To be determined by Matt Printz (or others).

## Maintenance and Support:

### Monitoring:

We could include but do not intend to include with the initial release, monitoring capabilities which record the performance of the model as dictated by usage of suggestion, decline of suggestions by the user.

### Anticipated future updates:

With new models and other data object changes, we may need to update our model text generation. We will also need to add new model examples to our evaluation dataset.

With changes to the domain knowledge graph we may want to use the knowledge graph more.

## Training and Documentation:

Standard api documentation will be included, as well as some example uses of the api.

We will provide an example notebook to live in the repo

This documentation

A detailed readme outlining usage, rational, evaluation, purpose, etc..

A video explaining the directory and going over the sample notebook