Use Cases

Use Case 1

Customers feel a lot more confident to buy an insurance policy when they feel well-informed. So, as a broker, you can make the customer feel confident by letting them compare different policies and ask questions about them whenever they want.

What we're building

- A chatbot that pulls information from **publicly available policy documents**
- Since we're using publicly available documents, all customers have access

Implementation

Sign in

1. Use the `login` operation

List models

1. Use the `list_models` operation

Create model

1. Create a new retrieval model with publicly available policy documents using the `create_retrieval_model` operation
2. Wait until model training finishes using the `await_train` operation
3. Make the model public using the `set_model_access_level` operation
4. Verify that the model is trained by checking the model list.

Deploy

1. Use the `deploy_model` operation, passing in `<username>/<modelname>` and the token from the sign in step.
2. Wait until deployment is complete using the `await_deploy` operation.
3. Verify that deployment is complete by checking the model list.

Get references

1. Retrieve relevant text chunks from the retrieval model using the `query_retrieval_model` operation. We need the model id, which we can get using the `list_models` operation

Generate answer

1. Make a prompt that combines the query and the references returned by the get references step.
2. Generate a response using the `generate_text_with_openai` operation

Upvote

1. Use the `upvote_reference` operation, passing in the query and the id of the reference as returned by the "get references" step

Associate

1. Use the `associate_keyphrases` operation, passing in any two arbitrary key phrases

---

Use Case 2

Customers often need help to navigate the intricacies of their current policies, or need new ones to replace policies that are about to expire. In the first case, we may want to ask questions about the customer's current insurance policies. In the second, we may want to compare current policies with new insurance policies. In both cases, the customer's private documents are involved, so we want to make sure that 1) other customers don't have access to this information, and 2) there are guardrails in place so that we don't expose sensitive information to generative AI service providers. In addition, since a customer's policies may change over time, we want to incrementally add new documents as they purchase new ones and remove old ones when they expire.

What we're building

- A chatbot that pulls information from a **customer's private documents** as well as **publicly available policy documents**
- Only give access to the broker and the customer
- Apply LLM guardrail to protect sensitive information from generative AI service providers
- The broker must be able to add and remove documents

Implementation

Preliminary

Create an account for a user whose documents will be indexed. This user account will get a special read access.

Sign in, List models, Deploy, Get references, Upvote, Associate

Same as use case 1

Create model

1. Create a new retrieval model using the `create_retrieval_model` operation, this time with the public document model as base model.
2. Wait until model training finishes using the `await_train` operation
3. Give read access to the user account created in the preliminary step using the `set_user_access` operation
4. Verify that the model is trained by checking the model list.

Generate answer

1. Use the `extract` operation to remove sensitive information from the query and references. The operation uses the token classification model we provided, called "PII", to detect sensitive information, then obfuscates it so that it does not leave your machine. You can find the ID associated with the PII model in the "list models" tab.
2. Make a prompt that combines the query and the references returned by the previous step.
3. Generate a response using the `generate_text_with_openai` operation
4. Restore sensitive information with the `restore_pii` operation, passing in the response from step 3 and the inverse map from step 1.

List documents

1. Use the `list_retrieval_model_documents` operation, passing in the model id (refer to the model list) and token from the login operation

Insert document

1. Use the `insert_retrieval_model_documents` operation, passing in the model id, login token, and new documents

Remove document

2. Find the document you want to remove in the document list. Pass the corresponding ID to the `delete_retrieval_model_document` endpoint.