Elemeno AI SDK Documentation

elemeno.ai

Jan 05, 2024

CONTENTS:

CHAPTER

ONE

GETTING STARTED

1.1 Overview

Elemeno AI SDK is the one stop shop for all the elements needed to build your own AI engine.

It includes helpers to use the Elemeno AI operating system, and supports both Elemeno Serverless AI and local installations.

Current features available in the SDK:

- Feature Store Management
- Data Ingestion
  - Big Query Datasource
  - Redshift Datasource
  - Elasticsearch Datasource
  - Pandas DF Datasource
- Training Data Reading
- Inference Data Reading
- ML Frameworks Conversion to ONNX
  - Scikit-learn
  - Tensorflow
  - Pytorch
  - Tensorflow-Lite
- Authentication Utils

1.1.1 First Steps

The first step is to install the SDK module via pip.

```
pip install elemeno-ai-sdk
```

You then run the command :code *mlops init* and follow the steps in the terminal to configure your MLOps environment.

That's all.

(optional) If you intend to leave the configuration files in a location different from the default, set the environment variable below.

```
export ELEMENO_CFG_FILE=<path to config directory>
```

1.1.2 Configuration file schema

A configuration file named elemeno.yaml is expected to be present in the root of the project (or where the variable ELEMENO_CFG_FILE points to).

The file has the following structure:

Table 1: Config File Structure

Field	Type	Example	Description
app	object		The general application configuration
app.mode	string	development	The execution mode, use development for local development and production when doing an oficial run.
cos	object		The S3-like Cloud Object Storage configuration. This is where your artifacts will be persisted. The bucket with name elemeno-cos should exist.
cos.host	string	http://minio.example.com:9000	The host of the cloud object storage server.
cos.key_id	string	AKIAIOSFODNN7EXAMPLE	The access key id for the cloud object storage server.
cos.secret	string	wJalrXUtn-FEMI/K7MDENG/bPxRfiCYEXAMPLEKEY	The secret access key for the cloud object storage server.
cos.use_ssl	boolean	true	Whether to use SSL or not.
cos.bucket_name	string	elemeno-cos	The name of the bucket to store binary files.
registry	object		The model registry configuration. Currently Elemeno supports MLFlow as registry.
registry.tracking_url	string	http://mlflow.tracking.url:80	The MLFlow tracking server url.
feature_store	object		The feature store configuration. Currently Elemeno supports Feast as feature store.
feature_store.feast_config_path	string	.	The path to the Feast configuration file.
feature_store.registry	string	s3://elemeno-cos/example_registry	The path in the cloud object storage to keep the metadata of the feature store.
feature_store.sink	object		The sink configuration. Currently Elemeno supports Redshift and BigQuery as sink.
feature_store.sink.type	string	Redshift	The type of the sink.
feature_store.sink.params	object		The parameters of the sink.
feature_store.sink.params.user	string	elemeno	The user name for the Redshift database.
feature_store.sink.params.password	string	${oc.env:REDSHIFT_PASSWORD}	The password for the Redshift database.
feature_store.sink.params.host	string	cluster.host.on.aws	The host of the Redshift database cluster.

continues on next page

Table 1 – continued from previous page

Field	Type	Example	Description
feature_store.sink.params.port	integer	5439	The port of the Redshift database cluster.
feature_store.sink.params.database	string	elemeno	The name of the Redshift database schema.
feature_store.source	object		The data source configuration. Currently Elemeno supports Elasticsearch, Pandas, Redshift and BigQuery as source.
feature_store.source.type	string	BigQuery	The type of the data source. Valid values are BigQuery, Elastic and Redshift
feature_store.source.params (When using Elastic as source)	object		The parameters of the data source.
feature_store.source.params.host	string	localhost:9200	The host of the Elasticsearch server.
feature_store.source.params.user	string	elemeno	The user name for the Elasticsearch server.
feature_store.source.params.password	string	${oc.env:ELASTIC_PASSWORD,changeme}	The password for the Elasticsearch server.
feature_store.source.params (When using Redshift as source)	object		The parameters of the Redshift data source.
feature_store.source.params.cluster_name	string	elemeno	The name of the Redshift cluster on AWS. When this parameter is specified the SDK uses IAM-based authentication, therefore it's not needed to specify host, port, user and password
feature_store.source.params.user	string	elemeno	The user name for the Redshift database.
feature_store.source.params.password	string	${oc.env:REDSHIFT_PASSWORD,changeme}	The password for the Redshift database.
feature_store.source.params.host	string	cluster.host.on.aws	The host of the Redshift database cluster.
feature_store.source.params.port	integer	5439	The port of the Redshift database cluster.
feature_store.source.params.database	string	elemeno	The name of the Redshift database schema.
feature_store.source.params (When using BigQuery as source)	object		The parameters of the data BigQuery source.
feature_store.source.params.project_id	string	elemeno	The project id of the BigQuery project.

1.1.3 Next Steps

Feature Store

Getting Started

The feature store is a powerful tool for ML practitioners. It abstracts away many of the complexities involved in the data engineering architecture to support both training and inference time.

Through this class, you can interact with Elemeno feature store from your notebooks and applications.

Here is a simple example of how to create a feature table in the feature store:

```
feature_store = FeatureStore(sink_type=IngestionSinkType.REDSHIFT, connection_
↪string=conn_str)
feature_table = FeatureTable("my_test_table", feature_store)

feature_store.ingest_schema(feature_table, "path_to_schema.json")
```

In the above snippet we did a few things that are necessary to start using the feature store.

1. We instantiated the FeatureStore object, specifying which type of sink we want to use. Sinks is the terminology used by the feature store to refer to the different types of data stores that it supports.
2. We instantiated a FeatureTable object, which is a wrapper around the feature store table.
3. We ingested the schema for the feature table using the ingest_schema method.

Ingesting Features

Once you have created a feature table, you can start ingesting features into it. The feature store supports two types of ingestion: batch and streaming (WIP).

Let's imagine you have your own feature engineering pipeline that produces a set of features for a given entity. You can use the feature store to ingest these features into the feature store.

```
# this is a pandas dataframe
df = my_own_feature_engineering_pipeline()

fs.ingest(feature_table, df)
```

That's all that's needed. There are some extra options you can pass to the ingest method, but this is the simplest way to ingest features into the feature store.

Reading Features

Once you have ingested features into the feature store, you can start reading them. The feature store supports two types of reads: batch and online.

The batch read is what you will usually need during training. It allows you to read a set of features for a given entity over a given time range.

```
# the result is a pandas dataframe
training_df = fs.get_training_features(feature_table, date_from="2023-01-01", date_to=
↪"2023-01-31", limit=5000)
```

For the online read, you can use the get_online_features method. This method will return an OnlineResponse object of features for a given entity. This type of object has a to_dict method that can be used to convert the features into a dictionary.

```
entities = [{"user_id": "1234"},{"user_id": "5678"}]

# the result is an OnlineResponse object, with all the features associated with the␣
↪given entities
features = fs.get_online_features(entities)
```

Reference

Feature Store Sink

The concept of a Sink is a way to store the output of a feature.

Reference

Feature Store Source

The concept of a Source is how we read data from a lake of data that not necessary contain ML friendly features.

Reference

CHAPTER

TWO

AUTHENTICATION UTILS

2.1 Overview

Often times, specially when dealing with the first steps of data engineering, you may need to connect to different services in the cloud. We build this module to help you streamline the process of authenticating with some of these services.

2.2 Google Cloud

There are a few ways to authenticate with Google Cloud SDK. The most common, is to use a service account file and specify its location in the environment variable GOOGLE_APPLICATION_CREDENTIALS. However, we understand this type of authentication requires some overhead to be handled in a secure way, specially if you're not in an one-person project.

For development time, you can use API-based authentication tokens through the google appflow package. Hence using service accounts only for production environments.

By using the Authenticator class, you can easily just call authentication, and depending on the existence of a configuration in the elemeno config yaml. The value of the config app.mode is what switches the behavior of what the authenticator class will use. When *development*, it will use appflow (user credentials based) authentication. When *production* it will use the service account file or the API-based authentication tokens specified in GOOGLE_APPLICATION_CREDENTIALS.

```
from elemeno_ai_sdk.datasources.gcp.google_auth import Authenticator
auth = Authenticator()
credentials = auth.get_credentials()
```

The *credentials* variable can then be passed around on google-sdk methods.

In order to configure different values to the authenticator, edit the following section of the file elemeno.yaml

```
...
gcp:
sa:
  file: /tmp/gcp-credentials.json
appflow:
  client_secret:
    file: /tmp/client_secrets.json
  scopes:
    - 'https://www.googleapis.com/auth/bigquery'
...
```

If you need help generating the client_secrets.json file, see Google documentation.

2.3 AWS

For AWS we recommend you use IAM authentication when possible. If you're running your workloads in Elemeno MLOps cloud, there's an option to generate IAM credentials for AWS integration, and then you can use that *arn* to allow necessary permissions on your account.

If using the opensource version of Elemeno, you can use the IAM roles for service accounts approach. Learn more

An easier setup would be to just use the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables. Or even the ~.aws/credentials file.

CHAPTER

THREE

MODEL CONVERSION

3.1 Overview

In order to deploy your ML models you usually need to first serialize them to a format that can be consumed by the ML service. At Elemeno ML Ops, we currently support native deployments of Tensorflow (and TFLite), Pytorch, Scikit-learn and Keras.

However, if you're looking for the maximum performance optimization, we have built a base server in GoLang, that is able to respond inference requests in very few ms of latency.

For users looking to deploy using Elemeno MLOps optimized server you will need to first convert your binary model to the open standard ONNX . Check below the SDK components that will help you on doing a frictionless conversion.

3.2 Reference

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search