**Surround Framework**

**Introduction:**

Surround is a lightweight framework for serving machine learning pipelines in Python.

1. It is intended to be adaptable and simple to utilize.
2. It helps Data scientist by let them concentrate on the current issue instead on the development of code.
3. It is still being under constant and continuous development and expansion.

**Audience:**

The framework of Surround has been primarily designed and developed to assist the Data-scientist who utilizes the python platform to perform their extraction and interpretation of Big-data. The Data-scientists usually spends a lot of time in the process of collection, cleaning and filtering data into datasets. This requires a lot of effort and time consumption. This is where this framework becomes extremely useful, not just because of its user-friendliness, but also it saves hours, the Data-scientist spends on writing the codes.

**Features:**

Surround framework was developed for Artificial intelligence to address the following issues.

1. Continuous restructuring of existing code before implementation. It is intended to improve non-functional attributes of the code.
2. No standardization for the configuration handling, pipeline architecture and building scripts.
3. Failure to provide and end-to-end serving solution between the models.
4. Existing serving approaches don't take into consideration the development of an AI pipeline without re-designing the arrangement.
5. Code was commonly being commented out to run other branches as experimentation was not a first class citizen in the code being written.

**Components:**

The below mentioned components are in this library that can be utilized to build surround solution.

1. Surround
2. Surround Data
3. Stage
4. Runner
5. **Surround:** It is a group of numerous stages or just an initial stage to change raw information into meaningful data. You can set the order of stages directly or by means of a config file. The config file enables you to characterize more than 1 pipeline execution and after that you can switch between them effectively.
6. **Surround Data:** A sharable item between stages that holds vital data for each stage. A phase will read some data from Surround Data, process it, at that point set back new data that will be utilized by different stage(s). When you broaden this class, you can include as many number of variables as you require to enable you to change input data into output data. In any case, there are 4 center factors that are being utilized.

* **stage\_metadata** is information that can be used to identify a stage.
* **execution\_time** is recorded time to complete a process.
* **errors** is information to identify failure of a stage.
* **warnings** is information when transformation is not 100% right.

1. **Stage:** A usage of information change. Here is the place Surround Data is altered to accomplish the outcome that you need. Each stage is just meant to execut out a lot of related actions. First stage can be where you get ready information to be prepared and last stage can be the place your populate information to be sent back to the client.
2. **Runner:** (optional)  An interface to connect Surround to/from data.

**CONFIG**

**RUNNER(S)**

**DATA**

OUTPUT

STAGE

STAGE

SURROUND

INPUT

**Fig(a):** Surround framework components

**Pythonic Interface:**

Surround has a pythonic interface and allows the developer to utilize this framework just as any of the already existed modules. This makes it a powerful framework of data-exploration purposes.

The common example of ‘Hello World” program implementation on surround framework is given below.

from surround import Stage, SurroundData, Surround

import logging

class HelloStage(Stage):

def operate(self, data, config):

data.text = "hello"

class BasicData(SurroundData):

text = None

if \_\_name\_\_ == "\_\_main\_\_":

logging.basicConfig(level=logging.INFO)

surround = Surround([HelloStage()])

output = surround.process(BasicData())

print(output.text)

Let us now understand how the code works;

* The example prints the text “hello” utilizing surround.
* The process consists of defining the operation in the operate() method of HelloStage.
* The object being processed is an instance of BasicData, which inherits from SurroundData.
* he surround object is initialised with only one stage as surround = Surround([HelloStage()], and the line output = surround.process(BasicData()) calls the operate() method of HelloStage.
* New instance BasicData is used as a parameter and finally its contents are printed to the screen.