Создание функции inference

immediate

1 марта 2025 г.

Версия фреймворка 0.3.6

1 Реализовать в новом модуле функцию расчета (inference) для API

Это функция с интерфейсом вида

```
def inference(parameters: list,
              inputs: list,
              output_fields: list,
              model_key: str,
              model: Any = None) -> (list, Any):
```

Пример доступен по ссылке - `https://platform-dev-cs-hse.objectoriented.ru/forgejo/unified-platform-demo/demo-project/src/branch/main/mnist/predict.py`

Интерфейс этой функции соответствует интерфейсу ML-компонента (ссылка на API mlcmp).

Входные параметры:

- *parameters* – список пар [*name, value*], произвольных параметров примитивных типов.
- *inputs* – список входных данных, где каждый элемент – это словарь. Ключи словаря:
  - *name* - название, произвольная строка, используется в коде модуля. Например, *"image"*.
  - *datatype* - тип входных данных, одна из строк *"FP32"*, *"FP64"*, *"INT32"*, *"FILE"*, *"str"*.
  - *content_type* - тип содержимого файла, если тип входной переменной *datatype* - это *"FILE"*.
  - *data* - строка с путём к входному файлу или одномерный список входных данных соответствующего типа.
  - *shape* - размерность входной переменной как массива, применимо даже к файлам (не размер файла в файловой системе).

- *output_fields* - список выходных данных, где каждый элемент - это словарь. Ключи словаря:
  - *name* - название, произвольная строка, используется в коде модуля. Например, *"prediction"*.
  - *datatype* - тип входных данных, одна из строк *"FP32"*, *"FP64"*, *"INT32"*, *"FILE"*, *"str"*.
  - *content_type* - тип содержимого файла, если тип входной переменной *datatype* - это *"FILE"*.
  - *data* - строка с путём к входному файлу или одномерный список входных данных соответствующего типа.
  - *shape* - размерность входной переменной как массива, применимо даже к файлам (не размер файла в файловой системе).
- *model_key* - строка, по которой можно загрузить модель (обычно путь).
- *model* - объект модели, с помощью которого проводится инференс.

Функция выводит tuple из двух объектов:

- список выходных данных, где каждый элемент - это словарь. Ключи словаря:
  - *name* - название, произвольная строка, используется в коде модуля. Например, *"prediction"*.
  - *datatype* - тип выходных данных, одна из строк *"FP32"*, *"FP64"*, *"INT32"*, *"FILE"*, *"str"*.
  - *content_type*- тип содержимого файла, если тип выходной переменной *datatype* - это *"FILE"*.
  - *data* - строка с путём к выходному файлу или одномерный список выходных данных соответствующего типа.
  - *shape* - размерность выходной переменной как массива, применимо даже к файлам (не размер файла в файловой системе).
- объект модели, который можно передать в следующий вызов функции как аргумент *model*.

1.0.1 Написать подробный docstring к этой функции для описания API

В начале docstring нужно описать API в основанном на Sphinx формате.

Пример:

```
def inference(parameters: list,
              inputs: list,
              output_fields: list,
              model_key: str,
```

```
            model: Any = None) -> (list, Any):
        """
    :param inputs.image: Входное изображение;
        Если передано массивом, то порядок индексов - "высота, ширина, канал".
    :type inputs.image: FP32 or image/jpeg or image/png

    :param outputs.predict: Результат прогнозирования
    :type outputs.predict: INT32 or 'FILE' or application/json
        """
        """
          Интегрирует функцию распознавания чисел базового модуля 99
        "Демонстрационный модуль" в сервис запуска моделей через API.

          Модуль принимает входные данные изображений в виде файлов типа PNG и JPG
        и в виде массивов FP32 чисел от 0 до 1 в порядке "высота, ширина, канал".
        Входные данные принимаются как именованный вход "image".
          Модуль возвращает данные как именованный выход "predict" в виде
        файла JSON или массива чисел INT32.
        ... (продолжение)
    """
```

Первая часть с указанием типов должна быть отдельной многострочной строкой, не включающей дальнейшее описание.

Более подробное описание синтаксиса:

```
def example_inference_func(parameters: list,
                           inputs: list,
                           output_fields: set,
                           model_key: str,
                           model: Any = None) -> (list, Any):
    """
    :param inputs.p: Параметр прогнозирования;
        Если вход опциональный, то строка типа завершается ', optional',
        это соответствует синтаксису sphinx rst
    :type inputs.p: FP32, optional

    :param inputs.X: Матрица экземпляров для классификации;
        Если типов несколько, они перечисляются через 'or',
        это соответствует синтаксису sphinx rst
    :type inputs.X: FP64 or FP32 or FILE or text/csv

    :param inputs.abc: Входной параметр;
```

```
    Тип может быть не указан, тогда считается,
    что поддерживаются все допустимые datatype и любые content-type
:type inputs.abc:

:param outputs.predict: Результат прогнозирования
:type outputs.predict: INT32 or 'FILE' or text/csv or application/json or BED

:param outputs.scores: Скоры прогнозирования;
    Для этого выхода datatype 'FILE' будет добавлен,
    потому что указан content-type 'text/csv'
:type outputs.scores: FP32 or FP64 or text/csv

:param outputs.obj: Только объектный выход;
    Для этого выхода тип только объектный
:type outputs.obj: INT32 or str, optional

:param model_parameters.mode: Режим модели;
    Допустимые типы параметров модели - примитивные типы OpenAPI Spec
:type model_parameters.mode: number or integer or float

:param model_parameters.param1: Параметр работы модели
:type model_parameters.param1: string, optional
"""
```

2 Реализовать pytest тесты вызова функции адаптера в папке ./test

Документация - `https://docs.pytest.org/en/8.2.x/`

Общий алгоритм работы с *pytest*:

1. Создать скрипт с названием *test_ *.py* или **_ test.py*.
2. Добавить вызовы адаптера в этот скрипт
3. Сравнивать результат работы адаптера с правильным ответом через стандарный *assert*.
4. Выполнить команду *pytest*.

3 Проверить, что вместо print используется модуль logging для логирования

Документация - `https://docs.python.org/3/library/logging.html`

4 Проверить, что результаты расчетов выводятся в файл или сохраняются в переменные в программе, а не выводятся в print

5 Вынести настройки модуля в отдельный файл .py и проверить, что модуль использует переменные окружения для считывания настроек

Пример такого файла

`https://platform-dev-cs-hse.objectoriented.ru/forgejo/unified-platform-demo/demo-project/src/branch/main/mnist/settings.py`

Результат: Функция-адаптер реализована, тесты ‘pytest‘ выполняются и сохраняют результат в выходные данные или файлы, логи запуска отображаются через библиотеку ‘logging‘.