Фреймворк для AI/ML-проектов

(R&D-версия)

Руководство пользователя Senior Data Scientist

Skoltech

Веб-страница проекта: http://dataskai.com

Содержание

1 Роли пользователей 3

2 Структура базы данных конфигурации и правила управления 4

3 Определение задач 5

4 Инициализация инструментов в Task Loader 8

5 Декорирование с параметрами по умолчанию 9

6 Определение записей функций для задачи 10

7 Определение записей необработанных данных для задачи 12

8 Определение цели и обучающего / тестового наборов для задачи 14

9 Определение метрик для задачи 18

1 Роли пользователей

В настоящем руководстве мы придерживаемся разделения всех дата саентистов (DS далее в этом документе), связанных с проектом проектом, на три группы:

Junior DS

Типичные задачи:

Выполнить исследовательский анализ данных (EDA),

создать и представить работу над моделями для задач DS.

Middle DS

Типичные задачи:

– проверка моделей,

– создание и проверка экстракторов функций,

– развертывание служб DATASKAI.

Senior DS

Типичные задачи:

– управление определениями задач DS,

– создание и проверка моделей предметной области,

– отслеживание результатов задач.

2 Структура базы данных конфигурации и правила управления

Задачи для DS хранятся в отдельной базе данных, которая называется в соответствии с именем проекта и состоит из коллекций (см. табл. 1).

Таблица 1. База данных проекта

Имя коллекции	Состав
feature_loader_configs	конфигурации для компонента Feature Loader
metric_service_configs	конфигурации для компонента Metric Service
raw_data_loader_configs	конфигурации для компонента Raw Data Loader
submits	основная часть сабмитов
submits.chunks	дополнительная часть отправленных файлов, хранящихся в GridFS
submits.files	дополнительная часть отправленных файлов, хранящихся в GridFS
submitter_configs	конфигурации для компонента Submitter
target_loader_configs	конфигурации для компонента Target Loader
tasks_loader_configs	конфигурации для компонента Task Loader

Эти коллекции содержат конфигурации для всех компонентов из Evaluation Tools и сабмитов датасаентистов с ноутбуками и метаданными.

По мере роста проекта DS вам необходимо управлять конфигурациями в этих коллекциях **в соответствии со следующими правилами:**

1) если требуется изменить какой-либо жизнеспособный аспект задачи (обучающие / тестовые наборы, цель, имя задачи, предметный домен и др.), пожалуйста, создайте новую задачу для этого в tasks_loader_configs;

2) если вы добавляете новую конфигурацию в любую из этих коллекций, используйте для нее новое уникальное имя;

3) если вы измените некоторую конфигурацию некритичных компонентов, настоятельно рекомендуется создать копию конфигурации в соответствующей коллекции и переключиться на эту новую конфигурацию в конфигурации загрузчика задач, как только в этом появится необходимость.

4) удаляйте сохраненные конфигурации только в том случае, если вы твердо уверены, что они не потребуются в будущем;

5) не удаляйте конфигурации, которые использовались загрузчиком задач, когда пользователи работают над задачами.

3 Определение задач

Конфигурация для задач хранится в базе данных mongoDB в соответствии с проектом в коллекции с именем tasks_loader_configs. Каждый документ в этой коллекции содержит одно определение задачи DS.

Определение задачи включает:

1) ссылки на конфигурации компонентов, используемых в задаче. Ссылки формируются по именам конфигураций. Вы можете одновременно использовать несколько компонентов из одной категории в одной задаче. Обратите внимание, что в качестве компонента можно использовать другой загрузчик задач;

2) настройки для параметров конструктора заполнения по умолчанию, используемых в инструментах оценки.

Пример конфигурации задачи приведен ниже:

```
{
    "config_name" : "task name",
    "mongo_config" : {
        "host" : "192.168.1.1",
        "port" : 27017,
        "db" : "test_db"
    },
    "description_as_markdown" : {
        "en" : "some markdown task description in english",
        "ru" : "some markdown task description in russian"
    },
    "autofill_parameters" : [
        {
            "function" : "self.submitter.submit_results",
            "parameter_name" : "task_name",
            "parameter_value" : "default task name"
        }
    ],
    "used_config_names" : {
        "submitter.Submitter" : {
            "field_names": ["submitter1", "submitter2", "submitter3"]
            "config_names" : ["submitter_config_name1",
            ↪  "submitter_config_name2", "submitter_config_name3"]
        },
        "target_loader.TargetLoader" : {
            "field_names": ["target_loader1", "target_loader2"]
            "config_names" : ["target_loader_config_name1",
            ↪  "target_loader_config_name2"]
        },
        "feature_loader.FeatureLoader" : {
            "field_names": ["feature_loader"]
            "config_names" : ["feature_loader_config_name"]
        },
```

```
        "raw_data_loader.RawDataLoader" : {
            "field_names": ["raw_data_loader1", "raw_data_loader2"]
            "config_names" : ["raw_data_loader_config_name1",
            ↪  "raw_data_loader_config_name2"]
        }
        "task_loader.TaskLoader" : {
            "field_names": ["task_loader2"]
            "config_names" : ["task_loader_config_name2"]
        }

    }
}
```

Также приведем вариант устаревшей конфигурации, который в ближайшее время не будет поддерживаться. Старый вариант не предоставляет возможности определить несколько компонентов в одной задаче:

```
{
    "config_name" : "task name",
    "mongo_config" : {
        "host" : "192.168.1.1",
        "port" : 27017,
        "db" : "test_db"
    },
    "description_as_markdown" : {
        "en" : "some markdown task description in english",
        "ru" : "some markdown task description in russian"
    },
    "autofill_parameters" : [
        {
            "function" : "self.submitter.submit_results",
            "parameter_name" : "task_name",
            "parameter_value" : "default task name"
        }
    ],
    "used_config_names" : {
        "submitter.Submitter" : {
            "config_name" : "submitter_config_name"
        },
        "target_loader.TargetLoader" : {
            "config_name" : "target_loader_config_name"
        },
        "feature_loader.FeatureLoader" : {
            "config_name" : "feature_loader_config_name"
        },
        "raw_data_loader.RawDataLoader" : {
            "config_name" : "raw_data_loader_config_name"
        }
    }
}
```

Все поля в этом документе обязательны:

config_name – имя конфигурации задачи;

mongo_config – параметры базы данных, в которой хранится конфигурация;

description_as_markdown – описание задач на двух основных языках: английском и русском.

Поле description_as_markdown может содержать шаблоны, которые будут заменены соответствующими значениями из конфигурации задачи json.

Записывайте шаблоны следующим образом:

```
{fieldname.fieldname_1 [0] .fieldname_2}
```

где fieldname_1 – массив. Если имя поля содержит точку (.), она должна быть экранирована знаком «обратный слеш», например:

```
{field \ .name.fieldname_1}
```

autofill_parameters – параметры по умолчанию для заполнения используемых инструментов;

used_config_names – инструменты и соответствующие имена конфигураций, которые будут использоваться инструментом Task Loader при загрузке других инструментов.

Эта конфигурация будет интерпретироваться и обрабатываться в два этапа:

1) будет инициализирован каждый инструмент, указанный в used_config_names (см. ниже); 2) некоторые методы / функции в загруженных инструментах будут с установленными по умолчанию значениями параметров.

4 Инициализация инструментов в Task Loader

Поле used_config_names заполнено инструментами, используемыми для анализа данных в текущей задаче. Имена документов соответствуют module_name.tool_class_name из Evaluation Tools, а значение документа – параметрам kwargs, которые передаются ему при инициализации загрузчика задач. Таким образом, поместите объект результата в объект Task Loader в поле, названном так же как модуль, из которого он был взят.

Например: если вы возьмете конфигурацию, указанную выше, загрузчик задач интерпретирует первый документ из used_config_names следующим образом:

— чтобы загрузить задачу, сначала требуется запустить все инструменты в ней, проинициализировав инструмент submitter.Submitter;

— импортировать модуль submitter;

— импортировать из него класс Submitter;

— построить объект этого класса с помощью kwargs

```
{"config\_name": "submitter\_config\_name"};
```

— поместить объект результата в поле submitter объекта Task Loader;

— повторить для всех документов, перечисленных в used_config_names

Внимание

Гарантированный порядок инициализации инструментов загрузчиком задач отсутствует.

5 Декорирование с параметрами по умолчанию

Для каждого документа, указанного в autofill_parameters, загрузчик задач возьмет метод / функцию, перечисленные в поле функции, и дополнит их значениями по умолчанию (используя partial и docstring, перенесенные в новую функцию).

Полученная декорированная функция точно такая же, но с одним заполненным параметром, отсутствующим в сигнатуре функции (по причине того, что он уже заполнен).

Этот метод на самом деле открывает способ реализовать множество функций с помощью Task Loader, например, вы можете поместить некоторые определенные инструменты в проект git и сразу же запустить их, чтобы они были доступны через объект Task Loader.

Внимание

Заполнение параметров выполняется в точном порядке, указанном в autofill_parameters.

6 Определение записей функций для задачи

Конфигурация MongoDB для задач машинного обучения включает ссылку на конфигурацию FeatureLoader (вместе с конфигурациями для RawDataLoader, TargetLoader и Submitter).

FeatureLoader позволяет получать функции (входные переменные для алгоритмов машинного обучения) из хранилища функций. Он также активно использует типы для данных: приведение типов применяется к данным, когда FeatureLoader возвращает объект pandas.DataFrame.

Пример коллекции MongoDB для использования FeatureLoader (с сокращениями):

```
{
    "config_name" : "aero__fw_classification_v1_features",
    "local_temp_dir" : "/tmp",
    "feature_manager_config" : {
        "feature_records" : [
            {
                "name" : "cur_val__aids_rep_04",
                "features_type" : "network_api",
                "info" : "AIDS report 01.",
                "feature_loader_args" : {
                    "files" : [
                        "http://10.30.16.181:8290/api/feature_builder/export?
                        ↪ build=1568896505205"
                    ],
                    "dtypes" : {
                        "cur_val__aids_rep_04__alt" : "np.float32",
                        "cur_val__aids_rep_04__egt_1" : "np.float32",
                        "cur_val__aids_rep_04__egt_2" : "np.float32",
                        "cur_val__aids_rep_04__flight_phase" : "str"
                    },
                    "default_dtype" : "np.float32",
                    "index_columns" : [
                        "AC",
                        "CYCLE_END_TS"
                    ],
                    "sep" : ","
                },
                "features_full_list" : [
                    [
                        "cur_val__aids_rep_04__alt",
                        "cur_val__aids_rep_04__egt_1",
                        "cur_val__aids_rep_04__egt_2",
                        "cur_val__aids_rep_04__flight_phase"
                    ]
                ]
            },
```

```
        ]
    }
}
```

```
In [8]:
feature_loader = task_loader.feature_loader
df = feature_loader.load_features(feature_selector=lambda x: 'cur_val__aids_rep_04' in x)
display(df.sample(5))
print(df.dtypes)
```

AC	CYCLE_END_TS	cur_val__aids_rep_04__alt	cur_val__aids_rep_04__egt_1	cur_val__aids_rep_04__egt_2	cur_val__aids_rep_04__flight_phase
VP-BTN	1453650660	2701.0	7310.0	7700.0	5
VP-BHG	1367005500	1751.0	7050.0	7220.0	5
VP-BHP	1401245460	1804.0	7170.0	7220.0	5
VP-BHL	1384588440	1923.0	6510.0	6560.0	5
VP-BTX	1445611080	1912.0	6710.0	6940.0	5

```
cur_val__aids_rep_04__alt            float32
cur_val__aids_rep_04__egt_1          float32
cur_val__aids_rep_04__egt_2          float32
cur_val__aids_rep_04__flight_phase    object
dtype: object
```

Рис. 1. Результирующий фрейм данных с функциями

Ключи первого уровня:

config_name, будет использоваться в конфигурации TaskLoader;

local_temp_dir, путь к каталогу для хранения временных файлов пользователей, например, для кеширования ответов;

feature_manager_config, хранит feature_records со списком различных записей функций.

Каждая запись функции (feature_records) должна включать:

— name записи признаков;

— features_type, указывает, используется ли локальный или сетевой исполнитель (local_disk_config или network_api);

— info, текстовое описание того, о чем эта запись;

— feature_loader_args:

– files, путь или ссылка на файл с данными (Feature Storage);

– dtypes, словарь с именами функций в качестве ключей и типами данных в качестве значений;

– default_dtype, используется для функций, которые не указаны в dtypes dict;

– index_columns, список имен столбцов индекса для Pandas;

– sep, разделитель, если используется для чтения содержимого файла, обычно для файлов, разделенных запятыми или табуляцией;

— features_full_list (необязательно), если он указан, то можно получить все имена функций и их типы без загрузки всех данных функций.

7 Определение записей необработанных данных для задачи

RawDataLoader позволяет получать необработанные данные для исследовательского анализа данных. Структура коллекции MongoDB для RawDataLoader идентична структуре для FeatureLoader (см. раздел «Определение записей функций для задачи») и имеет те же ключи и значения. Применение немного отличается:

— одновременно можно загрузить только одну запись сырых данных;

— files указывает на Data Service;

— имена функций не обязательно должны соответствовать соглашениям об именах функций;

— index_columns всегда содержит пустой список для необработанных данных.

```
{
    "config_name" : "aero__raw_data",
    "local_temp_dir" : "/tmp",
    "feature_manager_config" : {
        "feature_records" : [
            {
                "name" : "flights",
                "features_type" : "network_api",
                "info" : "Flight schedule.",
                "feature_loader_args" : {
                    "files" : [
                        "http://10.30.16.181:8190/datasets/aerophm/flights/da⌋
                        ↪  ta?namespace=aerophm&dataset=flights"
                    ],
                    "dtypes" : {

                        "aircraft_id" : "str",
                        "airport_from" : "str",
                        "airport_to" : "str",
                        "actual_departure_time" : "np.int64",
                        "actual_arrival_time" : "np.int64"
                    },
                    "default_dtype" : "str",
                    "index_columns" : [],
                    "sep" : "\t"
                },
                "features_full_list" : [
                    [
                        "aircraft_id",
                        "airport_from",
                        "airport_to",
                        "actual_departure_time",
                        "actual_arrival_time"
```

```
                    ]
                ]
            },
        ]
    }
}
```

In [6]:

```
raw_data_loader = task_loader.raw_data_loader
df = raw_data_loader.load_data_one_record(record_to_use='flights')
display(df.head())
print(df.dtypes)
```

	aircraft_id	airport_from	airport_to	actual_departure_time	actual_arrival_time
0	VP-BTU	OMS	DME	1322698260	1322709600
1	VP-BTN	TJM	DME	1322699160	1322707560
2	VP-BTW	KJA	DME	1322700060	1322716860
3	VP-BHV	SVX	DME	1322700120	1322707800
4	VP-BHL	NOZ	DME	1322700240	1322716560

```
aircraft_id              object
airport_from             object
airport_to               object
actual_departure_time     int64
actual_arrival_time       int64
dtype: object
```

Рис. 2. Результирующий фрейм с необработанными данными

8 Определение цели и обучающего / тестового наборов для задачи

Определение цели вместе с определением обучающего / тестовового наборов – одна из самых важных частей в настройке задачи.

Определение цели выполняется с помощью следующих основных шагов:

— программирование предметной модели;

— настройка компонента Target Loader для использования определенного поля предметной модели.

Подробное описание процесса программирования предметной модели будет предоставлено в будущих версиях DATASKAI, в текущей версии руководства перечислим основные шаги, которые нужно реализовать:

1) в git проекта создайте подкаталог внутри ./modules/subject_domain с именем новой модели;

2) решите, какое поле из какого объекта будет вашим целевым полем;

3) реализуйте объекты и отношения между ними в Python с помощью ООП;

4) создайте фабрику с методом создания целевого объекта. Метод должен принимать kwargs, которые станут индексами для будущих наборов данных;

5) при инициализации фабрика должна использовать данные, загруженные с помощью инструмента Raw Data Loader, для заполнения содержимого объектов.

Результатом создания предметной модели должна стать фабрика, которая инициализирует объекты предметной области с целевым полем. Вы должны иметь возможность инициализировать и запускать эту фабрику самостоятельно в своем jupyter notebook и воспроизводить создание объектов с помощью метода построения и kwargs с индексами.

Наборы для обучения / тестирования определяются через конфигурацию компонента Target Loader. Простая конфигурация целевого загрузчика приведена ниже:

```
{
    'config_name' : 'terminator_mass_prediction_v1',
    'config_version' : '0.1',
    'use_mongo_cache' : true,
    'cache_mongodb_address' : '10.30.16.181:27017',

    'config_parameters':
        {
            'subject_model':
                {'directory_path':
                 ↪  "../../modules/subject_domain/terminators_v1/",},
```

```
            'central_object':
                {
                    'module_and_class_name':'t_600.T600',
                    'id_fields':['robot_index',],
                    'constructor_fields':[''],
                    'target_fields':['robot_mass',],
                    'use_factory':True,

                },
            'factory_for_central_object':
                {
                    'factory_module_and_class_name':
                    ↪  't_factory.TerminatorFactory',
                    'factory_default_init_kwargs' : {
                                    'using_normalization_mass' : 21,
                                    'construct_only_target' : true
                                },
                    'factory_method_for_central_object':'construct_terminator
                    ↪  ',
                    'factory_method_fields':['t_prod_index',],

                },
            'validation_type':'train_test',
            'folds':[[(1,),(2,),(3,)],[(4,),(5,),(6,)]]
        },
    'override_parameters':
        {
            'tmp_dir':'/tmp',
            'subject_area_dir':'./',
        }
}
```

Укажем назначение всех **параметров первого уровня** из перечисленного выше:

— config_name – строка, название конфигурации;

— config_version – строка, версия конфигурации;

— use_mongo_cache – логический тип данных, настройка для сохранения результатов тренировочных / тестовых индексов и соответствующих целевых значений в mongodb. Должно быть установлено значение true, чтобы компонент Metric Service вычислял метрики;

— cache_mongodb_address – адрес кеша mongodb;

— config_parameters – документ с основными настройками для компонента Target Loader;

— override_parameters – внутренние поля компонента Target Loader, которые необходимо перезаписать.

Параметры, необходимые в первую очередь, сосредоточены внутри документа config_parameters:

— subject_model – описание кода предметной модели:

– directory_path – путь к коду предметной модели;

— central_object – документ с описанием объекта, используемого в качестве целевого источника для задачи:

– module_and_class_name – строка, модуль и класс объекта, который будет использоваться для построения цели;

– id_fields – список строк, полей, используемых в качестве полей индекса в случае, если построение объекта возможно без фабрики;

– constructor_fields – список строк, полей конструктора, используемых в случае, если построение объекта возможно без фабрики;

– target_fields – список строк, полей для извлечения из объекта и использования в качестве цели задачи;

– use_factory – логический тип данных, определяющий, использовать ли фабрику при построении объекта;

— factory_for_central_object – документ с описанием фабрики, использованной для получения объекта:

– factory_module_and_class_name – строка, имя модуля и класса, используемого как фабрика;

– factory_default_init_kwargs – kwargs, который передается на завод при инициализации фабрики;

– factory_method_for_central_object – строка, метод, используемый для построения объекта предметной области;

– factory_method_fields – список строк, имена аргументов, передаваемых в метод построения объектов;

— validation_type – строка, тип проверки, примененной к задаче (в настоящее время поддерживается только «tran_test»);

— folds – список, содержит списки с наборами аргументов train и tests для инициализации объектов для train и test соответственно.

Обратите внимание на поле складок в документе config_parameters, это место, которое нужно заполнить, чтобы исправить набор поездов / тестов для задачи.

Поскольку конфигурация для компонента Task Loader написана, вы можете связать его с компонентом Task Loader по имени и запустить Task Loader. При первом запуске компонент Target Loader будет создавать все объекты, указанные в наборе для обучения / тестирования, и извлекать целевые поля. По завершении этого процесса индексы и соответствующие значения будут помещены в базу данных кэша, поэтому следующий вызов Target Loader будет намного быстрее и не будет включать создание объектов предметной области.

9 Определение метрик для задачи

Метрики проекта хранятся в подкаталоге ./modules/metrics. На данный момент эти модули должны содержать функцию, которая принимает два фрейма данных pandas и выводит скаляр. Эта функция будет использоваться Metric Service для создания и сохранения результатов на тестовом наборе текущей задачи. Указанные функции, в свою очередь, могут использовать библиотеки, доступные из контейнера DS, поскольку почти каждая зависимость из контейнера DS переносится в контейнер сервиса Metric Service.

В настоящее время метрики определены в исходном коде Metrics Service, поэтому для их изменения вам необходимо добавить его в контейнер при сборке. Модули с метриками следует перенести в файл ./modules/metrics_provider.py и реализовать функцию provide_metrics.

Эта функция должна возвращать два списка, первый – названия метрик, второй – функции метрик.

Чтобы добавить новые метрики в список метрик, который возвращается с помощью provide_metrics, вы должны зарегистрировать функцию метрик в объекте METRIC_STORAGE с помощью метода register или декоратора $@metric(name ='')$ из ./modules/metrics/metrics_provider_tools.py (см. Metrics Service API).

Чтобы применить разные метрики для разных задач, вы должны отредактировать файлы конфигурации в metric_service / configs и указать, какие метрики должны использоваться в поле tasks_to_use_metrics (для значения «null» будут применяться все доступные метрики):

```
"tasks_to_use_metrics": {
        "some_task1": ["roc", "sss"],
        "some_task2": null,
        "some_task3": ["max_f1", "mae"]
    }
```

Вы можете заранее определить шаблоны для повторного использования списков метрик в задачах:

```
"tasks_templates":
{
    "template1":
    {
        "metrics": ["roc", "sss"]
    },
    "template2":
    {
        "metrics": null
```

```
    },
    "template3":
    {
        "metrics": ["max_f1", "mae"]
    }
},
"tasks_to_use_metrics": {
    "some_task1": {"task_template":"template1"},
    "some_task2": {"task_template":"template2"},
    "some_task3": {
      "task_template":"template3",
      "metrics": ["roc_custom"]}
},
```

Имена метрик используются сервисом Metrics Service для хранения соответствующих значений в коллекции submits. Пример такого реализованного интерфейса показан ниже:

```
from sklearn.metrics import precision_recall_curve
from modules.metrics.metrics_provider_tools import metric, METRIC_STORAGE


@metric(name="max_f1")
def max_f1_metric(true_result_df, predicted_result_df):
    max_f1_value = None
    precision, recall, thresholds =
    ↪ precision_recall_curve(true_result_df['TARGET'],
    ↪ predicted_result_df['TARGET'])

    for pr, rec, in zip(precision, recall):
        cur_max_f1 = 2 * pr * rec / (pr + rec)

        if max_f1_value is None:
            max_f1_value = cur_max_f1

        if cur_max_f1> max_f1_value:
            max_f1_value = cur_max_f1

    return max_f1_value

def provide_metrics(task_name=None):
    return METRIC_STORAGE.list_metrics(task_name)
```

Метод METRIC_STORAGE.list_metrics(task_name) используется для перечисления имен и функций метрик в двух разных списках.

Внимание

Интерфейс функции метрики использует фрейм данных Pandas для истинных и предсказанных тестовых ответов, полученных от пользователя.

Функция метрики может использовать дополнительную информацию из индексов для расчета метрик. Например, вы можете указать некоторые прогнозы для важного случая с большим весом, чем для других.

В будущих версиях определение задачи будет содержать подробную конфигурацию используемых метрик из проекта git, включая хэш git и ветку из репозитория git для каждой задачи.