Правила написания метрик

Метрики Predicate - это класс Python с определенной структурой. Данный класс не содержит специфичных зависимостей, кроме `pandas`.

Примеры кода метрики доступны на отдельной странице.

На данной странице описаны:

1. Структура класса с метрикой
2. Рекомендации по написанию метрик

Структура класса с метрикой

У класса метрики есть набор обязательных полей и методов. При этом нет ограничений на использование дополнительных полей или методов. Мы даже посоветуем использовать дополнительные методы и поля в разделе "Рекомендации по написанию метрик".

Обязательные поля:

1. `__desc__`;
2. `__tags__`;
3. `is_scalar`;
4. `is_signal`;

Обязательные методы:

1. `__init__`;
2. `__call__`;
3. `save`;
4. `scalar` (если флаг `is_scalar = True`);
5. `signal` (если флаг `is_signal = True`).

В разделах ниже поля и методы описаны подробнее.

Поля

Название класса становится названием метрики в системе

`__desc__` - понятное человеку описание метрики

`__tags__` - список тегов, присвоенных метрике в системе

`is_scalar` - флаг, что метрика имеет один скалярный результат. Если `is_scalar = True`, то метрика обязана содержать метод `scalar`

`is_signal` - флаг, что метрика имеет светофор. Если `is_signal = True`, то метрика обязана содержать метод `signal`

Таким образом, класс метрики "Тест Колмогорова-Смирнова", которая содержит и график, и скалярное значение, и светофор, должен начинаться следующим образом:

```
class r_2_5_KS_on_scale:
    __desc__ = "KS-test on scale. Тест Колмогорова-Смирнова"
    __tags__ = ["risk", "scalar"]

    is_scalar = True
    is_signal = True
```

Методы

__init__

Метод инициализации класса предназначен для ввода параметров метрики. Именно в этот метод будут передаваться все параметры метрики при вызове метрики в проекте Predicate или в рамках библиотеки.

Базовая сигнатура:

```
def __init__(self, *, **kwargs: typing.Any) -> None:
    ...
```

Определение метода `__init__` в классе метрики должно соответствовать следующим правилам:

1. В метод должен быть передан хотя бы один датасет.
2. Каждый параметр должен иметь аннотацию его типа согласно правилам параметризации метрик.
3. Значение параметра по умолчанию не должно нарушать логику валидации указанного типа параметра.
4. Передавать значения параметров в следующие методы нужно через атрибуты экземпляра класса `self`.

>>> **Типы параметров метрик и их использование** <<<

Рекомендуем проводить все проверки введённых значений в рамках данного метода

Пример для метрики "Тест Колмогорова-Смирнова":

```
def __init__(
        self,
        df: pd.DataFrame,
        scale_column: str,
        target_column: str,
        threshold_yellow: float = 10,
        threshold_red: float = 30,
    ):
        self.scale_column = scale_column
        self.target_column = target_column
        self.df = df.astype({self.target_column: "float"})
        self.threshold_yellow = threshold_yellow
        self.threshold_red = threshold_red

        if self.df.empty:
            raise Exception("Dataframe is empty")
        if self.target_column not in self.df:
            raise ValueError(f"Field {self.target_column} does not exist in
the dataframe")
        if self.scale_column not in self.df:
            raise ValueError(f"Field {self.scale_column} does not exist in the
dataframe")
        if self.df[self.scale_column].nunique() > 100:
            raise Exception("Ошибка: переменная scale не является
категориальной")
```

Таким образом, "Тест Колмогорова-Смирнова" принимает на вход:

- `pandas` датафрэйм с данными для расчёта;
- имя колонки со шкалой;
- имя колонки с целевой переменной;
- желтая и красная границы светофоров.

У границ светофора в данном примере есть значение по умолчанию.

В теле метода экземпляру метрики (`self`) присваиваются значения параметров для использования в остальных методах, и проводятся проверки правильности введенных данных.

> ***__call__***

Метод вызова класса содержит основные расчёты метрики. Обычно в нем расположен блок составления графиков метрики, но ограничений на содержимое нет, метод может содержать любые расчёты

Базовая сигнатура:

```
def __call__(self) -> None:
    ...
```

Метод не имеет параметров, кроме `self` и ничего не возвращает. Все рассчитанные величины должны быть сохранены в атрибуты экземпляра класса `self`.

Пример для метрики "Тест Колмогорова-Смирнова":

```
def __call__(self) -> None:
        dataset = self.df.loc[:, [self.target_column,
self.scale_column]].dropna()

        # номер разряда рейтинговой шкалы (способ получения зависит от формата
данных в столбце self.scale)
        dataset["bin_number"] = dataset[self.scale_column].map(
            lambda x: int(x.split("_")[-1])
        )  # dataset[self.scale].astype('category').cat.codes#

        dataset = dataset.sort_values(by=["bin_number"], ascending=False)

        good_cnt = dataset[dataset[self.target_column] == 0].shape[0]
        bad_cnt = dataset[dataset[self.target_column] == 1].shape[0]

        gr_bad = (
            pd.DataFrame(
                dataset.groupby("bin_number", observed=False)
[self.target_column].sum()
            ).cumsum()
            / bad_cnt
        )
        dataset["target_inverse"] = np.where(dataset[self.target_column] == 1,
0, 1)
        gr_good = (
            pd.DataFrame(
                dataset.groupby("bin_number", observed=False)
["target_inverse"].sum()
            ).cumsum()
            / good_cnt
        )

        ks_calc_temp = pd.merge(gr_good, gr_bad, how="left", left_index=True,
right_index=True)
        ks_calc_temp["diff"] = 0
        ks_calc_temp["diff"] = abs(
            ks_calc_temp.iloc[:, 0:1].values - ks_calc_temp.iloc[:,
1:2].values
        )

        self.scalar_value = 100 * ks_calc_temp["diff"].max()
        ks_result = round(self.scalar_value, 2)
        result_idx = ks_calc_temp["diff"].argmax()

        x_value1 = gr_bad.index.values.tolist()
```

```
        y_value1 = gr_bad[self.target_column].values.astype("float").tolist()
        x_value2 = gr_good.index.values.tolist()
        y_value2 = gr_good["target_inverse"].values.astype("float").tolist()
        x_value3 = [
            float(gr_bad.index.values[result_idx]),
            float(gr_bad.index.values[result_idx]),
        ]
        y_value3 = [
            float(gr_bad[self.target_column].values[result_idx]),
            float(gr_good["target_inverse"].values[result_idx]),
        ]

        line1 = go.Scatter(
            mode="lines",
            x=x_value1,
            y=y_value1,
            name="bad",
            line={"width": 3},
            marker={"color": "#63666A"},
        )
        line2 = go.Scatter(
            mode="lines",
            x=x_value2,
            y=y_value2,
            name="good",
            line={"width": 3},
            marker={"color": "#3eb489"},
        )
        line3 = go.Scatter(
            mode="lines",
            x=x_value3,
            y=y_value3,
            name=f"KS-statistic = {ks_result.astype('float')}",
            marker={"color": "black"},
        )
        self.fig = go.Figure(data=[line1, line2, line3])
        self.fig.layout = self.custom_layout()
```

Здесь для метрики "Тест Колмогорова-Смирнова" были рассчитаны следующие параметры:

- `self.scalar_value` - скалярное значение для данной метрики;
- `self.fig` - Plotly-график метрики.

Если метрика возвращает исключительно скалярные значения, допустимо пропустить описание данного метода. Так для метрики "Максимальное значение":

```
def __call__(self) -> None:
        pass
```

> ***save***

Метод `save` предназначен для сохранения всех графиков в файлы для дальнейшего вывода в отчетах.

Базовая сигнатура:

```
def save(self, output_dir: str) -> dict[str, str] | None:
    ...
```

Метод принимает на вход путь до папки сохранения в строковой переменной `output_dir`. Метод возвращает словарь значений <алиас графика>: <путь до файла>

Predicate умеет обрабатывать данные только следующих форматов:

1. HTML;
2. PNG;
3. JPG;
4. SVG.

Вы **можете** сохранить результаты работы метрики в другом формате, но вам необходимо в этом же методе составить HTML-файл и встроить в него ссылку на ваш файл в неизвестном для Predicate формате. Так система сможет отобразить его в отчете.

Пример метода `save` для метрики "Тест Колмогорова-Смирнова":

```
def save(self, output_dir: str) -> dict[str, str] | None:
        self.fig.write_html(
            f"{output_dir}/data.html",
            config={"displaylogo": False},  # remove the plotly logo
        )
        return {f"scale_{self.scale_column}": f"{output_dir}/data.html"}
```

В данном случае был сохранен один график из атрибута `self.fig`. Но также можно сохранить множество графиков, или сохранять один или множество графиков по условию, как в метрике "Плотность распределения":

```
def save(self, output_dir: str) -> dict[str, str] | None:
        if self.split_charts:
            result = {}
            for column_name, fig in self.figs.items():
                file_path = f"{output_dir}/data_{column_name}.html"
                fig.write_html(
                    file_path,
                    config={"displaylogo": False},  # remove the plotly logo
                )

                result[column_name] = file_path
            return result
        else:
```

```
            self.fig.write_html(
                f"{output_dir}/data.html",
                config={"displaylogo": False},  # remove the plotly logo
            )

            return {"fig_name": f"{output_dir}/data.html"}
```

Если метрика не возвращает ни одного графика, допустимо пропустить описание данного метода. Так для метрики "Максимальное значение":

```
def save(self, output_dir: str) -> dict[str, str] | None:
        pass
```

> ***scalar***

Данный метод должен появиться в классе метрики в том случае, если флаг `is_scalar = True`.

Метод предназначен для вычисления числового значения метрики.

Базовая сигнатура:

```
def scalar(self) -> int | float:
    ...
```

Метод не принимает на вход никакие параметры. Метод возвращает числовое значение в виде `int` или `float`.

Для нескалярных метрик (флаг `is_scalar = False`) этот метод определять не нужно. Если определить, Predicate воспримет это как ошибку.

Пример метода `scalar` для метрики "Тест Колмогорова-Смирнова":

```
def scalar(self) -> int | float:
        return self.scalar_value
```

В данном случае скалярное значение было рассчитано заранее, и его нужно было только вернуть из метода.

Для метрик без графиков характерно внесение логики расчета числового значения в метод `scalar`. Так для метрики "Максимальное значение":

```
def scalar(self) -> int | float:
        self.scalar_value = float(self.df[self.field_column].min(skipna=True))

        return self.scalar_value
```

> ***signal***

Данный метод должен появиться в классе метрики в том случае, если флаг `is_signal = True`.

Метод предназначен для вычисления светофора метрики.

Светофор - это индикатор качества метрики. Он выражается в трех цветах:

- Зеленый (`green`) - значения метрики в пределах нормы.
- Желтый (`yellow`) - значения метрики вызывают опасения.
- Красный (`red`) - значения метрики далеко за пределами нормы.

Базовая сигнатура:

```
def signal(self) -> Literal["red", "yellow", "green"]:
    ...
```

Метод не принимает на вход никакие параметры. Метод возвращает строковое значение светофора. Значения светофора должны быть строго из списка `["red", "yellow", "green"]`.

Пример метода `signal` для метрики "Тест Колмогорова-Смирнова":

```
def signal(self) -> Literal["red", "yellow", "green"]:
        signal_light = "green"

        if self.scalar_value > self.threshold_red:
            signal_light = "red"
        elif self.scalar_value > self.threshold_yellow:
            signal_light = "yellow"

        return signal_light
```

Порядок исполнения методов

Методы и в продукте Predicate, и в библиотеке Predicate исполняются в следующем порядке:

1. `__init__`;
2. `__call__`;
3. `save`;
4. `scalar` (если флаг `is_scalar = True`);
5. `signal` (если флаг `is_signal = True`).

Учитывайте порядок исполнения методов при разработке метрики. От него зависит, как можно передавать значения из метода в метод.

Рекомендации по написанию метрик

Кроме явных требований к структуре класса метрики, есть ряд необязательных рекомендаций к их написанию. Эти рекомендации упрощают работу с метрикой: ее поддержку разными членами команды; создание множества метрик по аналогичному шаблону; проверку данных, введенных в метрику.

Список рекомендаций не конечный. Мы будем увеличивать его при появлении новых идей.

Метод `custom_layout`

Если ваша метрика возвращает график в виде объекта **Plotly**, то мы рекомендуем создавать метод `custom_layout`.

Layout - это макета графика. В layout описываются параметры представления графика: заголовок; названия осей; границы отображаемых осей; и можество других параметров. Подробнее об этом - в документации Plotly.

Layout обычно занимает от десяти строчек кода. Метод `custom_layout` позволяет вам не загромождать метод `__call__` лишними описаниями, а вынести их в отдельный метод.

Метод `custom_layout` упрощает копирование кода из метрики в метрику. В вашем описании графика Plotly будет фигурировать только обращение к методу `custom_layout`, поэтому вы сможете перенести этот график в другую метрику с другим layout-ом без дополнительного переписывания кода.

Пример метода `custom_layout` для метрики "Тест Колмогорова-Смирнова":

```
def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": {"text": "<b>Тест Колмогорова-Смирнова</b>", "x": 0.1,
"y": 0.97},
            "legend": {"yanchor": "bottom", "y": 0.05, "xanchor": "right",
"x": 1},
            "yaxis": {"title": "Кумулятивная доля", "side": "left"},
            "xaxis": {
                "title": "Разряд рейтинговой шкалы",
                "side": "left",
                "type": "category",
                "domain": [0, 0.8],
            },
            "margin": {"t": 35, "b": 5, "l": 5, "r": 5},
        }
```

Поле `scalar_value`

Мы рекомендуем присваивать числовое значение метрики в атрибут `scalar_value`. Это поле упрощает жизнь: становится проще читать код; всегда понятно, что возвращать в методе `scalar`; легко обратиться к числовому значению в других методах.