1)Описание задачи

Реализуйте отбор фич (c sklearn-like API) с помощью генетических алгоритмов

Возможно, вам будет полезно воспользоваться библиотекой

- https://deap.readthedocs.io/en/master/

master/

2) Rotation Tree Ensembles: Forest

Описание задачи

В этом задании предлагается собственными руками сделать имплементацию в стиле библиотеки sklearn модель Rotation Forest для задачи классификации.

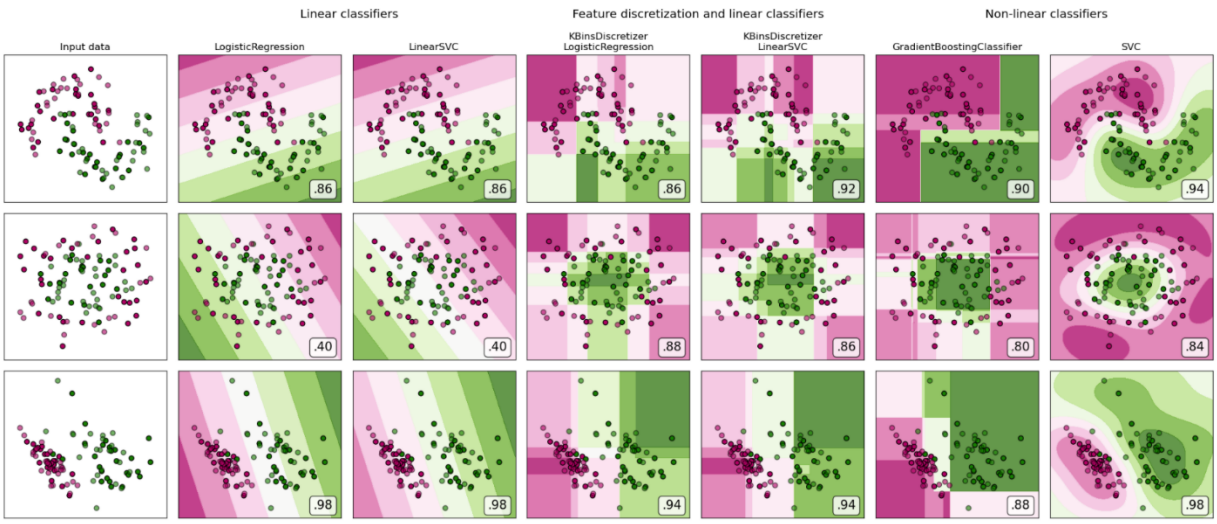
Требования к задаче

Будет необходимо написать класс RotationForestClassifier у которого есть 3 обязательных метода

```
# Конструктор
def __init__(
    self,
    something: # ваши параметры
    something: # ваши параметры
    something: # ваши параметры
    something: # ваши параметры
):
# Обучение
def fit(
    self,
    X: np.ndarray, # массив (n * k) с признаками.
    y: np.ndarray # массив (n) с целевой переменной.
) -> None:
    # fit the model
# Применение
def predict(self, X: np.ndarray) -> Iterable[float]:
    # compute predictions
    return predictions
```

Дополнительная задача:

Выведите разделительную плоскость как на графиках ниже.



Полезные ссылки:

ссылка на [feature discretization]

[Random Rotation Ensembles]

3) sMAPE

Существует множество метрик для задач регрессии, например, для задач прогноза помимо всем известных RMSE, RMSLE и MAE используют MAPE, WAPE, sMAPE и другие

MAPE (Mean Absolute Percentage Error) – метрика ошибки, учитывающая средний процент отклонения предсказываемого значение от реального

MAPE могут сильно портить случаи, когда мы плохо предсказываем маленькие значения: например, в основном значения порядка 100-1000, но в какой-то из дней факт был 0.5, а мы предсказали 50, получим ошибку 9900%, что уже существенно исказит агрегированную метрику по всем объектам. Если бы факт был 0.2, получили бы уже 24900%.

sMAPE (от symmetric MAPE) – логическое продолжение MAPE, где в знаменателе стоит уже сумма модулей предсказания и факта, а не только факта

Нетрудно проверить, что в предыдущих случаях значения оказываются уже не такими страшными:

- 0.5 и 50 → 98,0%
- 0.2 и 50 → 99,2%

Два месяца назад ваш коллега во время своей стажировки написал sMAPE, который корректно работал в проде всё это время, однако периодически метрика выдаёт ошибку и к сожалению логи ошибки нам недоступны.

Код метрики:

```
import numpy as np



def smape(x: np.array, y: np.array) -> np.float:
    return np.mean(2 * np.abs(x - y) / (np.abs(x) + np.abs(y)))
```

Ваша задача – отыскать случаи, когда данный код выдаёт ошибку, и исправить его.

4) Задача связана с регулярными выражениями. Пусть у нас возникла потребность генерировать случайные индексы, состоящие из 10 символов: цифры и строчные буквы, причём сначала идут буквы, затем цифры.

Пусть у нас имеется генератор ID, однако в нём есть проблема: цифры и буквы перемешаны в каждом индексе:

```
import pandas as pd

import numpy as np

import string

import random

import re



def series_of_ids(n_symbols: int, n_rows: int) -> pd.Series:

    x = pd.Series([

        "".join(random.choices(string.ascii_lowercase + string.digits, k=S))

        for _ in range(N)

    ])

    return x
```

Пример вызова:

```
S = 10  # number of characters in the string.

N = 1_000_000  # number of rows



series_of_ids(n_symbols=S, n_rows=N)
```

Когда ошибку заметили, ваш коллега написал дополнительную функцию, исправляющую порядок символов в каждой строке:

```
def chars_and_digits(x: pd.Series) -> pd.Series:

    y = x.apply(lambda s: re.sub(r"[^a-z]", "", s) + re.sub(r"[^\d]", "", s))

    return y
```

Теперь всё работает корректно, однако коллега намекнул, что можно слегка исправить функцию так, чтобы она работала как минимум в 1.5 раза быстрее, только он забыл синтаксис...

Ваша задача: оптимизировать функцию chars_and_digits

5) Similar Items Price

В задаче динамического ценообразования встречается потребность добавлять цены похожих товаров как фичу (например, для регуляризации или для заполнения пропусков).

У нас имеются эмбеддинги всех товаров, которые нам предоставила команда матчинга, но не для всех товаров модель смогла посчитать цену. Нам необходимо восполнить пропуски на основе топ-К ближайших товаров в этом пространстве эмбеддингов.

Вам дан датафрейм (см. пример входных данных), состоящий из 3 колонок:

- item (int) – ID товара
- price (float) – цена (либо целочисленная, либо NaN)
- embedding (array of float) – вектора товаров (все одинаковой размерности)

Напишите функцию, которая находит средневзешенную цену на основе топ-К ближайших товаров (топ-К среди товаров с известной ценой!). Запишите полученную цену в новую колонку knn_price (отбросив знаки после запятой).

Также добавьте колонку new_price, которая будет содержать исходную price, если она известна, и knn_price, если вместо цены был пропуск.

Пример входных данных :

item	price	embedding
0	NaN	[-0.3972673232428443, 0.8117145449000501, -0.5...
1	547.0	[-0.423810037080344, -0.717516373836237, -0.07...
2	428.0	[-1.2528132445487394, -0.07981211620438243, 0....
3	367.0	[1.3740072805837524, 1.620956128545331, 0.6745...
4	454.0	[-0.4381208811247264, -0.44593733358604665, -1...
5	21.0	[0.4243173730684717, -0.14372456706152034, -0....
6	830.0	[0.420327274039293, -0.3063289538865571, 0.252...
7	884.0	[-0.22565423231399115, 0.43854132745703683, 0....
8	231.0	[-1.0684281979814219, 0.7390074871312797, -1.2...
9	NaN	[0.5749637416117086, -0.6917105385863515, -0.0...

Пример выходных данных с топ-2 ближайших товаров:

item	price	embedding	knn_price	new_price
0	NaN	[-0.3972673232428443, 0.8117145449000501, -0.5...	319.0	319.0
1	547.0	[-0.423810037080344, -0.717516373836237, -0.07...	607.0	547.0
2	428.0	[-1.2528132445487394, -0.07981211620438243, 0....	365.0	428.0
3	367.0	[1.3740072805837524, 1.620956128545331, 0.6745...	620.0	367.0
4	454.0	[-0.4381208811247264, -0.44593733358604665, -1...	632.0	454.0
5	21.0	[0.4243173730684717, -0.14372456706152034, -0....	816.0	21.0
6	830.0	[0.420327274039293, -0.3063289538865571, 0.252...	511.0	830.0
7	884.0	[-0.22565423231399115, 0.43854132745703683, 0....	109.0	884.0

8 231.0 [-1.0684281979814219, 0.7390074871312797, -1.2... 438.0 231.0

9 NaN [0.5749637416117086, -0.6917105385863515, -0.0... 676.0 676.0

API вашего решения:

```
def neighbors_price(X: pd.DataFrame, n_neighbors: int = 5) -> pd.DataFrame:
  X = X.copy()
  ...
  return X
```

6) Stocks

Наша модель ценообразования предсказывает количество продаж на завтра в рублях (колонка gmv) на основе некоторого набора фичей и цены (price). Известно также, сколько товаров на складе, т.е. доступно для продажи (stock).

Предполагается, что на данные товары нет никаких дополнительных промо-акций, скидок и для каждого покупателя цена одинакова.

GMV – *это суммарная выручка. вычисляется как число проданных товаров, умноженное на соответствующие цены.*

Необходимо написать функцию для постобработки предсказаний модели:

```
def limit_gmv(df: pd.DataFrame) → pd.DataFrame:
  ...
  return df
```

Пример данных на вход:

| sku | gmv | price | stock |

———————————————————————————

| 100 | 400 | 100 | 3 |

| 200 | 350 | 70 | 10 |

| 300 | 500 | 120 | 4 |

Пример данных после применения функции

| sku | gmv | price | stock |

———————————————————————————

| 100 | 300 | 100 | 3 |

| 200 | 350 | 70 | 10 |

| 300 | 480 | 120 | 4 |

7) Описание задачи

Градиентный бустинг – это ансамбль решающих деревьев. Несмотря на скорость обучения и универсальность, как и все деревянные модели, бустинг имеет свои ограничения. Например, бустинг может выдавать одинаковые предсказания для очень близких, но не идентичных объектов (в каждом листе – константа) + бустинг не умеет экстраполировать (не может выйти за пределы

значений, которые видел в обучающей выборке). Попробуем частично избавиться от этих проблем.

Соберите датасет

1. Возьмите 3-5 произвольных временных рядов
2. Выберите целевую переменную (например, ARPU или средний чек за день)
3. Сгенерируйте признаки по истории: агрегации (мин/макс/среднее/медиана/квантили) за день/неделю/месяцскользящие средниелаги (1-2-3-... дня)отношения...

Скорее всего, вам пригодится библиотека:

tsfresh - tsfresh 0.18.1.dev21+g28dff2b documentation

Имплементируйте 2-уровневую модель

1. линейная регрессия для предсказания тренда
2. градиентный бустинг (из вашей любимой библиотеки), который обучен на остатках предыдущей модели

Оцените точность вашей модели на time-series cross-validation (например, 3-5 фолдов), например, RMSE, MAE, MAPE

Для каждого фолда визуализируйте:

- реальное значение
- предсказание 2-уровневой модели
- предсказание только линейной модели
- предсказание только градиентного бустинга (обучите отдельно)

Для всех 3 моделей (линейная, бустинг, линейная+бустинг)

1. на отложенной выборке оцените точность (RMSE, MAE, MAPE)
2. обоснуйте несмещенность остатков с помощью 1-2 статистических тестов (наверняка, вам пригодится лекция Воронцова о прогнозе временных рядов https://www.youtube.com/watch?v=RdTxLXmbvjY)

Сделайте вывод относительно того, насколько 2-х уровневая модель лучше/хуже предсказывает временные ряды с учётом наличия тренда