

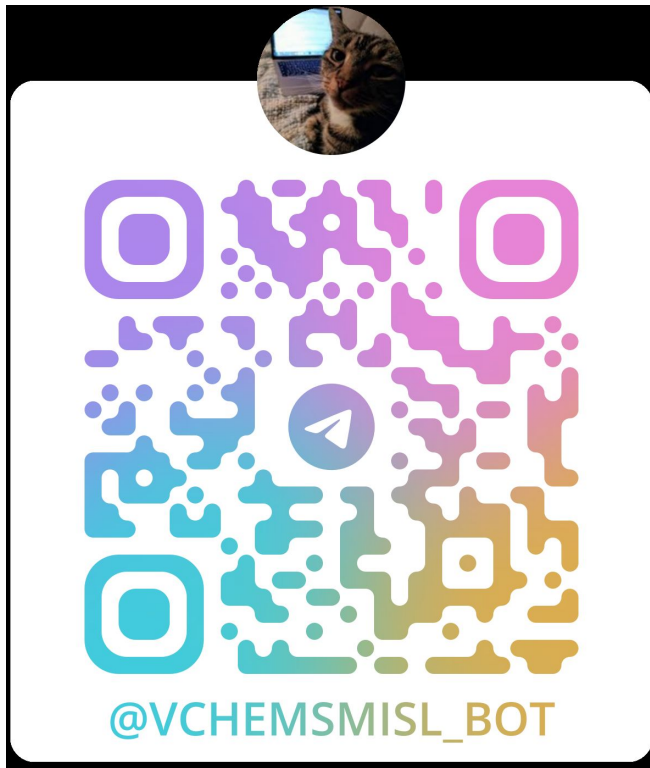
Simple ML Interface bot

проект Исакова Данилы, 21ФПЛ2.

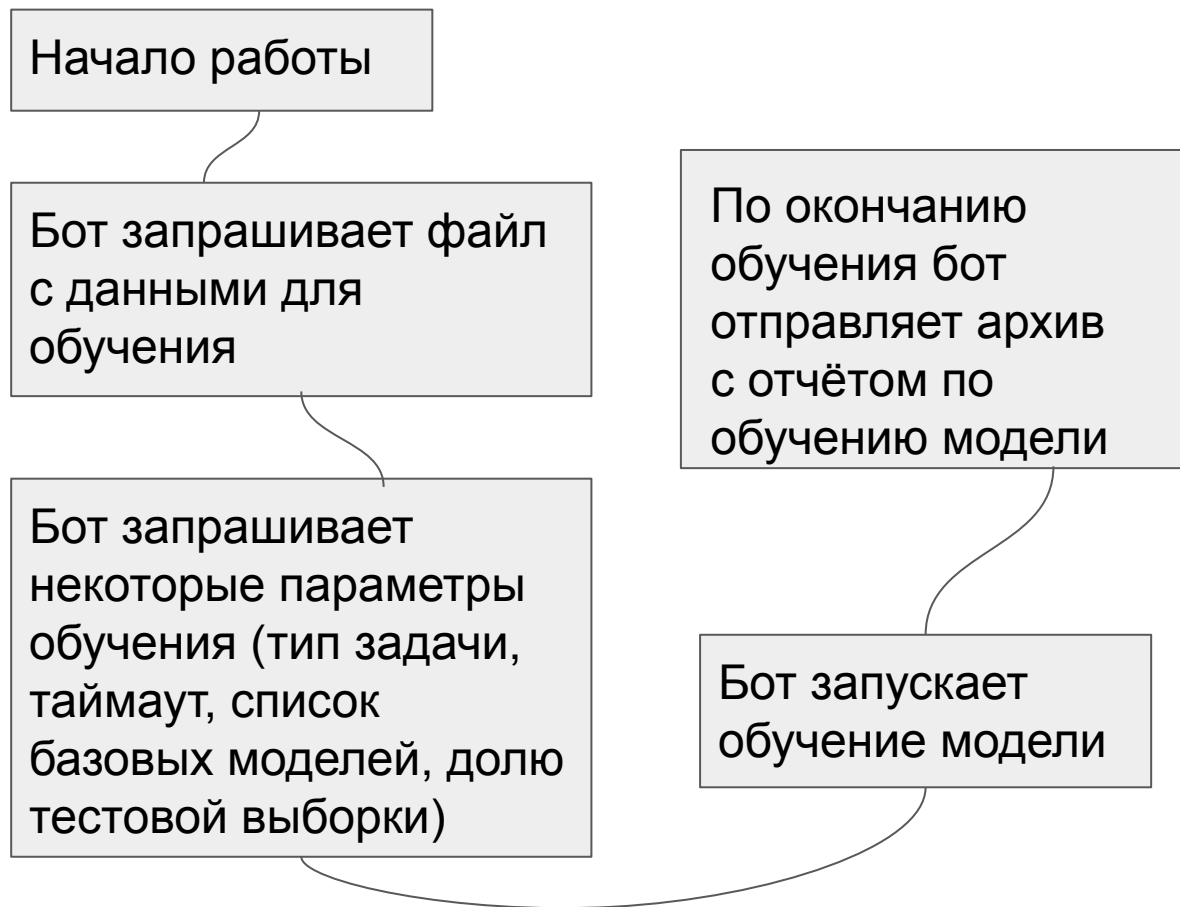
https://github.com/vchemsmisl/telegram_ml_interface_project


Суть проекта:

Был создан телеграм-бот, выступающий интерфейсом для обучения адаптируемой под данные модели машинного обучения. Цель его создания в том, чтобы предоставить удобный интерфейс для обучения модели, которым можно воспользоваться, не имея знаний в программировании и машинном обучении. Бот имеет простой функционал, но при этом позволяет эффективно обучать модели, подстраивающиеся под данные, благодаря использованию метода автоматического машинного обучения.




Работа бота со стороны пользователя:





 /start 16:05 ✓

Welcome to the Simple ML Interface bot!
Let's begin. 16:05

 **Start working**

Send the dataset with features and target values.
Supported formats: .csv
File size must be not more than 20 MB.
Also please set the name of your target column to "target". 16:05

  **mock_data.csv**
461.4 KB 16:05 ✓


Now choose the type of task, for which the model will be trained. 16:06

Regression


Binary classification


Multiclass classification

Multiple regression

 **Multiclass classification**


Now enter the proportion of test sample, which will be used for model validation.
Please enter the number from 0 to 1. 16:06

 0.3 16:06 ✓


 0.3 16:06 ✓

Now enter the names of machine learning models, which you want to use in AutoML algorithm.
Please write the names of models consequently, separating them by comma and space.

Possible variants are:
1. lgb, 2. lgb_tuned, 3. linear_l2, 4. cb, 5. cb_tuned.
Otherwise, if you want AutoML to define working model based on your data, please enter "auto". 16:06




 linear_l2 16:06 ✓

Now enter the number of timeout seconds.
Enter the non-negative integer number. 16:06

 60 16:06 ✓

Great! We have set all the necessary parameters and starting to train your AutoML model.
Now all you have to do is wait! The process will take several minutes.
When everything is ready, we will provide you with training results and a trained model. 16:06 ❤️

Congratulations!
We've just finished training AutoML model on your data.
In the following document you'll find the detailed report on model training. 16:06

  **model_training_report.zip**
318.9 KB 16:06 

Работа бота под капотом:

Запуск бота TgBot, сбор
ответов пользователя и
заполнение конфига
AutoMLModelConfig

Данные делятся на трейн
и валидацию и
передаются экземпляру
класса обучения модели
AutoMLModelEmployer

Модель обучается,
сохраняется отчёт
об обучении

Обращение к экземпляру
класса-фасада для
обработки данных и
обучения модели
ModelTrainingFacade

Внутри фасада
вызывается база
данных SQLiteDatabase,
из которой достаются
признаки и целевая
переменная

Экземпляр класса
TgBot отправляет
пользователю
архив с отчётом
об обучении
модели

```
import warnings
warnings.simplefilter(action='ignore', category=UserWarning)

from src.tgbot import TgBot
from src.model import AutoMLModelConfig

def main(): 1 usage 1 vchemsmisl

    config = AutoMLModelConfig()
    bot = TgBot(config)

    bot.execute_model_training_bot_interface()
    bot.start_infinity_polling()

if __name__ == '__main__':
    main()
```

main.py

```
@singleton_class 2 usages 1 vchemsmisl
class TgBot:

    def __init__(self, model_config): 1 vchemsmisl
        self.bot = telebot.TeleBot(BOT_TOKEN)
        self.model_config = model_config
        self.name = 'vchemsmisl_bot'
        self.chat_id = None
```

tgbot.py

```
class AutoMLModelConfig(ModelConfigBase): 8 usages 1 vchemsmisl

    def __init__(self): 1 vchemsmisl
        super().__init__()
        self.models = []
        self.timeout = 0
```

model.py

```
class AutoMLModelEmployer(ModelEmployerBase): 4 usages 1 vchemsmisl

    def __init__(self, model_config: AutoMLModelConfig): 1 vchemsmisl
        super().__init__(model_config)
```

model.py

```
class ModelTrainingFacade: 6 usages 1 vchemsmisl

    def __init__(self, filled_config): 1 vchemsmisl
        if (filled_config.models == []) or \
            (filled_config.timeout == 0) or \
            (filled_config.task_type == '') or \
            (filled_config.doc_type == '') or \
            (filled_config.test_proportion == 0):
            raise EmptyConfigError('One of parameters in conf

        self.config = filled_config
        self.database = None
        self.model = None
```

facade.py

```
class SQLiteDatabase: 8 usages 1 vchemsmisl

    def __init__(self, dataset_path): 1 vchemsmisl
        self.connector = SQLiteConnector()
        self.dataset_path = dataset_path
        self.database_name = 'database_for_training'
```

database.py

Тесты:

Несколько **unit-тестов** в папке **tests** + тесты непосредственно **через интерфейс чат-бота**

Используемые типы тестов (по разным классификациям):

- Позитивные: *test_get_raw_features_generator_yields_correct_data*, *test_get_raw_target_generator_yields_correct_data*
- Негативные: *test_get_raw_target_generator_rejects_incorrect_data*, *test_get_raw_features_generator_rejects_incorrect_data* и др.; отправка боту значений, выходящих за граничные
- Граничные: отправка боту граничных значений (например, 0 и 1 для размера тестовой выборки)
- Тесты состояния: *test_init_rejects_unfilled_config* и др. – проверки того, что на определённой стадии выполнения программы конфиг заполнен.
- Тесты как “чёрный ящик”, так и “белый ящик”
- Регрессионное тестирование

Оценка сложности алгоритма:

Точное количество времени выполнения кода и объём заполняемой памяти подсчитывать бессмысленно, так как эти параметры у нас могут сильно и непредсказуемо меняться от запуска к запуску в зависимости, в первую очередь, от объёма данных и величины таймаута.

Ниже представлены оценки сложности отдельных функций, для которых эта оценка представляется возможной, в O-нотации:

_get_raw_target_generator, *_get_raw_features_generator*,
_numpy_array_creation_from_raw – линейная $O(n)$

методы класса *AutoMLModelConfig* – константная $O(1)$

get_target, *get_features* – квадратная $O(n^2)$ (относительно *raw_target* и *raw_features* соответственно)

Объектно-ориентированное программирование:

```
class AutoMLModelEmployer(ModelEmployerBase): 4 usages 1 vchem
    def __init__(self, model_config: AutoMLModelConfig): 1 vc
        super().__init__(model_config)
```

```
class AutoMLModelConfig(ModelConfigBase): 8 us
```

Наследование от базовых классов

```
self.database = SQLiteDatabase(path_to_dataset)
```

```
self.connector = SQLiteConnector()
```

```
self.bot = telebot.TeleBot(BOT_TOKEN)
```

Композиции

```
class ModelConfigBase(ABC): 1 usage 1 vchemsmisl
```

```
    def __init__(self): 1 vchemsmisl
        self.task_type = ''
        self.test_proportion = 0
        self.doc_type = ''
```

```
@abstractmethod 1 usage (1 dynamic) 1 vchemsmisl
    def fill_task_type(self, task_type):
        pass
```

```
@abstractmethod 1 usage (1 dynamic) 1 vchemsmisl
    def fill_test_proportion(self, test_proportion):
        pass
```

```
@abstractmethod 1 usage (1 dynamic) 1 vchemsmisl
    def fill_doc_type(self, doc_type):
        pass
```

```
class ModelEmployerBase(ABC): 1 usage 1 vchemsmisl
```

```
    def __init__(self, model_config): 1 vchemsmisl
        if (model_config.models == []) or \
            (model_config.timeout == 0) or \
            (model_config.task_type == '') or \
            (model_config.doc_type == '') or \
            (model_config.test_proportion == 0):
            raise EmptyConfigError('One of parameters in
```

```
        self.model_config = model_config
        self.model = None
```

```
@abstractmethod 1 vchemsmisl
    def configure_model(self):
        pass
```

```
@abstractmethod 1 vchemsmisl
    def train_model(self, train_data, val_data):
        pass
```

```
@abstractmethod 1 vchemsmisl
    def make_prediction(self, test_data):
        pass
```

Абстрактные классы-интерфейсы

Итераторы, генераторы:

```
@staticmethod 3 usages vchemsmisl
def _get_raw_target_generator(raw_target):
    for val in raw_target:
        yield val[0]

@staticmethod 3 usages vchemsmisl
def _get_raw_features_generator(raw_target):
    for val in raw_target:
        yield list(val)

@staticmethod 3 usages vchemsmisl
def _numpy_array_creation_from_raw(iter_function):
    if not isinstance(iter_function, Iterator):
        raise NotIteratorError('The received function is not an iterator')

    array = []
    while True:
        try:
            array.append(next(iter_function))
        except StopIteration:
            break

    return numpy.array(array)
```

Самописные декораторы:

```
def singleton_class(cls): 1 usage vchemsmisl

    class_objects = {}

    def wrapper(*args, **kwargs): vchemsmisl

        if cls not in class_objects:
            class_objects[cls] = cls(*args, **kwargs)

        return class_objects[cls]

    return wrapper

@singleton_class 2 usages vchemsmisl
class TgBot:
```

Шаблоны проектирования:

1. Одиночка (синглтон)

```
@singleton_class 2 usages  vchemsmisl
class TgBot:

    def __init__(self, model_config):  vchemsmisl
        self.bot = telebot.TeleBot(BOT_TOKEN)
        self.model_config = model_config
        self.name = 'vchemsmisl_bot'
        self.chat_id = None
```

Так как бот у нас всегда один конкретный, логично использовать для создания соответствующего ему класса шаблон Одиночка.

2. Фасад

```
class ModelTrainingFacade: 6 usages  vchemsmisl

    def __init__(self, filled_config):  vchemsmisl
        if (filled_config.models == []) or \
            (filled_config.timeout == 0) or \
            (filled_config.task_type == '') or \
            (filled_config.doc_type == '') or \
            (filled_config.test_proportion == 0):
            raise EmptyConfigError('One of parameters in config is unfilled')

        self.config = filled_config
        self.database = None
        self.model = None
```

Используем фасад для объединения методов работы с БД и моделью для удобного взаимодействия этих методов с методами класса бота.

Докер:

1. Написал инструкции для сборки образа

```
FROM python:3.12
WORKDIR /telegram_ml_interface_project_image

COPY requirements.txt requirements.txt
COPY main.py main.py

COPY messages.py messages.py
COPY constants.py constants.py

COPY src/database.py src/database.py
COPY src/facade.py src/facade.py
COPY src/model.py src/model.py
COPY src/tgbot.py src/tgbot.py

RUN pip install --no-cache-dir -r requirements.txt

CMD ["python", "main.py"]
```

2. Запустил сборку образа

```
(.venv) PS C:\programming\telegram_ml_interface_project> docker build -t telegram_ml_interface_project_image .
[+] Building 98.4s (5/10)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 235B
=> [internal] load metadata for docker.io/library/python:3.12
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/python:3.12@sha256:251ef8e69b6ccdf3c7bf7effaa51179d59af35364dd9c86469142aa72a2c8fc
=> => resolve docker.io/library/python:3.12@sha256:251ef8e69b6ccdf3c7bf7effaa51179d59af35364dd9c86469142aa72a2c8fc
=> sha256:7aa817e66d893aff4b57b2c22574ec2afedff4673b8a095ae275244bfd3d7bc1 50.33MB / 64.39MB
=> sha256:fd304df3579db2df079c560619d7fdb87ff56bad7d61aa9aae36e1a2d0db 2.32kB / 2.32kB
=> sha256:54c7be425079efba0003054ee88abf721ffebca733bedd6f077d2809ee9aa6f 23.87MB / 23.87MB
=> sha256:0a96b8db280554b560ffeedf2e5f9843dc7b625f28192021ee103ecbcc2d629b 40.89MB / 48.50MB
=> sha256:251ef8e69b6ccdf3c7bf7effaa51179d59af35364dd9c86469142aa72a2c8fc 9.72kB / 9.72kB
=> sha256:acdb8b87a53fdbae5625dd1d52f64b95fe9f2c1eb92df388e53fcccc07c921b 6.06kB / 6.06kB
=> sha256:1523f4b3f502bf41ca8f8d8e649536ac0b3e5698ac81e9f518fb20c6516ca075 11.53MB / 211.31MB
=> [internal] load build context
=> => transferring context: 577B
```

Building 2412.0s (11/11) FINISHED

💀💀💀💀💀 (у автомл слишком много огромных зависимостей)

3. Образ создан и хранится локально (11 Гб 😊, поэтому не стал кидать на DockerHub)

<input type="checkbox"/>	Name	Tag	Image ID	Created	Size	Actions
<input type="checkbox"/>	telegram_mlInterface_proje	latest	add39d6a8793	4 minutes ago	11.09 GB	

4. Контейнер работает

romantic_nobel
a95a3e164636 telegram_ml_interface_project_image:latest
STATUS
Exited (137) (0 seconds ago)

Logs

Inspect

Bind mounts

Exec

Files

Stats

on instructions.
2024-12-28 14:29:51 index feature_4 feature_5 ... feature_25 feature_26 target
2024-12-28 14:29:51 0 1206.0 -0.284175 0.152852 ... -0.387116 -0.171989 0.0
2024-12-28 14:29:51 1 3931.0 0.902695 -0.277583 ... 1.658655 -0.701311 0.0
2024-12-28 14:29:51 2 4915.0 -0.324629 1.261273 ... 0.202136 0.560082 0.0
2024-12-28 14:29:51 3 3819.0 -0.484223 -1.171850 ... 1.052866 -0.191900 0.0
2024-12-28 14:29:51 4 4603.0 -1.593895 -0.173749 ... 0.120611 0.172678 0.0
2024-12-28 14:29:51 [5 rows x 10 columns]
2024-12-28 14:29:51 index feature_4 feature_5 ... feature_25 feature_26 target
2024-12-28 14:29:51 0 4282.0 0.023398 1.026587 ... 0.597747 -1.763222 0.0
2024-12-28 14:29:51 1 3619.0 1.114463 1.102542 ... 0.028098 -0.114579 0.0
2024-12-28 14:29:51 2 3092.0 0.228467 0.292785 ... -0.034821 -0.222046 0.0
2024-12-28 14:29:51 3 3177.0 1.277677 1.948489 ... 2.696636 -1.123427 0.0
2024-12-28 14:29:51 4 4824.0 -0.214934 -2.596850 ... -0.684957 -1.684571 0.0
2024-12-28 14:29:51 [5 rows x 10 columns]
2024-12-28 14:29:51 [11:29:51] Start auto ml utilizer with listed constraints:
2024-12-28 14:29:51 [11:29:51] - time: 60.00 seconds
2024-12-28 14:29:51 [11:29:51] - CPU: 4 cores
2024-12-28 14:29:51 [11:29:51] - memory: 16 GB
2024-12-28 14:29:51 [11:29:51] =====
2024-12-28 14:29:51 [11:29:51] Start 0 autonl preset configuration:
2024-12-28 14:29:51 [11:29:51] conf_0_sel_type_0.yml, random state: ('reader_params': ('random_state': 42), 'nn_params': ('random_state': 42), 'general_params': ('return_all_predictions': False))
2024-12-28 14:29:51 [11:29:51] Stdout logging level is INFO.
2024-12-28 14:29:51 [11:29:51] Task: binary
2024-12-28 14:29:51 [11:29:51] Start autonl preset with listed constraints:
2024-12-28 14:29:51 [11:29:51] - time: 60.00 seconds

Базы данных:

Использовал SQLite и библиотеку sqlite3:

- он проще
- быстрый и простой доступ к БД
- в коде выполняются маленькие и очень простые запросы к БД → нет надобности в более сложном инструменте

```
class SQLiteConnector: 1 usage  vchemsmisl

    def __init__(self):  vchemsmisl
        self.database_path = DATABASE_PATH
        self.connection = self.create_connection()

    def create_connection(self): 1 usage  vchemsmisl

        try:
            connection = sqlite3.connect(self.database_path)
        except Error as e:
            raise ConnectionToDatabaseError(f"The error '{e}' occurred")

        return connection
```

Класс, отвечающий за запросы к БД

```
class SQLiteDatabase: 8 usages  vchemsmisl

    def __init__(self, dataset_path):  vchemsmisl
        self.connector = SQLiteConnector()
        self.dataset_path = dataset_path
        self.database_name = 'database_for_training'

    def create_database(self): 1 usage  vchemsmisl
        dataframe = pandas.read_csv(self.dataset_path).drop(labels='Unnamed: 0', axis=1)

        dataframe.to_sql(name=self.database_name,
                        con=self.connector.connection,
                        if_exists='replace')
```

Класс, отвечающий за саму БД

CI/CD:

В основном для прогона тестов на гите:

7 workflow runs

✓ Dockerfile added	main
CI #7: Commit df64e32 pushed by vchemsmisl	
✓ dealing with bugs...	main
CI #6: Commit a0cd253 pushed by vchemsmisl	
✗ dealing with bugs...	main
CI #5: Commit 3b87751 pushed by vchemsmisl	
✗ dealing with bugs...	main
CI #4: Commit 9139570 pushed by vchemsmisl	
✗ dealing with bugs...	main
CI #3: Commit a2f2fa6 pushed by vchemsmisl	

```
name: CI
```

```
on:
```

```
  push:
```

```
    branches: [ main ]
```

```
  pull_request:
```

```
    branches: [ main ]
```

```
jobs:
```

```
  build:
```

```
    runs-on: ubuntu-latest
```

```
    steps:
```

```
      - uses: actions/checkout@v4
```

```
      - name: Set up Python
```

```
        uses: actions/setup-python@v5
```

```
        with:
```

```
          python-version: '3.12'
```

```
      - name: Install dependencies
```

```
        run: |
```

```
          python -m pip install --upgrade pip
```

```
          pip install -r requirements.txt
```

```
      - name: Run tests
```

```
        run: python -m pytest tests
```

Разные интересные библиотеки:

1. [telebot](#) – для всей логики бота.
2. [LightAutoML](#) – для обучения модели автоматического машинного обучения. Умеет делать feature selection, распознавать типы данных, комбинировать модели, в том числе подбирать их автоматически, оптимизировать процесс обучения ансамблей, выдавать подробный отчёт об обучении модели.
3. [requests](#), [pathlib](#), [shutil](#) – для работы с файловой системой.
4. [warnings](#) – для блокировки вывода некоторых предупреждений.
5. [typing](#) – для получения специфических классов объектов, существующих в дефолтном питоне.

Юнит-тесты:

Папка tests

```
class AutoMLModelEmployerTests(unittest.TestCase):  # vchemsmisl

    def test_init_rejects_unfilled_config(self):  # vchemsmisl
        config = AutoMLModelConfig()
        with self.assertRaises(EmptyConfigError):
            _ = AutoMLModelEmployer(config)

class ModelTrainingFacadeTests(unittest.TestCase):  # vchemsmisl

    def test_init_rejects_unfilled_config(self):  # vchemsmisl
        config = AutoMLModelConfig()
        with self.assertRaises(EmptyConfigError):
            _ = ModelTrainingFacade(config)

    def test_create_and_train_model_rejects_variables_of_wrong_type(self):  # vchemsmisl
        config = AutoMLModelConfig()
        config.test_proportion = 0.3
        config.doc_type = 'csv'
        config.models = ['auto']
        config.task_type = 'binary'
        config.timeout = 90

        facade = ModelTrainingFacade(config)
        train_data = ['1', '2', '3']
        val_data = None

        with self.assertRaises(WrongTypeOfVariablesError):
            facade.create_and_train_model(train_data, val_data)
```

```
class SQLiteDatabaseTests(unittest.TestCase):  # vchemsmisl

    def test_get_raw_target_generator_yields_correct_data(self):  # vchemsmisl
        path_to_dataset = f'{DATA_PATH.absolute().as_posix()}/data_for_training.csv'
        database = SQLiteDatabase(path_to_dataset)

        array = [(1, ), (2, ), (3, ), (4, )]
        yield_value = next(database._get_raw_target_generator(array))

        self.assertEqual(first=1, yield_value)

    def test_get_raw_target_generator_rejects_incorrect_data(self):  # vchemsmisl
        path_to_dataset = f'{DATA_PATH.absolute().as_posix()}/data_for_training.csv'
        database = SQLiteDatabase(path_to_dataset)

        array = [1, 2, 3, 4]
        with self.assertRaises(TypeError):
            _ = next(database._get_raw_target_generator(array))

    def test_get_raw_features_generator_yields_correct_data(self):  # vchemsmisl
        path_to_dataset = f'{DATA_PATH.absolute().as_posix()}/data_for_training.csv'
        database = SQLiteDatabase(path_to_dataset)

        array = [(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6)]
        yield_value = next(database._get_raw_features_generator(array))

        self.assertEqual(first=[1, 2, 3], yield_value)
```


Ограничения алгоритма:

1. Работает только с числовыми данными, при этом файл не должен превышать 20 Мб (ограничение телеграма). Также пока принимает файлы только в .csv формате.
2. Небольшая проблема: если в коде возникнут ошибки (то есть поднимется исключение), у пользователя не будет возможности об этом узнать.
3. Пока у пользователя нет возможности получить файл модели или произвести инференс модели с помощью бота.
4. Бот синхронный(

Fin!

