Программный код signess-app

signess-app/config.py:

```
from signess.network import FedotCNN


from inskrib.autograph import Autograph

from inskrib.documents import Document


title = "Signess App"


width = 680

height = 350


autograph = Autograph(size=(380, 380))

document = Document()

network = FedotCNN()



class Config():

    def __init__(self, root):

        self.autograph = autograph

        self.document = document


        self.network = network
```

```
self.dataset = None
self.path_to_dataset = None

self.epochs_count = 5

root.title(title)
screenwidth = root.winfo_screenwidth()
screenheight = root.winfo_screenheight()

alignstr = '%dx%d+%d+%d' % (
    width,
    height,
    (screenwidth - width) / 2,
    (screenheight - height) / 2
)

root.geometry(alignstr)
root.resizable(width=False, height=False)
```

signess-app/main.py:

```
import sv_ttk
import tkinter as tk
from config import Config
from core.comands import Comands
from core.components import Components



def run():
    root = tk.Tk()
    sv_ttk.set_theme("dark", root)
    App(root)
    root.mainloop()



class App(Config, Comands, Components):
    def __init__(self, root):
        Config.__init__(self, root)
        Components.__init__(self, root)



if __name__ == "__main__":
    run()
```

signess-app/requirements.txt:

signess

sv_ttk

signess-app/core/comands.py:

```
import fitz
from threading import Thread
import tkinter.filedialog as tkFile
from core.service.model import ModelService
from core.service.dataset import DatasetService
from core.service.classification import ClassificationService
from core.utils import show_info, show_error, check_path, check_dataset


class Comands():
    """

    В этом классе находится все основные команды

    Логика для команд находится в сервисах

    """

    def load_model(self):
        show_info(
            "Загрузка модели работает только с версией Fedot >= 0.7.3"
        )

        path = tkFile.askdirectory()
        if not check_path(path):
            return

        try:
            ModelService.load(self.network, path)
            self.label_model.configure(
```

```
            text="Модель готова",
            foreground="lime"
        )
    except:
        show_error()

def save_model(self):
    path = tkFile.askdirectory()
    if not check_path(path):
        return

    try:
        ModelService.save(self.network, path)
    except:
        show_error()

def train_model(self):
    if not check_dataset(self.dataset):
        return

    def train():
        ModelService.train(self.network, self.dataset, self.epochs_count)
        self.label_model.configure(
            text="Модель готова",
            foreground="lime"
        )
        show_info("Модель обучена!")

    try:
        self.label_model.configure(
            text="Модель обучается...",
```

```
            foreground="orange"
        )

        thread = Thread(target=train)
        thread.start()
    except:
        self.label_model.configure(
            text="Модель не обучена",
            foreground="red"
        )
        show_error()

def blunt_model(self):
    try:
        ModelService.blunt(self.network)
        self.label_model.configure(
            text="Модель не обучена",
            foreground="red"
        )
        show_info("Модель сбросила обучение!")
    except:
        show_error()

def accuracy_model(self):
    filetypes = (("Датасет", "*.npz"),)
    path = tkFile.askopenfilename(
        title="Загрузка npz датасета",
        filetypes=filetypes
    )

    if not check_path(path):
```

```
            return

        def accuracy():
            dataset = DatasetService.load(self.network, path)
            (roc_auc, files_count, drop_count) = ModelService.accuracy(
                self.network, dataset)
            show_info(
                f"Точность модели (rog auc): {(roc_auc * 100):.3f}%\n\nКоличество
ошибочных распознавания {drop_count} из {files_count}"
            )

        try:
            thread = Thread(target=accuracy)
            thread.start()
        except:
            show_error()

    def load_dataset(self):
        filetypes = (("Датасет", "*.npz"),)
        path = tkFile.askopenfilename(
            title="Загрузка npz датасета",
            filetypes=filetypes
        )

        if not check_path(path):
            return

        try:
            dataset = DatasetService.load(self.network, path)
            self.path_to_dataset = path
            self.dataset = dataset
```

```
            self.label_dataset.configure(
                text="Датасет загружен",
                foreground="lime"
            )
            show_info("Датасет загружен!")
        except:
            show_error()

    def generate_dataset(self):
        path = tkFile.askdirectory()
        if not check_path(path):
            return

        def generate():
            path_to_dataset = DatasetService.generate(
                path,
                self.autograph,
                self.document
            )
            self.path_to_dataset = path_to_dataset

            dataset = DatasetService.load(self.network, path_to_dataset)
            self.dataset = dataset

            self.label_dataset.configure(
                text="Датасет загружен",
                foreground="lime"
            )
            show_info("Датасет загружен!")

        try:
```

```
            thread = Thread(target=generate)
            thread.start()
            show_info("Датасет генерируется...")
        except:
            show_error()

    def classification(self):
        if not check_dataset(self.dataset):
            return
        if not check_dataset(self.path_to_dataset):
            return

        filetypes = (("Файл", "*.png *.jpg *.jpeg *.bmp *.pdf"),)
        path = tkFile.askopenfilename(
            title="Открыть файл",
            filetypes=filetypes
        )

        if not check_path(path):
            return

        filetype = path.split('.')[-1]
        if (filetype == "pdf"):
            with fitz.open(path) as pdf:
                path = "./temp.png"
                page = pdf.load_page(0)
                pix = page.get_pixmap()
                pix.save(path)

        try:
            classify = ClassificationService.classificate(
```

```
            self.network,
            self.autograph,
            self.path_to_dataset,
            path
        )

        result = ""
        for item in classify:
            result += f"{item[0][0]}: {item[1]*100}% \n"

        person = f"{classify[0][0][0]}: {classify[0][1]*100}%"
        self.label_result.configure(text=person)

        show_info(result)
    except:
        show_error()

def epochs(self):
    new_epochs_count = self.entry_epochs.get()
    self.entry_epochs.delete(0, 9999)

    try:
        new_epochs_count = int(new_epochs_count)
    except ValueError:
        show_error("Нужно ввести число!")
        return

    new_epochs_count = abs(new_epochs_count)

    if (new_epochs_count == 0):
        new_epochs_count = 1
```

```
self.label_epochs.configure(
    text=f"Количество эпох: {new_epochs_count}"
)

self.epochs_count = new_epochs_count
```

signess-app/core/components.py:

```
from tkinter import ttk


class Components():
    """

    Класс отвечает за отрисовку UI

    """

    def __init__(self, root):
        self.__generate_components(root)
        self.__render_components()

    def __generate_components(self, root):
        self.btn_load_model = ttk.Button(
            master=root,
            text="Загрузить модель",
            command=self.load_model,
        )

        self.btn_save_model = ttk.Button(
            master=root,
            text="Сохранить модель",
            command=self.save_model,
        )

        self.btn_train_model = ttk.Button(
            master=root,
            text="Обучить модель",
```

```
    command=self.train_model,
)

self.btn_train_blunt = ttk.Button(
    master=root,
    text="Сбросить обучение",
    command=self.blunt_model,
)

self.btn_accuracy = ttk.Button(
    master=root,
    text="Проверить точность",
    command=self.accuracy_model,
)
self.btn_generate_dataset = ttk.Button(
    master=root,
    text="Создать датасет",
    command=self.generate_dataset,
)

self.btn_load_dataset = ttk.Button(
    master=root,
    text="Загрузить датасет",
    command=self.load_dataset,
)

self.btn_classification = ttk.Button(
    master=root,
    text="Классификация",
    command=self.classification,
)
```

```
self.label_dataset = ttk.Label(
    master=root,
    text="Нет датасета",
    foreground="red",
    anchor="center"
)

self.label_model = ttk.Label(
    master=root,
    text="Модель не обучена",
    foreground="red",
    anchor="center"
)

self.label_epochs = ttk.Label(
    master=root,
    font=('Helvetica', 8),
    text="Количество эпох: 5",
    foreground="white",
    anchor="center",
)
self.entry_epochs = ttk.Entry(master=root)
self.btn_apply_epochs = ttk.Button(
    master=root,
    text="Применить",
    command=self.epochs,
)

self.label_result = ttk.Label(
    master=root,
```

```
            text="...",
            foreground="white"
        )

    def __render_components(self):
        self.btn_load_dataset.place(x=30, y=30, width=200, height=40)
        self.btn_generate_dataset.place(x=30, y=80, width=200, height=40)
        self.btn_accuracy.place(x=30, y=130, width=200, height=40)

        self.btn_load_model.place(x=240, y=30, width=200, height=40)
        self.btn_save_model.place(x=240, y=80, width=200, height=40)
        self.btn_train_blunt.place(x=240, y=130, width=200, height=40)

        self.btn_train_model.place(x=30, y=180, width=410, height=40)
        self.btn_classification.place(x=30, y=230, width=410, height=40)

        self.label_dataset.place(x=450, y=30, width=210, height=40)
        self.label_model.place(x=450, y=80, width=210, height=40)

        self.label_epochs.place(x=450, y=160, width=210, height=20)
        self.entry_epochs.place(x=450, y=180, width=80, height=40)
        self.btn_apply_epochs.place(x=540, y=180, width=120, height=40)

        self.label_result.place(x=30, y=280, width=620, height=40)
```

signess-app/core/utils.py:

```
import csv
import tkinter.messagebox as tkMb
from signess.dataset import Dataset


def create_dataset(path, autograph, document):
    dataset = Dataset(autograph, document)
    path_to_dataset = dataset.generate(path)
    return path_to_dataset


def check_path(path):
    if path:
        return True
    show_error("Нет пути! (No way!)")


def check_dataset(dataset):
    if dataset:
        return True
    show_error("Нет датасета!")


# оставлю так, вряд-ли пригодится
# def check_model_training(network):
#     if network.is_fitted:
#         return True
#     show_error("Модель не обучена!")
```

```
def csv_to_array(path_to_csv: str):
    csv_result = []
    with open(path_to_csv) as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            csv_result.append(row)

    return csv_result


def show_error(message="Произошла критическая ошибка!"):
    tkMb.showerror("Ошибка!", message)


def show_info(message="Отлично!"):
    tkMb.showinfo("Информация", message)
```

signess-app/core/service/classification.py:

```
import os
import cv2
from core.utils import csv_to_array


class ClassificationService():
    def classificate(network, autograph, path_to_dataset, path_to_picture,
path_to_csv="./result/persons.csv"):
        temp_path = "./temp.png"

        picture = autograph.get_clear_autograph(path_to_picture)
        cv2.imwrite(temp_path, picture)

        classify = network.classify(temp_path, path_to_dataset)
        os.remove(temp_path)

        persons = csv_to_array(path_to_csv)

        def sort(item):
            return item[1]

        result = sorted(
            zip(persons, classify[0]),
            key=sort,
            reverse=True
        )[:3]

        return result
```

signess-app/core/service/dataset.py:

```
from core.utils import create_dataset


class DatasetService():
    def load(network, path):
        dataset = network.load_dataset(path)
        return dataset

    def generate(path, autograph, document):
        path_to_dataset = create_dataset(path, autograph, document)
        return path_to_dataset
```

signess-app/core/service/model.py:

```
from sklearn.metrics import roc_auc_score
from core.utils import csv_to_array


class ModelService():
    def load(network, path):
        network.load(path)

    def save(network, path):
        network.save(f"{path}/model")

    def train(network, dataset, epochs):
        network.train(dataset, epochs)

    def blunt(network):
        network.blunt()

    def accuracy(network, dataset):
        predicts = network.predict(dataset)
        roc_auc = roc_auc_score(
            y_true=dataset.target,
            y_score=predicts.predict,
            multi_class="ovo"
        )

        drop_count = 0
        drop_path = "./drops.csv"
        open(drop_path, "w")

        files = csv_to_array("./result/filenames.csv")
```

```
files_count = len(files)

labels = dataset.class_labels
for i, predictArr in enumerate(predicts.predict):
    predictIndex = max(enumerate(predictArr), key=lambda x: x[1])[0]
    predict = labels[predictIndex]
    target = dataset.target[i]

    if predict != target:
        drop_count += 1
        with open(drop_path, 'a') as file:
            file.write(
                f'{files[i][0]},predict:{predict},expected:{target}\n'
            )

return (roc_auc, files_count, drop_count)
```