Основные моменты приложения model.py

1. Импорты

- **Библиотеки для работы с данными:** `os`, `pandas`, `numpy`.
- **Машинное обучение:** `sklearn` (разделение данных, кодирование меток, метрики).
- **Word2Vec:** `gensim`.
- **Нейронные сети:** `tensorflow`, `keras`.
- **База данных:** `psycopg2` (PostgreSQL).
- **Логирование:** `logging`.
- **Сериализация:** `joblib`.
- **Веб-сервер:** `flask`.

2. Flask-приложение

- **Инициализация:** `app = Flask(__name__)`.
- **Глобальные переменные:**
  - `main_model`: Основная модель для классификации.
  - `w2v_model`: Word2Vec модель для преобразования текста в векторы.
  - `label_encoder`: Кодировщик меток.

3. Маршруты

- **`/ping`:** Проверка работоспособности сервера. Возвращает `{"message": "pong"}`.
- **`/predict`:** Принимает JSON с текстом, делает предсказание и возвращает результат в формате JSON.
- **`/train`:** Обучает модель на данных из базы данных, сохраняет модели и возвращает сообщение об успешном завершении.

4. Функции

- **`connect`:** Устанавливает соединение с PostgreSQL.
- **`get_data`:** Получает данные из таблицы `sample_table`.
- **`text_to_vector`:** Преобразует текст в вектор с помощью Word2Vec.
- **`save_model`:** Сохраняет основную модель в файл.
- **`save_W2V_model`:** Сохраняет Word2Vec модель.
- **`loadWord2VecModel`:** Загружает Word2Vec модель из файла.
- **`loadModel`:** Загружает основную модель из файла.

5. Обучение модели

- Загрузка данных из базы.
- Обучение Word2Vec модели.
- Преобразование текстов в векторы.
- Кодирование меток.
- Обучение нейронной сети.
- Сохранение моделей.

6. Запуск приложения

- Загрузка моделей и кодировщика меток.
- Запуск Flask-сервера на порту 8081.

7. Логирование

- Логирование всех этапов работы для отладки и мониторинга.

Ключевые этапы работы:

1. **Инициализация:** Загрузка моделей и кодировщика меток.
2. **Обучение:** Загрузка данных, обучение Word2Vec модели, преобразование текстов в вектора, обучение нейронной сети.
3. **Предсказание:** Преобразование входного текста в вектор, предсказание метки и возврат результата.
4. **Логирование:** Логирование всех этапов работы для отладки и мониторинга.

db/Dockerfile
db/init-scripts/

Процесс поднятия Docker-контейнера с PostgreSQL и SQL-скриптами:

1. **Подготовка Dockerfile:**
   - Убедитесь, что у вас есть Dockerfile, который использует официальный образ PostgreSQL (в данном случае `postgres:14`).
   - В Dockerfile заданы переменные окружения:
     - `POSTGRES_USER=pagamov`
     - `POSTGRES_PASSWORD=multipass`
     - `POSTGRES_DB=database`

- SQL-скрипты для инициализации базы данных должны быть размещены в директории `init-scripts/`. Эти скрипты автоматически выполняются при первом запуске контейнера, так как они копируются в `/docker-entrypoint-initdb.d/`.

2. **Сборка Docker-образа:**
   - Перейдите в директорию, где находится ваш Dockerfile.
   - Выполните команду для сборки образа:

     ```
     docker build -t db .
     ```

3. **Запуск Docker-контейнера:**
   - После успешной сборки образа запустите контейнер с помощью команды:

     ```
     docker run -d -p 5432:5432 db
     ```

app/redis.go

Redis Integration in Go

Основные функции

- **`initRedis()`** : Инициализация подключения к Redis.
- **`checkIfInRedis(c *gin.Context) bool`** : Проверка наличия данных в Redis.
- **`addToRedis(jsonData []byte, analyz Analyz)`** : Добавление данных в Redis.
- **`getFromRedis(jsonData []byte) Analyz`** : Получение данных из Redis.

Зависимости

- `github.com/gin-gonic/gin`
- `github.com/redis/go-redis/v9`
- `encoding/json`
- `context`
- `log`, `fmt`

Использование

1. Инициализируйте Redis: `initRedis()`.
2. Проверьте данные: `checkIfInRedis(c)`.
3. Добавьте данные: `addToRedis(jsonData, analyz)`.
4. Получите данные: `getFromRedis(jsonData)`.

Пример

```
func main() {
    initRedis()
    r := gin.Default()

    r.POST("/check", func(c *gin.Context) {
        if checkIfInRedis(c) {
            c.JSON(200, gin.H{"message": "Data found"})
        } else {
            c.JSON(404, gin.H{"message": "Data not found"})
        }
    })

    r.POST("/add", func(c *gin.Context) {
        var analyz Analyz
        jsonData := getJsonData(c)
        addToRedis(jsonData, analyz)
        c.JSON(200, gin.H{"message": "Data added"})
    })

    r.POST("/get", func(c *gin.Context) {
        jsonData := getJsonData(c)
        analyz := getFromRedis(jsonData)
        c.JSON(200, analyz)
    })

    r.Run()
}
```

app/model.go

HTTP POST запрос к модели

Основные функции

- **`makePostRequestToModel(c *gin.Context) Analyz`** : Отправляет POST запрос к модели и возвращает результат в виде структуры `Analyz` .
- **`getJsonData(c *gin.Context) []byte`** : Преобразует данные запроса в JSON формат.

Зависимости

- `net/http`
- `encoding/json`
- `io`
- `log`
- `github.com/gin-gonic/gin`

Использование

1. Отправьте POST запрос к модели: `makePostRequestToModel(c)` .
2. Преобразуйте данные запроса в JSON формат: `getJsonData(c)` .

Пример

```
func main() {
    r := gin.Default()

    r.POST("/predict", func(c *gin.Context) {
        analyz := makePostRequestToModel(c)
        c.JSON(200, analyz)
    })

    r.Run()
}
```

app/main.go

HTTP API с использованием Gin и Redis

Основные функции

- **Инициализация и запуск сервера**:
  - `main()` : Инициализирует Redis, настраивает маршруты и запускает сервер на порту 8080.
  - `Router.Init()` : Инициализирует Gin-роутер.
  - `Router.AddMethod()` : Добавляет маршруты для обработки запросов.
  - `Router.Start(port string)` : Запускает сервер на указанном порту.
- **Маршруты**:
  - **`POST /analyze`** : Анализирует текст, используя Redis для кэширования данных.
  - **`GET /statistics/:begin/:end`** : Возвращает статистику за указанный период.
  - **`GET /ping`** : Проверка работоспособности сервера (возвращает "pong").
  - **`GET /logs`** : Возвращает все логи.
- **Вспомогательные функции**:
  - `validateDate(dateStr string) bool` : Проверяет корректность даты в формате `dd.mm.yyyy` .
  - `parseDate(date string) time.Time` : Преобразует строку даты в объект `time.Time` .
  - `getJsonData(c *gin.Context) []byte` : Преобразует данные запроса в JSON.

Зависимости

- `github.com/gin-gonic/gin`
- `encoding/json`
- `log`
- `net/http`
- `time`

Пример использования

```
func main() {
        initRedis()

        var router Router
        router.Init()
        router.AddMethod()
        router.Start("8080")
}

func (api *Router) Init() {
        api.router = gin.Default()
}

func (api *Router) AddMethod() {
        api.router.POST("/analyze", analyze)
        api.router.GET("/statistics/:begin/:end", statistics)
        api.router.GET("/ping", ping)
        api.router.GET("/logs", getLogs)
}
```

app/db.go

Работа с PostgreSQL

Основные функции

- **`addLog(text string, label string, info string)`** : Добавляет запись в таблицу `log_table` в PostgreSQL.
- **`getLog(date_start string, date_end string) []Statistics`** : Возвращает записи из таблицы `log_table` за указанный период.

Зависимости

- `database/sql`
- `encoding/json`
- `log`
- `time`
- `github.com/lib/pq` (драйвер PostgreSQL)

./make.sh

Docker и настройка окружения

Основные функции

- **Запуск контейнеров Docker**:
  - Redis, основное приложение, PostgreSQL и модель.
- **Копирование и настройка данных**:
  - Копирование резервных данных, сборка и перенос данных в PostgreSQL.
- **Сборка и запуск контейнеров**:
  - Сборка и запуск контейнеров для каждого компонента системы.

Зависимости

- Docker
- Python 3.12
- Go
- pgloader