Пример_2_Создания ИИ-агента для по строения карт депонирования глерода лесом с использованием **LangChain** и LLM

Создание ИИ-агента для построения карт депонирования углерода лесом с использованием LangChain и LLM

Ниже представлен пример создания ИИ-агента, который поможет анализировать, визуализировать и строить карты депонирования углерода лесными массивами на основе их таксационных характеристик.

Необходимые компоненты

PYTHON

```
# Установка необходимых библиотек

# pip install langchain langchain-openai openai pandas geopandas matplotlib numpy scikit-learn rasterio pydantic
```

Основной код агента

PYTHON

```
import os

from typing import List, Dict, Any, Optional

import pandas as pd

import geopandas as gpd

import numpy as np

import matplotlib.pyplot as plt

from rasterio import features

from rasterio.transform import from_origin

import rasterio


from langchain.agents import AgentExecutor, create_openai_functions_agent

from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder

from langchain_openai import ChatOpenAI

from langchain.memory import ConversationBufferMemory

from langchain.tools import BaseTool, StructuredTool

from langchain.pydantic_v1 import BaseModel, Field
```

```
# Установка API ключа (в реальности лучше использовать переменные окружения)
os.environ["OPENAI_API_KEY"] = "ваш-ключ-api"

# Класс для работы с данными таксации леса
class ForestInventoryHandler:
    """Обработчик таксационных данных леса"""

    def __init__(self):
        self.inventory_data = None
        self.carbon_data = None
        self.gdf = None

    def load_data(self, file_path: str) -> str:
        """Загружает данные таксации из CSV или гео-формата"""
        try:
            if file_path.endswith('.csv'):
                self.inventory_data = pd.read_csv(file_path)
                return f"Загружены CSV данные: {len(self.inventory_data)} записей"
            elif file_path.endswith('.shp'):
                self.gdf = gpd.read_file(file_path)
                self.inventory_data = pd.DataFrame(self.gdf.drop(columns='geometry'))
                return f"Загружены данные из шейп-файла: {len(self.inventory_data)} записей с геометрией"
            else:
                return "Неподдерживаемый формат файла. Используйте CSV или SHP."
        except Exception as e:
            return f"Ошибка при загрузке данных: {str(e)}"
```

```
    def calculate_carbon_sequestration(self) -> str:
        """Рассчитывает накопление углерода на основе таксационных характеристик"""
        if self.inventory_data is None:
            return "Сначала загрузите данные таксации"

        try:
            # Проверяем наличие необходимых колонок в данных
            required_columns = ['age', 'height', 'volume', 'species']
            missing = [col for col in required_columns if col not in self.inventory_data.columns]

            if missing:
                return f"Отсутствуют необходимые колонки: {', '.join(missing)}"

            # Коэффициенты для разных пород деревьев
            species_factors = {
                'pine': 1.2,      # сосна
                'spruce': 1.3,    # ель
                'birch': 0.8,     # берёза
                'oak': 1.5,       # дуб
                'aspen': 0.7,     # осина
                'larch': 1.1,     # лиственница
                'cedar': 1.4,     # кедр
                'fir': 1.25,      # пихта
                'mixed': 1.0      # смешанный
            }

            # Базовая формула расчета накопления углерода
```

```
        # В реальности использовались бы специальные аллометрические уравнения
        self.inventory_data['carbon'] = (
            0.45 *                                  # доля углерода в сухом веществе
            self.inventory_data['volume'] *                 # объем древесины (м³/га)
            0.65 *                                  # базовая плотность древесины (т/м³)
            np.log1p(self.inventory_data['age'] * 0.01) *       # возрастной коэффициент
            np.sqrt(self.inventory_data['height'] * 0.1)        # коэффициент по высоте
        )

        # Применяем коэффициенты по породам деревьев
        for species, factor in species_factors.items():
            mask = self.inventory_data['species'].str.contains(species, case=False, na=False)
            self.inventory_data.loc[mask, 'carbon'] *= factor

        self.carbon_data = self.inventory_data.copy()

        if self.gdf is not None:
            self.gdf['carbon'] = self.carbon_data['carbon']

        # Базовая статистика
        mean_carbon = self.carbon_data['carbon'].mean()
        max_carbon = self.carbon_data['carbon'].max()
        total_carbon = self.carbon_data['carbon'].sum()

        return (f"Расчёт накопления углерода выполнен. "
```

```
                f"Среднее значение: {mean_carbon:.2f} т/га, "
                f"Максимальное значение: {max_carbon:.2f} т/га, "
                f"Общий запас углерода: {total_carbon:.2f} т")

        except Exception as e:
            return f"Ошибка при расчете накопления углерода: {str(e)}"

    def create_carbon_map(self, output_path: str, color_scheme: str = 'viridis') -> str:
        """Создает карту депонирования углерода и сохраняет ее в файл"""
        if self.gdf is None or 'carbon' not in self.gdf.columns:
            return "Необходимо загрузить геоданные и рассчитать накопление углерода"

        try:
            # Создаем карту
            fig, ax = plt.subplots(figsize=(12, 10))

            # Строим тепловую карту
            self.gdf.plot(column='carbon', cmap=color_scheme, legend=True,
                          legend_kwds={'label': "Накопление углерода (т/га)"},
                          ax=ax)

            # Добавляем заголовок и настраиваем внешний вид
            plt.title('Карта депонирования углерода лесными массивами', fontsize=15)
            plt.axis('on')
            plt.grid(True, linestyle='--', alpha=0.5)

            # Сохраняем карту
```

```
            plt.savefig(output_path, dpi=300, bbox_inches='tight')
            plt.close()

            return f"Карта депонирования углерода сохранена в {output_path}"

        except Exception as e:
            return f"Ошибка при создании карты: {str(e)}"

    def create_raster_map(self, output_path: str, resolution: float = 100.0) -> str:
        """Создает растровую карту депонирования углерода"""
        if self.gdf is None or 'carbon' not in self.gdf.columns:
            return "Необходимо загрузить геоданные и рассчитать накопление углерода"

        try:
            # Определяем границы данных
            bounds = self.gdf.total_bounds

            # Рассчитываем размеры растра
            width = int((bounds[2] - bounds[0]) / resolution)
            height = int((bounds[3] - bounds[1]) / resolution)

            # Создаем растр
            transform = from_origin(bounds[0], bounds[3], resolution, resolution)

            # Растеризуем геоданные
            raster = features.rasterize(
                [(geom, value) for geom, value in zip(self.gdf.geometry, self.gdf.carbon)],
```

```
                out_shape=(height, width),
                transform=transform,
                fill=0,
                all_touched=True,
                dtype=np.float32
            )

            # Сохраняем растр в GeoTIFF
            with rasterio.open(
                output_path,
                'w',
                driver='GTiff',
                height=height,
                width=width,
                count=1,
                dtype=raster.dtype,
                crs=self.gdf.crs,
                transform=transform
            ) as dst:
                dst.write(raster, 1)

            return f"Растровая карта депонирования углерода создана и сохранена в {output_path}"

        except Exception as e:
            return f"Ошибка при создании растровой карты: {str(e)}"

    def get_carbon_statistics(self) -> Dict:
        """Возвращает статистику по накоплению углерода"""
```

```
        if self.carbon_data is None or 'carbon' not in self.carbon_data.columns:
            return {"error": "Сначала выполните расчёт накопления углерода"}

        stats = {
            "mean_carbon": float(self.carbon_data['carbon'].mean()),
            "median_carbon": float(self.carbon_data['carbon'].median()),
            "max_carbon": float(self.carbon_data['carbon'].max()),
            "min_carbon": float(self.carbon_data['carbon'].min()),
            "total_carbon": float(self.carbon_data['carbon'].sum()),
            "std_dev": float(self.carbon_data['carbon'].std())
        }

        # Если есть данные о породах, добавляем статистику по породам
        if 'species' in self.carbon_data.columns:
            species_stats =
self.carbon_data.groupby('species')['carbon'].mean().to_dict()
            stats["carbon_by_species"] = {k: float(v) for k, v in species_stats.items()}

        return stats

# Создаем экземпляр обработчика данных
forest_handler = ForestInventoryHandler()

# Определяем инструменты (tools) для LangChain агента

class LoadDataInput(BaseModel):
    file_path: str = Field(description="Путь к файлу с таксационными данными
(CSV или SHP)")
```

```
class CreateMapInput(BaseModel):
    output_path: str = Field(description="Путь для сохранения карты")
    color_scheme: Optional[str] = Field(default="viridis", description="Цветовая схема карты")

class CreateRasterInput(BaseModel):
    output_path: str = Field(description="Путь для сохранения растровой карты")
    resolution: Optional[float] = Field(default=100.0, description="Разрешение растра в метрах")

# Создаем инструменты для агента
tools = [
    StructuredTool.from_function(
        func=forest_handler.load_data,
        name="load_forest_data",
        description="Загружает данные таксации леса из файла CSV или SHP",
        args_schema=LoadDataInput
    ),
    StructuredTool.from_function(
        func=forest_handler.calculate_carbon_sequestration,
        name="calculate_carbon",
        description="Рассчитывает накопление углерода на основе таксационных данных"
    ),
    StructuredTool.from_function(
        func=forest_handler.create_carbon_map,
        name="create_carbon_map",
        description="Создает векторную карту депонирования углерода лесом",
        args_schema=CreateMapInput
    ),
```

```
    StructuredTool.from_function(
        func=forest_handler.create_raster_map,
        name="create_raster_map",
        description="Создает растровую карту депонирования углерода",
        args_schema=CreateRasterInput
    ),
    StructuredTool.from_function(
        func=forest_handler.get_carbon_statistics,
        name="get_carbon_statistics",
        description="Получает статистику по накоплению углерода"
    )
]

# Создаем шаблон промпта для агента
prompt = ChatPromptTemplate.from_messages([
    ("system", """Ты - специализированный ассистент по анализу депонирования углерода лесными экосистемами.
    Ты помогаешь пользователям загружать данные таксации леса, анализировать их, рассчитывать накопление углерода и создавать картографические материалы.

    Твои знания:
    1. Таксация леса - это совокупность характеристик лесов (возраст, высота, объем, породный состав и т.д.)
    2. Депонирование углерода - это процесс накопления углерода в биомассе лесов
    3. Разные породы деревьев имеют разные коэффициенты накопления углерода

    Сначала пользователь должен загрузить данные, затем выполнить расчет, и только после этого создавать карты.
```

```
    Давай подробные объяснения своих действий и рекомендации по интерпретации результатов."""),
    MessagesPlaceholder(variable_name="chat_history"),
    ("human", "{input}"),
    MessagesPlaceholder(variable_name="agent_scratchpad")
])

# Создаем память для агента
memory = ConversationBufferMemory(memory_key="chat_history", return_messages=True)

# Создаем LLM
llm = ChatOpenAI(model="gpt-4", temperature=0)

# Создаем агента
agent = create_openai_functions_agent(llm, tools, prompt)
agent_executor = AgentExecutor(
    agent=agent,
    tools=tools,
    memory=memory,
    verbose=True,
    return_intermediate_steps=True
)

# Примеры использования агента
result = agent_executor.invoke({"input": "Помоги мне создать карту депонирования углерода. Какие шаги нужно сделать?"})
print(result["output"])
```

Пример использования агента

PYTHON

```
# Загрузка данных
result = agent_executor.invoke({"input": "Загрузи файл с таксационными данными forest_inventory.csv"})
print(result["output"])

# Расчет накопления углерода
result = agent_executor.invoke({"input": "Теперь рассчитай накопление углерода по этим данным"})
print(result["output"])

# Получение статистики
result = agent_executor.invoke({"input": "Покажи мне статистику по накоплению углерода"})
print(result["output"])

# Создание карты
result = agent_executor.invoke({"input": "Создай карту депонирования углерода и сохрани ее как carbon_map.png с цветовой схемой 'plasma'"})
print(result["output"])

# Создание растровой карты
result = agent_executor.invoke({"input": "Создай растровую карту с разрешением 50 метров и сохрани как carbon_raster.tif"})
print(result["output"])

# Задание вопроса для интерпретации
result = agent_executor.invoke({"input": "Какие участки леса наиболее эффективны в накоплении углерода на основе нашего анализа?"})
print(result["output"])
```

Особенности агента:

1. **Многофункциональность**: агент умеет загружать данные таксации, рассчитывать накопление углерода и создавать различные типы карт
2. **Интеллектуальная обработка**: использует различные коэффициенты для разных пород деревьев
3. **Визуализация**: создает как векторные, так и растровые карты депонирования углерода
4. **Анализ данных**: предоставляет статистику по накоплению углерода
5. **Консультации**: помогает интерпретировать результаты анализа

Этот пример можно расширить дополнительными функциями, такими как временной анализ изменений накопления углерода, прогнозирование будущего депонирования на основе моделей роста леса, и интеграция с другими геоданными.