AINER

Ссылки проекта

- https://posta-map.ru/ - главная страница
- https://posta-map.ru/openapi.json - конфигурация свагера
- https://posta-map.ru/docs - свагер
- https://disk.yandex.ru/d/Y7HtHY_975v88g - датасеты
- https://disk.yandex.ru/d/AuP16fsHNOOeHw - презентация

Описание

Приложение позволяет выборать оптимальные точки размещения постаматов. Опирается на открытые данные.

Функционал

- Просмотр карты
- Фильтрация
- Экспорт в .csv
- Расчет на лету
- Быстрый снимок
- Мощное встраиваемое API

Основные файлы проекта

- analytics/ - файлы с аналитикой
- .env - настройки проекта (пример заполнения .env.example)
- backend/app.py, backend/app/ - FastAPI приложение
- backend/load_data.py, backend/loaders - Загрузчики данных
- backend/datasets/data_src - входные датасеты, с которыми работают загрузчики. **Обратить внимание! Датасеты в репозиторий не включены, поскольку гитхаб не дает заливать столь большие файлы. Их стоит скачать с яндекс диска https://disk.yandex.ru/d/Y7HtHY_975v88g и распаковать архив в backend/datasets/data_src**
- backend/datasets/extrapolated_houses.json - результат работы нейросети, которая экстраполировала данные жилых домов.
- backend/datasets/data_src/dadata - результат работы сервиса **Spiders**
- backend/store - работа с БД

- backend/services/candidates.py - расчет кандидатов
- backend/train_analytic_distance_rate_model.py, backend/analytic - нейросеть из заголовка **Загрузка кандидатов**
- frontend - React фронтенд проекта
- share - Монтируемая в docker volume директория, там лежит статика, а также падают дампы Spiders

Docker сервисы

- loaders
- spiders
- backend
- build_frontend
- pgweb **Обратить внимание! - локальной базы нет. Так как данные огромных размеров, сразу использовалась продовская, иначе на повторное заполнение ушло бы приличное кол-во времени.**

Прод

На продакшен сервере хостится все через nginx

Внутренние backend блоки:

- loaders
- spiders
- app
- train distance rate model

Spiders

Скрипты, целью которых является сбор данных. На текущий момент - лишь один скрипт - скрипт получения домов с **DaData**. - Сервис предоставляет полную функцию геокодирования, которую мы могли бы использовать, если бы это было бесплатно - что не так. Однако их апи предоставляет возможность отправить координаты и по ним получить дома поблизости, и таких запросов дают аж 10000 в день на аккаунт. В итоге мы создали порядка 20 аккаунтов, и с помощью скрипта прошлись почти по каждой точке Москвы, и так получили необходимые дома с адресами и координатами, которые в последствии отправили в датасеты.

Необходимость всего этого следовала из того, что вся логика у нас заточена на работу с координатами, а в датасете домов есть только адреса.

Loaders

Скрипты, которые подгружают данные в БД, составляют реляционные связи между ними, а также выполняют первичный анализ данных. Данные черпают из датасетов. По итогу скриптов есть заполненные в БД таблицы. У каждого объекта есть свой модификатор - его значение также формируется загрузчиком. Всего насчитывается 14 штук:

- загрузка округов
- загрузка районов
- загрузка метро
- загрузка постаматов
- загрузка парковок
- загрузка остановок
- загрузка домов культуры
- загрузка библиотек
- загрузка спорт. объектов
- загрузка бумажных киосков
- загрузка остальных киосков
- загрузка мфц
- загрузка жилых домов
- и наконец загрузка кандидатов (кандидат - точка, потециальный постамат)

На каждый из них есть отдельный скрипт, у каждого своя, но схожая с остальными логика. Выделяются лишь некоторые:

- округа и районы. - сразу в датасетах есть поля "rate" - это рейтинг, который мы заполнили сами исходя из аналитики, расчитанной в excel
- дома. чтобы заполнить таблицу с домами, потребовалось геокодировать адреса и экстраполировать недостающие данные. Данные об адресах из DaData (см. **Spiders**). Недостающие данные достраивались искуственным интеллектом, который в свою очередь обучался на этом же датасете, - но на заполненных
- кандидаты - отдельный следующий заголовок

Загрузка кандидатов

Кандидат - точка, для которой был и будет расчитан конечный показатель рейтинга.

В качестве набора точек используем все, уже ранее загруженные объекты (их порядка 55 тысяч). Значит для каждой точки нам нужно получить агрегирующие показатели по каждому типу объектов вокруг, а затем использовать средневзвешенную модель для расчета итогового рейтинга. Первым серьезным вопросом стал: "Как понять, что лучше ? А тем более объяснить компьютеру ?" - Здесь подразумевается то, что проще показать на примере:

Рассмотрим несколько наборов данных об близстоящих метро:

- *Сама точка находится в метро, но метро не сильно то популярное (соотв. модификатор низкий)*
- *У точки 5 разных метро в радиусе расчета, но не ближе, чем 100м. И все они имеют хороший модификатор*
- *У точки 1 метро через 50м и еще 3 через 500м*

Так у какого из этих трех вариантов показатель по метро должен быть выше ? Сначала мы пытались составить какую-то формулу, но быстро бросили это дело, отдав предпочтение нейросети. Чтобы натренировать ее, она генерирует нам случайные точки в случайном радиусе со случайными модификаторами, а мы выдаем ей результирующий показатель. Так нейросеть достаточно быстро нашла закономерность и сейчас справляется со своей задачей хорошо.

В итоге, прогоняя пул объектов каждого типа по нейросети, мы получали новый единый коэфициент для объектов одного типа.

Итак теперь у нас есть общие модификаторы для всех учитываемых объектов. - Дело за малым, - осталось добавить взвешенные веса. Тут появилось две модели расчета. - первая основана на личной аналитике происходящего. Вторая же на пользовательском опросе.

Пара ремарок в отношении модификаторов - все их мы нормализовывали, чтобы были в промежутке [0; 1]. (Тем самым мы в будущем облегчим работу новым нейросетям, которые появятся по итогам MVP вместо наших моделей расчета.)

Конечная формула расчета рейтинга кандидата выглядит примерно так:

$$F_{(\cdots)} = \frac{1}{\sum_{i=1}^{n} w_{im}} \sum_{i=1}^{n} w_{i_m} \cdot NDR\Big(G(p,i,r)\Big)$$

Где *n* - кол-во учитываемых типов объектов (14), *i* - номер типа объекта, **w_i_m** - агрегационный вес объектов *i*-того типа, *m*-той модели расчета. *G(p, i, r)* - функция получения всех объектов *i*-типа в радиусе *r* с центром в точке *p*, *NDR* - предикт нейросети (возвращает одно число).