Финальный проект — разработка сервиса на основе моделей ранжирования с использованием полученных знаний. В качестве бонуса обсудим задачи, которые можно решить с минимальными изменениями текущего пайплайна.

В качестве финального проекта необходимо реализовать систему подсказок похожих вопросов на уже знакомых вам данных сайта Quora. Поиск производится исключительно по основному вопросу (заголовку) без уточняющих деталей (доп. описания).

Система представлена микросервисом на основе Flask. Для задания не предусмотрен шаблон, вам предоставлена максимальная вольность в реализации. Верхнеуровнево пайплайн и критерии можно представить так:

Сначала происходит фильтрация запроса по языку (с помощью библиотеки LangDetect) — исключаются все запросы, для которых определённый язык не равняется "en". Затем происходит поиск вопросов-кандидатов с помощью FAISS (по схожести векторов) — в этой части предлагается ограничиться векторизацией только тех слов, эмбеддинги которых есть в исходных GLOVE-векторах. Эти кандидаты реранжируются KNRM-моделью, после чего до 10 кандидатов выдаются в качестве ответа.

Для ускорения проверки и уменьшения времени итераций вам не нужно загружать полноценную KNRM-модель и эмбеддинги. Для вашего решения в качестве переменных окружения, в котором происходит запуск, предлагаются пути до следующих файлов:

EMB\_PATH\_KNRM — матрица эмбеддингов KNRM, полученная в рамках решения ДЗ к 5-й лекции (torch.save(Solution(...).model.embeddings.state\_dict())). Порядок токенов в ней соответствует задаваемому JSON-словарём VOCAB\_PATH. В нём ключ — это слово (препроцессинг аналогичен ДЗ 5), значение — индекс в матрице эмбеддингов. MLP-часть вы должны обучать и загружать в грейдер самостоятельно — путь до загруженного в систему файла находится в переменной среды MLP\_PATH (если вы не понимаете, почему возможно запускать ваш MLP над обученными другим человеком эмбеддингами и как это должно работать, вернитесь к лекции 5). Сырые эмбеддинги GLOVE (к примеру, для векторизации при поиске соседей — чтобы знать, какие вектора инициализированы случайно, а какие были предобучены) доступны в EMB\_PATH\_GLOVE.

Ваш сервис будет запущен командой FLASK\_APP=user\_input/solution.py flask run --port 11000, где solution.py — загружаемый вами файл решения со всей необходимой логикой. На странице сабмита доступен Dockerfile с описанием окружения для запуска (можете сверить версии библиотек для локальной отладки).

На загрузку эмбеддингов, создание моделей, словарей и т.д. будет предоставлено 120 секунд. В этот момент вам не доступны текстовые данные, потому о создании индекса для поиска соседей речь не идёт.

В течение этих 120 секунд будут отправляться запросы на ручку /ping (под ручкой подразумевается веб-страница по адресу 127.0.0.1:11000/ping). Привязать метод питона к запросу по веб-адресу можно с помощью декоратора перед основным методом с логикой, например: @app.route('/ping') (см. примеры в [документации Flask](https://flask.palletsprojects.com/en/1.1.x/tutorial/layout/)).

Как только сервер ответит json'ом с ключом status и значением ok, начнётся проверка решения. Обратите на это внимание: ручка не должна отвечать ok до тех пор, пока не закончится загрузка моделей и прочее.  Здесь и далее подразумевается, что все данные будут поступать и должны отдаваться в json-виде. Получить их можно, к примеру, так: content = json.loads(request.json), где request — это объект из библиотеки Flask.

Далее необходимо реализовать две ручки: для запросов (для поиска похожих вопросов) и для создания FAISS-индекса.

/query — принимает POST-запрос. Должна вернуть json, где status='FAISS is not initialized!' в случае, если в решение не были загружены вопросы для поиска с помощью второго метода.

Формат запроса для query:

json-запрос, с единственным ключом 'queries', значение которого — список строк с вопросами (Dict[str, List[str]]).

Формат ответа (в случае созданного индекса) — json с двумя полями. lang\_check описывает, был ли распознан запрос как английский (List[bool], True/False-значения), suggestions — List[Optional[List[Tuple[str, str]]]].

В этом списке для каждого запроса из query необходимо указать список (до 10) найденных схожих вопросов, где каждый вопрос представлен в виде Tuple, в котором первое значение — id текста (см. ниже), второе — сам непредобработанный текст схожего вопроса. Если проверка на язык не пройдена (не английский), либо произошёл какой-то сбой в обработке — оставьте None в списке вместо ответа (например, [[(..., ...), (..., ...), ...], None, ... ]).

/update\_index — принимает POST-запрос, в котором в json присутствует поле documents, Dict[str,str] — все документы, где ключ — id текста, значение — сам текст. На предобработку и создание индекса даётся 200 секунд. Подразумевается, что инициализация происходит единоразово, поэтому не нужно беспокоиться о повторном вызове этого метода. В возвращаемом json'е должно быть два ключа: status (ok, если всё прошло гладко) и index\_size, значение которого — единственное целое число, хранящее количество документов в индексе.

Решение будет оцениваться на валидационном сете запросов, не входящих в тренировочную выборку. Метрика — доля запросов, для которых в топ-10 был выведен как минимум один истинный дубликат вопроса (подразумевается, что для каждого англоязычного запроса такой кандидат есть).

Сейчас порог установлен на уровне 0.5, но он может (и будет) меняться (в том числе в большую сторону) в зависимости от отправленных решений.

Таким образом, от вас ожидаются два файла:

solution.py — файл с веб-сервисом на Flask и всей необходимой логикой работы.

knrm\_mlp.bin — бинарный файл с весами верхней части KNRM для реранжирования кандидатов.

Желаем успехов!

**> Сабмит**

Для финального проекта шаблон ДЗ не предоставляется.

[Ссылка на Dockerfile](https://disk.yandex.ru/d/wAe2O3sFL3jrTw) с описанием окружения, в котором развёртывается сервис.