МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ С РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМОЙ НА ОСНОВЕ МАШИННОГО ОБУЧЕНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Дёрова Василия Андреевича

Научный руководитель
доцент, к. ф.-м. н. _____ Ю. Н. Кондратова

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2021

ВВЕДЕНИЕ

Машинное обучение (Machine Learning) — обширный подраздел искусственного интеллекта, который имеет огромное влияние на современный мир.

Одной из задач машинного обучения является создание умных рекомендательных систем, способных предлагать пользователю что-нибудь новое на основании уже известных предпочтений других людей. Такие системы всегда актуальны, как и поиск и проработка способов их создания, потому что они помогают улучшить опыт пользователя, а также являются отличным вспомогательным инструментом для бизнеса.

В эпоху сети Интернет большое количество задач можно решать без установки дополнительных программ на компьютер. Современные веб-страницы являются полноценными приложениями, которые обеспечивают широкий выбор способов взаимодействия с пользователем. Такие приложения удобны в использовании и зачастую более доступны, чем классические компьютерные программы.

Интеграция моделей машинного обучения в веб-приложения является популярным и актуальным решением и помогает предоставить все преимущества данной подобласти искусственного интеллекта в удобной и доступной форме.

Существует множество подходов и инструментов для создания современных веб-приложений. Многие инструменты делятся на две основные категории: для создания серверной части веб-приложения, также известной как бэкенд (back-end), и для создания интерфейса пользователя, или фронтенд (front-end).

Целью данной работы является создание веб-приложения, способного на основе вопросов о взглядах человека на жизнь предложить хобби и виды деятельности, которые могут ему понравиться.

Для достижения этой цели необходимо решить следующие задачи:

- подготовить входные данные для создания модели;
- проанализировать данные и подобрать методы построения рекомендательной системы;
- испытать выбранные модели и подобрать лучшие параметры;
- сравнить качество моделей и выбрать лучшие;
- сохранить лучшие обученные модели для использования в веб-приложении;

- создать серверную часть веб-приложения опросника, используя возможности фреймворка Flask;
- интегрировать рекомендательную систему в бэкенд веб-приложения;
- создать пользовательский интерфейс веб-приложения опросника, используя возможности React.

Основное содержание работы

Для построения рекомендательной системы используются данные опроса молодых людей с сайта kaggle.com. Опрос проводился по 150 категориям, 139 из которых — целочисленные, 11 — категориальные, всего было получено 1010 ответов, категории могут быть разделены на различные тематические группы. Данные сохранены в формате .csv, в ответах присутствуют пропуски.

На основе этих данных необходимо построить систему, способную выдавать рекомендации по группе «Хобби и интересы», используя в качестве задаваемых пользователю вопросов группу «Черты характера, взгляды на жизнь и убеждения».

Построение модели проводится на языке Python. Используются библиотеки `pandas` — для работы с данными в .csv таблице, `NumPy` — для работы с большими массивами, `matplotlib` — для вывода графиков, `scikit-learn` — для использования оптимизированных алгоритмов и инструментов машинного обучения, `joblib` — для сохранения обученной модели на диск, `skmultilearn` — для создания сбалансированных обучающих и тестовых выборок в задаче многозначной классификации. Программа пишется в среде `Jupyter Notebook`.

Для решения задачи используется подход с обучением классификаторов.

Результаты опроса сохранены в формате `.csv`, работу с таким форматом поддерживает библиотека `pandas`. Данные считываются в объект `DataFrame`, который является специальной структурой данных, используемой в библиотеке.

После считывания данные нужно обработать.

Для замены строковых значений числовыми была выбрана следующая стратегия: текстовые значения в столбцах заменяются на числа — в порядке субъективного возрастания значимости ответа, если такое возможно. Если же у значений нет смысловой нагрузки, которую можно «отсортировать», то числа в порядке возрастания присваиваются произвольным ответам.

Стратегия заполнения пропущенных значений заключается в том, что для колонок «Возраст», «Вес» и «Рост» вместо пустых ответов добавляется округленное среднее значение по столбцу. Для колонок, которые отвечают за целевые признаки, а именно хобби и интересы, пропуски заполняются минимально возможным значением в этих столбцах — 1. Такое решение связано с тем, что можно предположить, что если человек не выбрал степень свое-

го увлечения определенным хобби, то это хобби его не интересует. Для всех остальных колонок пустые значения заполняются модой.

Проводится анализ корреляции между предсказываемыми признаками и теми, которые используются для получения этих предсказаний. Построение матрицы корреляции происходит инструментами библиотеки `pandas` при помощи использования коэффициента корреляции Пирсона. Для каждой черты характера выводится информация о двух самых позитивно и двух самых негативно коррелирующих хобби.

Целевые признаки отделяются от использующихся при получении предсказания и преобразовываются в формат, подходящий для решения задачи (массив из библиотеки `NumPy`). Для выбора нужных признаков используются списки, в которых содержатся названия всех интересующих столбцов. В данном случае это список `personality`, в котором хранятся значения всех колонок, отвечающих за черты характера респондента, и `hobbies`, где хранятся названия колонок с хобби и увлечениями.

После получения массивов `NumPy` значения в массиве `y`, который содержит целевые признаки, изменяются по правилу: все значения больше 3 становятся равными 1, а меньшие либо равные — 0.

Следующим шагом удаляются так называемые «выбросы» — ответы, которые выделяются на фоне остальных.

Первым подходом для выявления выбросов является выделение строк данных, где человек указал высокую заинтересованность абсолютно во всех хобби из 32, а также строк, где человек не отметил ни одного интереса. Это достигается путем подсчета суммы вдоль каждой строки. Если сумма равна 0 — в каждом столбце для данного респондента указан 0. Если же сумма равна количеству хобби, то это означает, что в каждой колонке содержится значение 1.

Вторым этапом удаления выбросов является использование встроенного в `scikit-learn` алгоритма локального уровня выброса (local outlier factor) на массиве с чертами характера респондентов.

После работы алгоритма для каждой строки считается коэффициент, чем он ближе к -1, тем менее вероятно, что строка является выбросом. В данном случае для удаления выбросов было решено выбрать порог, равный -1.4. Все строки со значением коэффициента ниже этого порога удаляются из массивов.

В сумме из данных было удалено 19 выбросов.

Можно посмотреть, сколько значений каждого из классов встречается в каждом из столбцов с целевыми признаками. Для этого используется структура данных `Counter` из коллекций Python. Данные, как показывает использование `Counter`, достаточно несбалансированные, причем как и в сторону положительного класса (класс 1), так и в сторону отрицательного (класс 0).

В работе было решено попробовать подход, при котором целевые признаки делятся в 3 категории: те, которые содержат преимущественно отрицательный класс, те, которые содержат преимущественно положительный, и те, в котором соблюден относительный баланс между классами. Порогом для определения баланса была выбрана разница между количеством элементов разных классов менее 200.

Для выделения категорий создаются специальные массивы-маски. Это позволяет, например, посмотреть, какие именно хобби оказались в каждой из категорий.

Для большего представления о том, какие именно признаки и как сильно влияют на определение степени заинтересованности человека тем или иным хобби, для каждой категории подсчитывается коэффициент важности признака при принятии решения классификатором `RandomForestClassifier`.

Можно говорить о том, что для каждой из категорий существует небольшое количество признаков, которые больше всего влияют на принятие решения классификатором, а разброс между оценкой важности признаков недостаточно большой, чтобы можно было убрать какие-нибудь из вопросов, которые будут задаваться пользователю для получения предсказаний.

Оценивается, сколько в каждой из категорий хобби респондентов, которым нравится или не нравится вся группа целиком. Количество людей, которые имеют все самые популярные хобби сразу — достаточно большое, а именно 297.

Для возможности классификаторов более точно распознавать случаи, когда рекомендовать популярные увлечения пользователю не следует, решено убрать из данных некоторое количество строк, которые соответствуют пользователям, положительно оценившим все популярные хобби сразу.

Стратегия для удаления ответов была выбрана следующая: из массива хобби с преобладающим классом 1 удаляется 80% случайным образом вы-

бранных строк из тех, в которых все ответы сразу являются положительными. Это соответствует 238 ответам.

На этом этап обработки данных закончен, остается только разделить выборку на тренировочную и тестовую. Данные являются преимущественно несбалансированными, а каждая из подзадач является задачей многозначной классификации, поэтому для разделения данных было решено использовать специальный алгоритм `iterative train test split` из библиотеки `scikit-multilearn`.

Разбиение проводится отдельно для каждой из групп хобби, а также для всего набора. Это требуется для обучения разных классификаторов и их оценки на каждой из категорий.

После разбиения каждая из тестовых и обучающих выборок признаков стандартизируется. Это рекомендуемый шаг, хотя он требуется не всегда и не для всех видов классификаторов.

Первым подходом для получения предсказаний был выбран достаточно простой — базирующийся на оценке похожести пользователей с помощью косинусного расстояния. Строится матрица схожести каждого респондента из тестового набора с респондентами из тренировочного по их чертам характера, затем для каждого тестового случая предсказывается рекомендация целевых признаков — как округленное среднее всех ответов 150-и самых похожих респондентов для каждого целевого признака.

Оценки получаются не слишком хорошие. Хотя минимальная точность и полнота на полном наборе хобби выше нуля, это можно объяснить тем, что в каких-то из столбцов среднее всегда с большим шансом будет равняться либо 0, либо 1 ввиду несбалансированности данных, поэтому существует высокая вероятность, что эти значения в тестовом наборе (в котором также присутствует несбалансированность) будут просто часто угадываться. Это подтверждается при применении метода к сбалансированным категориям хобби: оценки сразу резко падают. Проверять подобный способ на двух оставшихся категориях нецелесообразно, потому что там шанс угадать самый популярный ответ еще выше.

Следующий отчет строится для оценки классификатора случайного леса «из коробки». Под этим подразумевается, что классификатор используется в его реализации по умолчанию — большинство гиперпараметров остаются

нетронутыми.

Оценки на сбалансированном наборе стали лучше по сравнению с предыдущим подходом. Также стоить отметить, что средняя точность и полнота при предсказывании хобби из категории с преобладающим количеством значения 0 получились достаточно низкими. Это говорит о том, что классификатор «из коробки» плохо справляется с предсказыванием положительного класса на этом наборе данных. В случае с набором, где преобладают значения 1, за положительный класс для подсчета точности и полноты берется 0, потому что на этой группе интерес представляют предсказания именно данного класса. Оценки уже являются достаточно неплохими, что можно объяснить тем, что из этого набора предварительно было удалено много вхождений с преобладающими значениями 1 и, как итог, способность предсказывать отрицательный класс улучшилась даже на базовом классификаторе.

Последним отчетом на данном этапе работы будет оценка классификатора на основе полносвязной нейронной сети (многослойного перцептрона) «из коробки». Единственным существенным отличием в созданном классификаторе от значения по умолчанию является оптимизатор: он был установлен в значение `'lbfgs'` для ускорения и гарантии сходимости (по умолчанию используется оптимизатор `'adam'`).

На основании оценок можно судить о том, что в среднем такой классификатор выдает результаты хуже, чем основанный на алгоритме случайного леса, но гораздо меньше переобучен в ситуации, когда класс 0 преобладает.

Завершающим этапом для создания рабочей модели является тонкая настройка — подбор гиперпараметров модели. Подбор осуществляется сразу для двух групп классификаторов: 3 классификатора на основе алгоритма случайного леса, каждый из которых предсказывает одну из категорий, на которые были разбиты все целевые признаки, и 3 — на основе многослойного перцептрона.

Для подбора параметров решено использовать оценку `F1`, потому что она основана как на полноте, так и на точности, а именно эти две оценки представляют интерес в рамках решаемой задачи. Для предсказания категорий хобби, входящих в группу с преобладающим классом 1, в качестве положительного класса для получения оценки `F1` считается класс 0. В остальных категориях положительному классу соответствует 1.

Стандартная реализация оценщиков в `scikit-learn` не подходит для

решения поставленной задачи, поэтому функции для подсчета оценок создаются вручную.

Каждая из используемых оценок (`F1` и доля правильных ответов) определяется следующим образом: одно полное предсказание модели состоит из последовательности предсказанных значений для каждого из целевых признаков в категории, оценка `F1` и доля правильных ответов вычисляется между таким полным предсказанием и истинной последовательностью значений целевых признаков. Итоговая оценка строится как среднее от всех индивидуальных оценок, полученных при сравнении предсказанной и истинной последовательности ответов.

Помимо адаптированного оценщика, для использования метода подбора гиперпараметров из `scikit-learn` задается нестандартная функция для реализации стратегии кросс-валидации. В рамках программы используется специальный объект-итератор из библиотеки `scikit-multilearn`, который называется `IterativeStratification`. Он позволяет делать деления выборки во время кросс-валидации с сохранением соотношения классов для многозначных задач, как это было сделано для разбиения выборки на тренировочные и проверочные данные ранее.

Перед запуском алгоритма подбора гиперпараметров для базового классификатора выводятся обе заданные оценки (`F1` и доля правильных ответов) для каждой из групп целевых признаков на кросс-валидации, чтобы можно было судить об их улучшении после завершения тонкой настройки.

Для подбора гиперпараметров используется перебор по сетке с кросс-валидацией (`GridSearchCV`) из библиотеки `scikit-learn`.

Первым происходит тонкая настройка классификатора случайного леса. В результате работы поиска гиперпараметров по сетке для каждой из моделей был подобран уникальный набор гиперпараметров.

Еще раз подсчитываются оценки на кросс-валидации для каждой группы — на этот раз на классификаторах с подобранными параметрами.

Оценка доли правильных ответов упала на каждой категории целевых признаков, причем в случае категории с преобладающим классом 0 — достаточно заметно. При этом оценка `F1` возросла на каждой категории, и особенно существенно на группе с преобладающим классом 0. Это говорит о том, что была уменьшена степень переобучения классификаторов на преобладающий

класс — результат, которого и требовалось достичь.

Весь процесс повторяется для классификатора на основе многослойного перцептрона.

Стоить отметить, что у базовой версии классификатора предварительно заданы вручную такие гиперпараметры как `learning_rate_init=0.02`, `early_stopping=True` и `n_iter_no_change=50`. Это сделано для того, чтобы во время подбора параметров гарантировалась сходимость оптимизатора `'adam'`.

В данном случае также получились модели с разными подобранными параметрами — вплоть до различного количества скрытых слоев.

Наблюдается ситуация схожая с той, которая прослеживалась после тонкой настройки классификатора случайного леса: доля правильных ответов немного падает, в то время как оценка `F1` возрастает. Для категории с преобладающим классом 0 вновь показывается самый значительный прирост оценки `F1`, но при этом доля правильных ответов не падает так же сильно, как это было в случае классификатора случайного леса.

Для каждого из классификаторов строятся отчеты об оценках на тестовых данных, аналогичные тем, что создавались ранее для моделей «из коробки». После рассмотрения всех конечных результатов на тестовых выборках было решено использовать для каждой категории классификатор на основе многослойного перцептрона. Обученные на всем доступном и обработанном наборе данных модели сохраняются на диск.

Одним из современных подходов к разработке веб-приложений является подход, который строится на принципе разделения ответственности. В этом подходе приложение делится на две части: клиентская часть, или фронтенд, — та часть приложения, которая отвечает за взаимодействие с пользователем; программно-аппаратная часть, или бэкенд, — отвечает за функционирование серверной части.

Основными технологиями для разработки фронтенд являются HTML, CSS и JavaScript (и все сопутствующие библиотеки).

Для бэкенд это может быть любой универсальный язык программирования (Java, Ruby, Python, PHP) и системы управления базами данных (MySQL, PostgreSQL, MongoDB).

Одним из фреймворков для создания бэкенда является `Flask`, который

позволяет создавать веб-приложения на языке Python для применения модели на сервере. Данный фреймворк идеально подходит для решения поставленных задач, так как бэкенд часть приложения будет иметь достаточно небольшой функционал — хранить список вопросов, который будет выводиться пользователю, а также получать и отправлять результат работы модели машинного обучения.

`React` был выбран для написания клиентской части веб-приложения, потому что он удобен, гибок и достаточно прост в освоении.

В качестве среды разработки (как для фронтенд, так и для бэкенд) был выбран `WebStorm` от JetBrains.

Для работы серверной части дополнительно понадобились библиотеки `NumPy`, `scikit-learn` и `joblib` — для работы с обученной моделью машинного обучения, а также `python-dotenv` для удобства установки переменных среды.

В первую очередь создается виртуальная среда Python, для того чтобы обеспечить изолированность библиотек, установленных для работы приложения, от других сред и установки Python в рамках операционной системы. Создается и активируется виртуальная среды Python с помощью консоли.

Следующим шагом создается файл `.flaskenv`, в котором будут храниться объявленные переменные среды. Этот файл автоматически загружается при запуске Flask, если установлен пакет `python-dotenv`.

Для запуска приложения можно использовать команду `flask run`.

Для общения сервера и браузера на бэкенд части приложения создаются так называемые «конечные точки» — адрес, по которому нужно обратиться одним из видов запросов (например, POST или GET), чтобы получить определенный ответ. Формат и содержание ответа обычно указывается в документации.

В приложении используются две конечные точки: первая отправляет список вопросов, которые будут задаваться пользователю, а вторая получает ответы пользователя на вопросы, а затем отправляет результат работы модели машинного обучения на этих ответах.

Первая конечная точка доступна по адресу `/api/questions`, она считывает список вопросов, который хранится в отдельном файле, и возвращает его в формате `JSON`. Список вопросов хранится в файле `questions.json`. Этот

формат был выбран для удобства внесения изменений и потому, что он позволяет напрямую отправлять такие данные в ответе сервера — без модификаций.

Файл `questions.json` состоит из поля `Questions`, значением которого является список из пар ключ-значение, где первым ключом в паре является поле `value` — в нем хранится непосредственно сам вопрос, а вторым ключом — `answers`, всего таких пар 57 — по количеству вопросов. В поле `answers` также хранятся пары, ключом является числовое представление ответа, а значением — его описание.

Категория вопросов «Черты характера, взгляды на жизнь и убеждения», которая использовалась при обучении моделей машинного обучения и которая предлагается пользователю для получения предсказания, изначально представлена на английском языке. Для использования этих вопросов в приложении был предварительно выполнен их перевод.

Вторая конечная точка находится по адресу `/api/result`. Эта точка отвечает на запросы по методу `POST`, так как она должна получать список всех ответов пользователя и возвращать результат работы моделей машинного обучения.

Реализация точки получает на вход данные, отправленные вместе запросом к серверу. Сначала данные нужно привести из строкового формата к числовому. Затем список ответов приводится к виду, который подходит для использования в моделях, и происходит стандартизация данных, так как каждая модель также обучалась на стандартизированных значениях. Эти значения используются для получения предсказаний с помощью предобученных моделей, которые загружаются при запуске сервера с использованием библиотеки `joblib`. Каждая модель предсказывает свою категорию хобби. Для каждой категории результатом предсказания является список из вероятностей, соответствующих рекомендации того или иного хобби. Элементы списков вероятностей затем объединяются в пары с элементами списков названий соответствующих хобби, полученный набор пар сортируется по убыванию вероятности, и до 6 наиболее вероятных рекомендаций (минимальная вероятность должна быть не менее $0,5$) образуют список, который отправляется в виде ответа сервера.

Для создания одностраничного приложения с бэкендом на `Flask` используется специальный инструментарий (toolchain) `Create React App`. Сре-

да разработки WebStorm поддерживает создание подобного приложения при создании проекта.

Первое, что требуется сделать после создания, это модифицировать файл package.json, в котором находятся различные параметры конфигурации, таким образом, чтобы можно было запускать бэкенд вместе с фронтендом, а все запросы с нераспознанным адресом перенаправлялись на порт 5000, на котором запущен Flask.

Для создания приложения используются библиотеки Material-UI — для создания дизайна страниц и управления стилями, clsx — для применения стилей к объекту по условию (используется для дизайна страницы), axios — для упрощения написания логики отправки запросов к точкам доступа бэкенда, React Query — для дальнейшего упрощения работы с запросами и полученными из них результатами, React Router — для управления маршрутизацией.

За вывод заголовка отвечает компонент Header.js, а за логику вопроса — компонент Question.js.

Для прорисовки компонента Question.js данные получаются путем отправки запроса GET к бэкенду по конечной точке /api/questions.

Использование так называемого «хука» useQuery позволяет после отправки запроса получить различную информацию о состоянии результата данного запроса. В isLoading хранится состояние, которое обозначает, что данные еще загружаются. В error хранится информация, была ли получена ошибка, и, в случае ее обнаружения, сопутствующее сообщение. В data хранится полученный ответ. Используя isLoading и error, можно обработать ситуацию, когда данные еще не получены или произошла ошибка.

При попытке сменить вопрос до выбора варианта ответа строка «Выберите ответ» подсветится красным, а перехода к следующему вопросу не произойдет.

Также в приложении реализовано динамическое отображение кнопок и вспомогательного текста в зависимости от номера вопроса и выбран ли ответ.

Для реализации логики отображения кнопок используются специальные булевы константы, которые передаются в компонент Question.js как атрибуты, называемые пропсы (props).

Чтобы при обновлении страницы пользователь оставался на последнем вопросе, который был им выбран, а также сохранялся список всех его отве-

тов, приложение сохраняет текущий индекс вопроса и все текущие ответы в `sessionStorage` — специальное хранилище данных браузера. Эти значения, при их наличии в памяти браузера, подгружаются в специальные объекты состояния (state) страницы во время их инициализации. Если же данные в памяти отсутствуют, состояние инициализируется значениями по умолчанию.

Все данные о текущем состоянии страницы хранятся при помощи специальных хуков, которые позволяют запоминать необходимую отображаемую информацию и обновлять ее во время работы с приложением.

Каждое состояние реализуется с помощью двух объектов: первый хранит текущее значение состояния, а второй является функцией, которая позволяет его изменять.

В приложении также реализовано два дополнительных хука, реагирующих на обновление текущего индекса вопроса или списка ответов и задающих наличие или отсутствие вспомогательного текста.

Для обработки событий нажатия на различные кнопки при прохождении опроса созданы специальные функции, которые передаются как пропсы в компонент `Question.js`.

Функция `const handleNextQuestion = ()` обрабатывает переход к следующему вопросу.

Функция `const handlePrevQuestion = ()` обрабатывает возврат к предыдущему вопросу.

Функция `const handleSelectAnswer = e` обрабатывает событие выбора нового варианта ответа в специальном компоненте `RadioGroup` из библиотеки `Material-UI`.

Функция `const handleGetResults = ()` обрабатывает нажатие на кнопку «Посмотреть результат».

Переход между разными страницами происходит при помощи библиотеки `React Router`. Хотя сам по себе шаблон всегда остается один, то есть вся информация выводится на одну и ту же страницу, эта библиотека позволяет управлять состоянием адресной строки и выводом различной информации в зависимости от введеной в нее ссылки.

У приложения есть три основных адреса. Начальный адрес, на котором выводится сам опросник, адрес `/results`, на котором выводится результат прохождения опроса, а также адрес `/404`, куда пользователь перенаправляет-

ся, если он попытается пройти по пути, которого не существует в приложении, или открыть страницу с результатами, пока в localStorage еще нет данных, полученных после прохождения опроса.

Логика перехода между адресами реализуется с помощью специального компонента Switch из библиотеки React Router.

Страница /404 содержит лишь небольшой неформатированный текст, уведомляющий пользователя, что страница не найдена.

После прохождения опроса и успешного получения ответа от сервера, пользователь попадает на страницу /results, где выводятся рекомендации хобби, которые могут понравиться пользователю. Всего выводится до 6 самых вероятных рекомендаций, отсортированных по степени уверенности классификаторов — начиная с самой вероятной в верху списка.

Библиотека React Router также позволяет управлять историей переходов между страницами приложения, для этого в ней реализован специальный хук useHistory(), с помощью которого можно создать объект истории. Добавление перехода в стэк объекта истории производится с помощью метода push().

Для создания оформления в приложении вместо стандартного подхода с прямым использованием языка разметки CSS используется решение на базе библиотеки Material-UI. Оно, в свою очередь, строится на основе решения CSS-in-JS, то есть компиляции JavaScript кода непосредственно в CSS.

В приложении создано три файла со стилями: Question.style.js, Header.style.js и Result.style.js. Первый хранит информацию о стилях для вывода списка вопросов компонентом Question.js, второй отвечает за стили для заголовка главный страницы, а третий — за оформление страницы с результатами.

Один из примеров использования стилей в приложении: при наведении курсора на свободный вариант ответа, он затемняется.

ЗАКЛЮЧЕНИЕ

В ходе написания работы было создано веб-приложение опросник, содержащее 57 вопросов, касающихся различных личностных качеств и взглядов на жизнь человека, и по результатам прохождения опроса дающее пользователю список рекомендаций хобби и увлечений.

Для построения рекомендательной системы, используемой в приложении, были обработаны и подготовлены данные, а также проведен их анализ, чтобы выбрать подходящие методы построения системы. Отобранные модели были протестированы, а затем были подобраны лучшие параметры для каждой из них. Для полученных моделей был проведен сравнительный анализ качества, что позволило выбрать три самые подходящие — на основе классификатора многослойного перцептрона. Эти классификаторы были обучены на всем наборе данных и сохранены на диск. Серверная часть веб-приложения была создана при помощи фреймворка Flask, в ней были реализованы две конечные точки для поступления запросов, а также загружены ранее сохраненные модели машинного обучения, настроен запуск и обработка ими данных с получением предсказания. Был создан пользовательский интерфейс веб-приложения с использованием React и вспомогательных библиотек, представляющий с собой динамический одностраничный опросник, который обменивается информацией с серверной частью приложения — получает список вопросов, отправляет ответы пользователя и выводит рекомендации хобби, полученные в качестве ответа сервера.