МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра технологий программирования

ПАВЛИВ
Елизавета Александровна

РАЗРАБОТКА БОТ-ПРИЛОЖЕНИЯ С СИСТЕМОЙ РЕКОМЕНДАЦИЙ НА ОСНОВЕ МАШИННОГО ОБУЧЕНИЯ

Дипломная работа

Научный руководитель:
кандидат технических наук,
доцент И.С. Войтешенко

Допущена к защите
«___» _____ 2021 г
Зав. кафедрой технологий программирования
доктор технических наук, профессор,
Заслуженный деятель науки РБ А.Н. Курбацкий

Минск, 2021

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

API — программный интерфейс приложения (Application Programming Interface).

CSS — каскадные таблицы стилей (Cascading Style Sheets).

CSV — текстовый формат, предназначенный для представления табличных данных (Comma-Separated Values).

HTML — язык гипертекстовой разметки (HyperText Markup Language).

HTTP — протокол передачи гипертекста (HyperText Transfer Protocol).

JSON — обозначение объектов JavaScript (JavaScript Object Notation).

LINQ — набор технологий, создающих и использующих возможности интеграции запросов в языки программирования платформы .NET Framework (Language Integrated Query).

LUIS — распознавание речи (Language Understanding).

ML — машинное обучение (Machine Learning).

NoSQL — тип базы данных, которая хранит и извлекает данные без необходимости сначала определять ее структуру (Not only SQL).

OData — открытый веб-протокол для запроса и обновления данных (Open Data Protocol).

ONNX — открытая библиотека программного обеспечения для построения нейронных сетей глубокого обучения (Open Neural Network Exchange).

REST — передача состояния представления (Representational State Transfer).

SQL — язык программирования структурированных запросов, применяемый для создания, модификации и управления данными в реляционной базе данных (Structured Query Language).

РЕФЕРАТ

Дипломная работа, 70 с., 36 рис., 26 источников, 7 приложений.

Ключевые слова: СИСТЕМА РЕКОМЕНДАЦИЙ, МАШИННОЕ ОБУЧЕНИЕ, ML.NET, НАБОР ДАННЫХ, МАШИНЫ ФАКТОРИЗАЦИИ, КОЛЛАБОРАТИВНАЯ ФИЛЬТРАЦИЯ, ПРОБЛЕМА ХОЛОДНОГО СТАРТА, КОСИНУСНОЕ СХОДСТВО, БОТ-ПРИЛОЖЕНИЕ, AZURE, CLOUD-ТЕХНОЛОГИИ.

Объект исследования — набор рейтинговых данных о фильмах с веб-сайта MovieLens, алгоритмы машинного обучения для систем рекомендаций, платформа ML.NET, сервисы Azure Bot Service и Bot Framework, сервисы облачных вычислений Azure Cloud Services.

Цель работы — построение модели машинного обучения для системы рекомендаций, проектирование и разработка бот-приложения с использованием Cloud-технологий, умеющего распознавать естественный язык пользователя, и интеграция бот-приложения с системой рекомендаций.

Методы исследования — анализ алгоритмов для построения систем рекомендаций, возможностей платформы ML.NET и сервисов Azure Bot Service, Bot Framework, Azure Cloud Services, проектирование модели машинного обучения для системы рекомендаций, проектирование и разработка бот-приложения, интеграция бот-приложения с системой рекомендаций.

Результатами являются — модель машинного обучения для системы рекомендаций фильмов пользователям на основе их предпочтений и бот-приложение, умеющее распознавать естественный язык пользователя и интегрированное с системой рекомендаций фильмов. Адаптированное под систему рекомендаций тренингов приложение в настоящий момент используется в системе тренингов ИООО «Эксадел» для рекомендации курсов и вебинаров сотрудникам в соответствии с их предпочтениями.

Область применения — в сфере услуг, на информационных порталах, стриминговых сервисах, в социальных сетях и так далее для получения рекомендаций в ходе взаимодействия с бот-приложением на естественном языке.

РЭФЕРАТ

Дыпломная работа, 70 с., 36 мал., 26 крыніц, 7 дадаткаў.

Ключавыя словы: СІСТЭМА РЭКАМЕНДАЦЫЙ, МАШЫННАЕ НАВУЧАННЕ, ML.NET, НАБОР ДАДЗЕНЫХ, МАШЫНЫ ФАКТАРЫЗАЦЫІ, КАЛАБАРАТЫЎНАЯ ФІЛЬТРАЦЫЯ, ПРАБЛЕМА ХАЛОДНАГА СТАРТУ, КОСІНУСНАЕ ПАДАБЕНСТВА, БОТ-ПРЫКЛАДАННЕ, AZURE, CLOUD-ТЭХНАЛОГІІ.

Аб'ект даследавання — набор рэйтынгавых дадзеных аб фільмах з вэб-сайта MovieLens, алгарытмы машыннага навучання для сістэм рэкамендацый, платформа ML.NET, сэрвісы Azure Bot Service и Bot Framework, сэрвісы воблачных вылічэнняў Azure Cloud Services.

Мэта працы — пабудова мадэлі машыннага навучання для сістэмы рэкамендацый, праектаванне і распрацоўка бот-прыкладання з выкарыстаннем Cloud-тэхналогій, якое ўмее распазнаваць натуральную мову карыстальніка, і інтэграцыя бот-прыкладання з сістэмай рэкамендацый.

Метады даследавання — аналіз алгарытмаў для пабудовы сістэм рэкамендацый, магчымасцяў платформы ML.NET і сэрвісаў Azure Bot Service, Bot Framework, Azure Cloud Services, праектаванне мадэлі машыннага навучання для сістэмы рэкамендацый, праектаванне і распрацоўка бот-прыкладання, інтэграцыя бот-прыкладання з сістэмай рэкамендацый.

Вынікамі з'яўляюцца — мадэль машыннага навучання для сістэмы рэкамендацый фільмаў карыстальнікам на аснове іх пераваг і бот-прыкладанне, якое можа распазнаваць натуральную мову карыстальніка і інтэграванае з сістэмай рэкамендацый фільмаў. Адаптаванае пад сістэму рэкамендацый трэнінгаў прыкладанне ў цяперашні момант выкарыстоўваецца ў сістэме трэнінгаў ЗТАА «Эксадэл» для рэкамендацыі курсаў і вэбінараў супрацоўнікам у адпаведнасці з іх перавагамі.

Вобласць ужывання — у сферы паслуг, на інфармацыйных парталах, стрымінгавых сэрвісах, у сацыяльных сетках і гэтак далей для атрымання рэкамендацый у ходзе ўзаемадзеяння з бот-прыкладаннем на натуральнай мове.

ESSAY

Graduate work, 70 p., 36 fig., 26 sources, 7 appendices.

Keywords: RECOMMENDATION SYSTEM, MACHINE LEARNING, ML.NET, DATASET, FACTORIZATION MACHINES, COLLABORATIVE FILTERING, COLD START PROBLEM, COSINE SIMILARITY, BOT APP, AZURE, CLOUD TECHNOLOGIES.

The object of research is a movie rating data set from the MovieLens website, machine learning algorithms for recommendation systems, ML.NET framework, services Azure Bot Service and Bot Framework, cloud computing services Azure Cloud Services.

The purpose of the work is to build a machine learning model for a recommendation system, to design and develop a bot application that can recognize the user's natural language using Cloud technologies, and to integrate the bot application with the recommendation system.

The research methods are analysis of algorithms for building recommendation systems, capabilities of the ML.NET platform and Azure Bot Service, Bot Framework, Azure Cloud Services, designing a machine learning model for a recommendation system, designing and developing a bot application, integrating a bot application with a recommendation system.

The results are a machine learning model for a system of recommending movies to users based on their preferences and a bot application that can recognize the user's natural language and was integrated with the movie recommendation system. The application adapted to the training recommendation system now is used in the training system of Exadel FLLC to recommend courses and webinars to employees in accordance with their preferences.

The scope is in the service sector, on information portals, streaming services, social networks and so on, to get recommendations while interacting with a bot application in natural language.

ВВЕДЕНИЕ

Системы рекомендаций занимают все больше и больше места в нашей жизни. Сегодня рекомендательные системы используются в различных областях: от электронной коммерции — например, покупателям предлагается посетить страницы сайта, которые их могут заинтересовать, до онлайн-рекламы — пользователи видят только «правильный» контент, то есть рекламу, которая может заинтересовать и соответствует их предпочтениям. Рекомендательные системы действительно важны во многих отраслях, потому что они могут является источником огромного дохода, в случае если они эффективны, а также быть способом значительно выделиться среди конкурентов.

Системы рекомендаций — это программы, которые решают задачу прогноза того, какие продукты, например, фильмы, новостные статьи или товары в онлайн-магазине, понравятся пользователям. Исходные данные для таких программ — определенная информация о профиле пользователя, в первую очередь о его предпочтениях в данном продукте и социально-демографических характеристиках, а также характеристики самого продукта [1, с. 187].

Системы рекомендаций реализуются с помощью алгоритмов, основанных на машинном обучении. Ранее разработка моделей машинного обучения была возможна только с помощью таких популярных языков программирования как Python и R. Однако по ряду причин иногда разработчики не могут использовать данные языки, например, если у них есть уже готовая кодовая база на другом языке или отсутствие опыта работы с Python или R. В то же время одним из самых популярных языков сегодня является C#, который используется для разработки многих типов программного обеспечения. Чтобы использовать возможности машинного обучения C#, недавно компания Microsoft реализовала платформу ML.NET. Это среда с открытым исходным кодом, разработанная для обучения, создания и оценки моделей машинного обучения [2].

Боты — это программы, предназначенные для решения задач пользователя и использующие удобный для них интерфейс. При этом все действия выполняются по некоторому заранее заданному алгоритму. Ключевое отличие ботов от других программ заключается в своеобразной имитации действий людей, будь то голос, или текстовые сообщения, так что пользователю иногда не сразу удается понять, что он общается не с человеком, а всего лишь с программой. Боты могут быть использованы для переноса простых, повторяющихся задач, таких как сбор профильной информации, на автоматизированные системы, и это больше не будет требовать непосредственного участия человека. Людям свойственно уставать от

монотонных действий, при этом теряется острота реакции и нарушается концентрация на деталях, в то время как бот не устает и выполняет те же задачи быстрее и точнее. Пользователи могут общаться с ботом, используя текстовые сообщения, интерактивные карточки и речь, при этом взаимодействие с ботом может представлять собой простой диалог с вопросами и быстрыми ответами, или это может быть сложный интеллектуальный разговор, который позволит обеспечить доступ пользователя к различным услугам или службам.

Именно из-за растущей популярности, актуальности и повсеместного использования ботов, для интеграции с ранее разработанной системой рекомендаций было выбрано бот-приложение. Бот-приложения могут быть использованы на сайтах, а также же в пользовательских приложениях, именуемых каналами, — Skype, Telegram, Cortana и других, чтобы всегда быть под рукой. Боты предоставляют помощь пользователю в режиме реального времени, ускоряя путь к достижению цели. Ему достаточно лишь задать вопрос, как если бы пользователь задавал вопрос своему другу, например: «Какой фильм порекомендуешь?», — и мгновенно получить ответ. Бот-приложение с системой рекомендаций может быть интегрировано, например, с веб-сайтом для просмотра фильмов, что позволяет пользователю избежать необходимости листать каталоги и тратить время на сложный выбор: достаточно задать вопрос на естественном языке, и система рекомендаций подберет наиболее подходящие ему кинокартины.

Целью работы является построение модели машинного обучения для рекомендации фильмов пользователям на основе их предпочтений, проектирование и разработка бот-приложения с использованием Cloud-технологий, умеющего распознавать естественный язык пользователя, и интеграция бот-приложения с системой рекомендаций. Однако не ограничивая общности модель может быть обучена на основе данных из других доменных областей и применена в других сферах. Язык разработки — C#.

Задачами работы являются:

1. Обзор алгоритмов для построения рекомендательных систем и областей их применения.
2. Изучение возможностей платформы ML.NET для разработки моделей машинного обучения и, в частности, систем рекомендаций.
3. Обучение и оценка модели машинного обучения для системы рекомендаций.
4. Решение проблемы холодного старта для пользователя для рекомендательной системы.
5. Проектирование пользовательского приложения.
6. Обзор классификации бот-приложений.

7. Изучение и применение средств для разработки, тестирования, развертывания бот-приложений и управления ими в единой среде, предоставляемых Azure Bot Service и Bot Framework.

8. Определение возможностей Azure Cloud Services для разработки бот-приложения, использование некоторых из доступных сервисов при разработке.

9. Разработка бот-приложения с возможностью распознавания естественного языка пользователя.

10. Интеграция бот-приложения с системой рекомендаций.

Объектом исследования являются набор рейтинговых данных о фильмах с веб-сайта MovieLens, алгоритмы машинного обучения для систем рекомендаций, платформа ML.NET, сервисы Azure Bot Service и Bot Framework, сервисы облачных вычислений Azure Cloud Services.

Методы исследования, примененные в данной работе, включают в себя анализ алгоритмов для построения систем рекомендаций, возможностей платформы ML.NET и сервисов Azure Bot Service, Bot Framework, Azure Cloud Services, проектирование модели машинного обучения для системы рекомендаций, проектирование и разработка бот-приложения, интеграция бот-приложения с системой рекомендаций.

Глава 1 носит теоретический характер, в ней проводится обзор существующих систем рекомендаций, рассматриваются существующие алгоритмы машинного обучения для построения систем рекомендаций, а также метрики качества моделей машинного обучения.

Глава 2 посвящена определению возможностей платформы ML.NET для разработки моделей машинного обучения для систем рекомендаций.

В Главе 3 описываются этапы проектирования и оценки модели машинного обучения для системы рекомендаций фильмов.

Глава 4 посвящена решению проблемы холодного старта для систем рекомендаций, а также проектированию пользовательского приложения, интегрированного с системой рекомендаций на основе машинного обучения.

В Главе 5 исследуются технологии разработки бот-приложения, излагаются общие сведения об облачной платформе Azure и предоставляемых ей сервисах.

В Главе 6 описываются этапы проектирования и разработки бот-приложения для рекомендации фильмов, использующего сервис распознавания естественного языка LUIS.

ГЛАВА 1 ИССЛЕДОВАНИЕ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ ДЛЯ СИСТЕМ РЕКОМЕНДАЦИЙ

1.1 Обзор существующих систем рекомендаций и их ограничений

Чтобы пользователи какого-либо программного продукта наиболее вероятно переходили к финальной стадии потребления, например, покупки товара в онлайн-магазине или просмотра фильма на стриминговом сервисе, важно, чтобы данный продукт предоставлял некоторые сервисы для помощи пользователю в выполнении его запросов. Такой сервис должен понимать предпочтения клиентов и в соответствии с этим рекомендовать им наиболее подходящие варианты товаров/продуктов. Пользователи более вероятно вернутся к тому веб-сайту/программе, с помощью которого они достигают своих целей, прикладывая меньше усилий и тратя меньше времени, что достигается наличием сервиса для предоставления ему помощи в выборе / поиске необходимых продуктов.

Системы рекомендаций решают поставленные выше задачи, поэтому широко используются известными брендами. Например, платформа электронной коммерции Amazon предоставляет такие категории товаров как «Часто покупают вместе» и «Покупатели, которые купили это также купили», после того как пользователь перешел на страничку некоторого товара, что существенно увеличивает прибыль данной организации. Однако такой подход больше используется в сфере коммерции и привычен для онлайн-магазинов. Все рекомендации привязаны к товару, выбранному пользователем в данный момент, у него нет возможности увидеть некоторый единый список товаров, подходящий именно для него, ведь переход на страницу продукта не значит, что пользователю он обязательно понравился, и он хочет видеть похожие рекомендации.

Веб-сайт MovieLens предназначен для рекомендации фильмов. Во время регистрации пользователь должен пройти некоторое предварительное анкетирование и после этого может получить рекомендации. При дальнейшем оценивании фильмов рекомендации уточняются, но после выставления рейтингов, чтобы получить новый список фильмов, необходимо обновить страницу. Пользователю также отображаются и фильмы, которые он смотрел, что может быть неудобно.

Все большую популярность приобретают чат-боты — виртуальные ассистенты, которые предоставляют помощь в режиме реального времени. Пользователь может отправлять боту сообщения на естественном языке, как человеку, и тут же получать ответ. Бот распознает намерения и запросы

пользователя и может задавать уточняющие вопросы также на естественном языке, что создает более комфортную среду для клиента приложения.

Например, бренд одежды H&M использует бот в мобильном приложении Kik и предоставляет рекомендации образов на основе предпочтений пользователя в его компонентах: цвете, стиле и так далее. Однако при этом нельзя получить рекомендацию только на основе предыдущих понравившихся образов, не уточняя их компоненты. Аналогично работает бот в мобильном приложении Kik для бренда косметики Sephora, который используется для рекомендации макияжа, на основе предпочтений пользователя в стиле макияжа для глаз, губ в отдельности. Оба бота ограничены только использованием в мобильном приложении, что может создать неудобство, если пользователь захочет в процессе беседы с ботом, посмотреть, например, цену рекомендуемого товара или его характеристики — для этого ему отдельно придется открыть веб-сайт. Беседа в этих ботах построена на схеме «вопрос-ответ», без использования распознавания запросов пользователя на естественном языке.

В результате проведенного обзора существующих систем рекомендаций и бот приложений с учетом всех обнаруженных достоинств и недостатков было решено спроектировать бот-приложение, интегрированное с системой рекомендаций фильмов на основе машинного обучения, с требованиями, сформулированными ниже.

Для создания модели машинного обучения была выбрана платформа ML.NET, которая предоставляет среду для обучения, тестирования и оценки модели. Данные могут быть выбраны из другой предметной области и с помощью нескольких изменений в программе для соответствия новым данным будет обучена совершенно новая модель, например, для рекомендации тренингов, а не фильмов. Таким образом полученная система рекомендаций в первую очередь универсальна. Также она будет обладать гибкостью — ML.NET предоставляет средства для экспорта модели в формат ONNX, что позволяет использовать ее в приложениях не только платформы .NET, но и на других языках, а также в облачных сервисах.

Для системы рекомендаций должна быть решена проблема холодного старта с помощью поиска похожего пользователя в наборе данных, на которых была обучена модель. Таким образом не будет необходимости проводить регулярное трудоемкое переобучение модели, что является безусловным минусом многих систем рекомендаций.

Чтобы пользователь мог получать рекомендации в режиме реального времени и в удобном формате — с помощью запросов на естественном языке, было решено использовать бот-приложение, разработанное, с использованием сервисов Azure Bot Service и Bot Framework, а также сервиса распознавания

естественного языка LUIS. Такое бот-приложение может быть интегрировано как на веб-сайт — в виде виджета чата, так и в любое пользовательское приложение, например, Telegram, что позволяет не ограничивать пользователя и дать возможность ему использовать услуги бота в удобном формате.

Рекомендации пользователю будут предоставляться в списке из пяти наиболее подходящих ему фильмов. Для фильмов должна быть реализована возможность их оценки (выставления рейтинга) — таким образом каждый следующий список рекомендаций будет более точным. Уже оцененные фильмы будут исключены из последующих рекомендаций, чтобы избежать повторения и рекомендации пользователю одних и тех же фильмов. Для новых пользователей должны предоставляться начальные рекомендации — список из двадцати фильмов, наиболее популярных по среднему рейтингу среди всех пользователей. После оценки некоторых из этих фильмов, пользователю могут быть составлены персонализированные рекомендации.

1.2 Типы систем рекомендаций

Основная идея систем рекомендаций заключается в идее, что если пользователи выбирают похожие товары при покупке в интернет-магазине, читают одинаковые новостные статьи или смотрят одни и те же фильмы, то велика вероятность того, что в дальнейшем они будут вести себя так же похоже. То есть системы рекомендаций оценивают предпочтения пользователей в отношении товаров, сервисов, продукции, и делают прогноз, что другим пользователям они тоже могут понравиться.

Системы рекомендаций анализируют товары двумя способами: с точки зрения того, насколько они нравятся пользователям: по принципу, что людям с похожими предпочтениями нравятся одни и те же продукты, или с точки зрения того, насколько они популярны среди пользователей, независимо от сходства предпочтений отдельных покупателей.

На основе этих двух подходов формируются три основных типа систем рекомендаций: фильтрация на основе содержания, коллаборативная фильтрация и гибридная фильтрация [3, с. 2].

Фильтрация на основе содержания представляет собой анализ некоторой дополнительной информации о пользователях и/или продуктах, ранее оцененных этими пользователями. Главная идея этого метода состоит в том, чтобы построить модель, основанную на доступных характеристиках объекта, которые объясняют взаимодействие пользователей с ним. При этом для каждого пользователя создается свой профиль. Например, фильмы-боевики больше нравятся мужчинам, а романтические — женщинам.

Коллаборативная фильтрация анализирует действия пользователей и пытается найти сходство между их предпочтениями. То есть если у пользователей совпали вкусы в прошлом, то предполагается, что они совпадут и в будущем. Например, если двум пользователям нравятся одинаковые фильмы, а первый посмотрел фильм, который еще не видел второй, то есть смысл порекомендовать этот фильм второму.

Системы рекомендаций с гибридной фильтрацией сочетают в себе оба предыдущих метода для создания еще более точных рекомендаций. Гибридная фильтрация может быть реализована путем создания, а затем объединения отдельных прогнозов, полученных с помощью фильтрации на основе содержания и коллаборативной фильтрации.

Таким образом систему рекомендаций можно определить как прогнозирование рейтинга, проставленного пользователем для некоторого объекта, или же как прогнозирование k лучших объектов для конкретного пользователя. Далее в данной работе будет рассмотрено прогнозирование пяти наиболее подходящих для пользователя фильмов.

Рассмотрим коллаборативную фильтрацию подробнее.

1.3 Коллаборативная фильтрация

Главным преимуществом коллаборативной фильтрации является тот факт, что чем больше пользователей взаимодействуют с продуктами, тем более точными становятся новые рекомендации, поскольку для фиксированного набора пользователей и продуктов запись новых взаимодействий приносит новую информацию и делают всю систему эффективней.

Однако так как коллаборативная фильтрация учитывает только прошлые взаимодействия пользователей с продуктами для прогнозирования новых рекомендаций, ее минусом является потенциальная проблема «холодного старта»: новые пользователи не могут получить какие-либо рекомендации или новый продукт не может быть рекомендован каким-либо пользователям. Чтобы решить эту проблему можно рекомендовать [4]:

- случайно выбранные продукты новым пользователям или новые продукты случайным выбранным пользователям (случайная стратегия);
- популярные продукты новым пользователям или новые продукты пользователям, имеющим наибольшее количество взаимодействий с продуктами (стратегия максимального ожидания);
- набор различных продуктов новым пользователям или новый продукт набору различных пользователей (исследовательская стратегия).

Традиционно выделяют два подхода коллаборативной фильтрации: основанный на памяти и основанный на модели [4].

В методах, основанных на памяти (соседстве) алгоритмы непосредственно работают с взаимодействиями пользователя и продукта. В методах же, основанных на модели, предполагается некоторая модель взаимодействия, которая обучается прогнозировать значения взаимодействия пользователя и продукта. Второй подход наиболее популярен на сегодняшнее время, поэтому далее он будет рассмотрен подробнее в виде исследования алгоритмов матричного разложения (англ. matrix factorization), машин факторизации (англ. factorization machines) и машин факторизации с учетом специфики поля (англ. field-aware factorization machines).

1.4 Алгоритмы коллаборативной фильтрации

Когда пользователь дает оценку некоторым фильмам, например, ставит баллы от одного до пяти, выставленные рейтинги могут быть представлены в виде матрицы. Каждая строка представляет пользователя, а каждый столбец — определенный фильм. Матрица при этом будет разреженной, так как не все пользователи посмотрели и оценили все фильмы. Эту матрицу рейтингов можно рассматривать как произведение матриц меньшей размерности, представляющих пользователей и фильмы. Данный подход называется матричным разложением (факторизацией) [5]. При этом матрица может быть представлена только как матрица пользователей и фильмов, куда не могут быть включены различные характеристики, например, жанр фильма, главные актеры и так далее.

Машины факторизации представлены как улучшенная версия матричного разложения. То есть это более общий подход, который позволяет имитировать большинство моделей взаимодействия с помощью использования признаков объектов, а матричное разложение — это просто частный случай машины факторизации. Взаимодействие пользователя с объектом представляется в виде вектора, полученного в ходе прямого унитарного кодирования (англ. one hot encoding), при этом каждая преобразованная строка в матрице будет иметь только одну единицу в векторе (взаимодействие одного пользователя и одного объекта). Затем добавляются дополнительные признаки (например, жанр фильма, последний просмотренный фильм и так далее), которые также преобразуются с помощью прямого унитарного кодирования, либо нормализации векторов. Таким образом, машины факторизации могут оценивать взаимодействия в изначально разреженных параметрах, так как они нарушают независимость параметров взаимодействия, факторизуя их. Это означает, что данные для одного взаимодействия пользователя и объекта помогают также оценить параметры для связанных с ними взаимодействий.

Следующий способ реализации коллаборативной фильтрации — машины факторизации с учетом специфики поля, который является расширением машин факторизации. Здесь поля — это независимые переменные, например, жанр фильма, а категориальные значения, которые принимает каждое поле, называются признаками, например, жанр комедия или боевик [5].

В машинах факторизации у каждого признака есть только один вектор, который позволяет изучить его взаимодействие с остальными признаками. Например, изучается взаимодействие конкретного пользователя с жанром фильма комедия, и конкретным главным актером. Машина факторизации с учетом специфики поля разбивает этот один вектор на несколько — по одному для представления каждого другого поля, так как интуитивно понятно, что вектора взаимодействия пользователя с жанром и актером будут разные. Так образуются вектора взаимодействия поля пользователь с жанром комедия, поля пользователь с конкретным актером, поля жанр с конкретным пользователем и так далее. То есть машины факторизации с учетом специфики поля представляют взаимодействия на более детальном уровне, поэтому можно использовать меньше признаков чем в обычных машинах факторизации, при этом сохраняя ту же эффективность. Для больших, разреженных наборов данных со многими категориальными характеристиками этот алгоритм работает лучше обычных машин факторизации.

1.5 Метрики качества бинарной классификации

Задача прогнозирования того, какой фильм будет рекомендован пользователю, а какой нет, и какова вероятность того, что фильм понравится, может быть представлена как задача бинарной классификации [6]. Для каждого пользователя фильмы распределяются по вероятности того, что они будут высоко им оценены. Затем с помощью некоторого порогового значения (например, рейтинг фильма, начиная с которого фильм считается хорошим для рекомендации) генерируются самые лучшие рекомендации для пользователя.

Машина факторизации с учетом специфики поля позволяет решить эту задачу с предварительной подготовкой данных: преобразованием признаков с помощью прямого унитарного кодирования или нормализации векторов, а рейтингов в бинарный тип данных: может ли фильм быть рекомендован или нет.

После обучения модели необходимо оценить ее эффективность. Для каждого класса задач машинного обучения есть свои метрики. Далее рассмотрим метрики качества для задач бинарной классификации [7]:

1. Точность (англ. accuracy). Представляет собой долю правильных прогнозов в тестовом наборе.

2. Матрица несоответствий / ошибок (англ. confusion matrix). Данные делятся на два класса: положительный и отрицательный. Матрица содержит две строки и два столбца, и каждый ее элемент равен числу объектов класса, соответствующего строке, которым алгоритм присвоил метку класса, соответствующего столбцу. Прогнозы называются положительными (англ. positive), если алгоритм относит их к одноименному классу, при этом те из них, которые на самом деле принадлежат положительному классу — истинно положительными (англ. true positive), а остальные — ложно положительными (англ. false positive). Таким же способом разделяются прогнозы, принадлежащие отрицательному классу (англ. negative): истинно отрицательные (англ. true negative) и ложно отрицательные (англ. false negative).

3. Отрицательная точность (англ. negative precision). Определяет долю истинно отрицательных прогнозов среди всех распознанных отрицательных прогнозов (истинно и ложно отрицательных).

4. Истинный отрицательный рейтинг или специфика (англ. negative recall). Определяет долю истинно отрицательных прогнозов среди всех реально отрицательных прогнозов (истинно отрицательных и ложно положительных).

5. Положительная точность или точность (англ. positive precision). Определяет долю истинно положительных прогнозов среди всех распознанных положительных прогнозов (истинно и ложно положительных).

6. Истинный положительный рейтинг или полнота, чувствительность (англ. positive recall). Определяет долю истинно положительных прогнозов среди всех реально положительных прогнозов (истинно положительных и ложно отрицательных).

7. Площадь под кривой полноты и точности (англ. area under precision recall curve). Используют в качестве метрики качества алгоритма.

8. Площадь под кривой ошибок (англ. area under ROC curve). ROC (англ. receiver operating characteristic) представляет собой график, который отображает зависимость истинно положительных прогнозов от ложно положительных прогнозов. В свою очередь площадь под данным графиком представляет собой вероятность того, что случайно выбранный прогноз из положительного класса будет более вероятно распознан как элемент положительного класса, чем прогноз, случайно выбранный из отрицательного класса.

9. F1-мера (англ. F1-score) — мера качества с учетом и точности, и полноты.

ВЫВОДЫ

1. Проведен обзор существующих систем рекомендаций, составлены требования к разрабатываемому приложению.

2. Исследованы типы систем рекомендаций: фильтрация на основе содержания, коллаборативная фильтрация и гибридная фильтрация.

3. Рассмотрены алгоритмы коллаборативной фильтрации, основанные на модели: матричное разложение, машины факторизации и машины факторизации с учетом специфики поля.

4. Для создания системы рекомендаций фильмов выбран алгоритм машин факторизации с учетом специфики поля.

5. Рассмотрены метрики качества обученной модели для выбранного алгоритма.

ГЛАВА 2 ИЗУЧЕНИЕ ВОЗМОЖНОСТЕЙ ПЛАТФОРМЫ ML.NET

2.1 Задачи машинного обучения в ML.NET

ML.NET — это среда с открытым исходным кодом, разработанная для обучения, создания и оценки моделей машинного обучения [2]. Она позволяет перенести механизм конвейера машинного обучения в .NET.

Центральным элементом ML.NET является модель машинного обучения. Модель определяет шаги, необходимые для преобразования входного набора данных в прогноз. ML.NET позволяет обучить новую пользовательскую модель, подобрав нужный алгоритм, а также импортировать предварительно обученные модели. Созданную и обученную модель также можно экспортировать для интеграции с другими проектами и приложениями.

ML.NET поддерживается такими операционными системами как Windows, Linux и macOS при использовании .NET Core, а также Windows с использованием .NET Framework.

Платформа ML.NET предоставляет несколько основных алгоритмов машинного обучения. Она может использоваться для решения следующих типов задач [8]:

- Классификация/категоризация — включает в себя двоичную (бинарную) классификацию и многоклассовую классификацию. Используется для прогнозирования к какому из двух или более классов (категорий) принадлежит объект. Например, для разделения комментариев пользователей на сайте на положительные и отрицательные, или определения породы собаки по фото.
- Регрессия / прогнозирование непрерывных значений — используется для прогнозирования значения метки по набору связанных признаков. Например, для прогноза цены на некоторый товар, исходя из его характеристик.
- Кластеризация — используется для группировки объектов, содержащие сходные характеристики, в кластеры. Например, для определения потребительских сегментов и их демографических характеристик, чтобы исходя из этого построить целевую рекламную кампанию.
- Обнаружение аномалий — используется для решения таких задач как выявление операций, потенциально являющихся мошенническими, или создание обучающих шаблонов, указывающих на то, что произошло сетевое вторжение.
- Системы рекомендаций — используются для создания списков продуктов или сервисов, которые могут понравиться пользователям. Например,

для предложения продуктов в онлайн-магазине, которые покупатели могут захотеть купить.

- Временные ряды / последовательности — используются для создания прогнозов на будущее. Например, для прогнозов погоды.

2.2 Построение модели и ее использование в ML.NET: обзор архитектуры

На рисунке 2.1 изображена схема процесса построения модели в ML.NET и возможности ее дальнейшего использования.

Процесс построения модели в ML.NET начинается с подбора набора данных, которые будут использоваться для обучения модели.

Данные поступают на вход приложения ML.NET — сервиса для обучения модели. Там набор данных в первую очередь преобразовывается в зависимости от выбранного алгоритма обучения. Далее происходит обучение модели и ее оценка. Обучение данных зачастую ресурсоемкий процесс, поэтому часто для него используются мощные вычислительные кластеры, предоставляемые Azure Cloud Services или Amazon Web Services. В результате на выходе этого процесса получается обученная модель ML.NET.

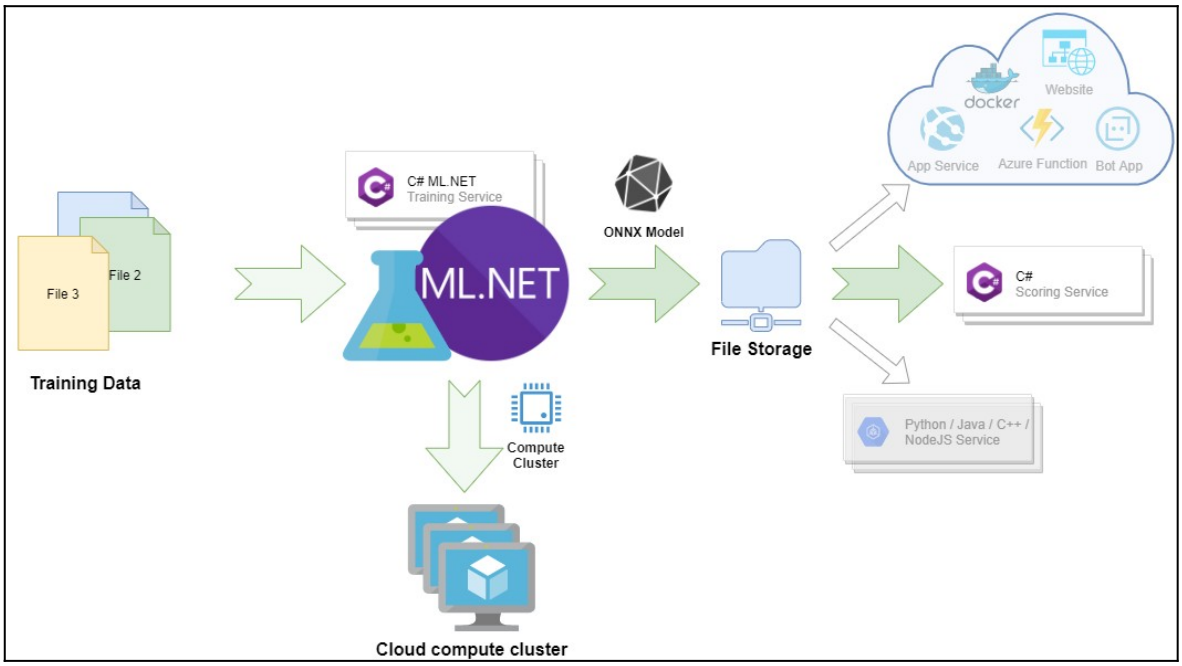


Рисунок 2.1 — Схема построения модели в ML.NET и ее использование

Как ранее упоминалось, ML.NET позволяет импортировать предварительно обученные модели (в формате TensorFlow и ONNX), а также экспортировать обученную модель в данные форматы.

TensorFlow — библиотека с открытым исходным кодом для машинного обучения, которая была разработана компанией Google и может быть использована для решения задач построения и обучения нейронной сети для классификации образов [9].

ONNX — это открытый и совместимый стандартный формат для представления моделей машинного обучения (а также глубокого обучения), который позволяет сохранять обученные модели, созданные на любой платформе, в формате ONNX и запускать их на различных платформах, включая .NET [10].

ONNX является самым распространенным форматом для моделей машинного обучения. Главное его преимущество — поддержка взаимодействия между различными платформами. Это означает, что модель можно обучить, используя одну из многих популярных платформ для машинного обучения, например, PyTorch, преобразовать ее в формат ONNX и использовать полученную модель ONNX уже на другой платформе, например, ML.NET.

При этом ML.NET предоставляет возможность не только импортировать обученные модели, но и экспортировать их в формат ONNX и затем использовать в различных сервисах, в том числе реализованных на других платформах. На рисунке 2.1 отображены возможные пути использования модели: в .NET приложении на языке C#, в различных облачных сервисах, а также в приложениях на других языках программирования (Java, Python и так далее).

Таким образом модель, обученную в ML.NET, можно использовать для создания прогнозов на различных устройствах, операционных системах, платформах — там, где поддерживается ONNX Runtime — высокопроизводительный механизм вывода для моделей машинного обучения в ONNX [11].

2.3 Процесс машинного обучения в ML.NET

Приложение ML.NET начинается с объекта типа MLContext, который содержит различные каталоги. Каталог — это фабрика для загрузки и сохранения данных, функций преобразования и обучения данных, а также компонентов операций модели [8]. Каждый объект каталога имеет свои методы для создания различных типов этих компонентов. Например, для загрузки, кэширования и сохранения данных предназначен каталог DataOperationsCatalog, для преобразования данных перед обучением — TransformsCatalog, для сохранения, загрузки модели и создания прогнозов — ModelOperationsCatalog, а алгоритмы обучения представлены в нескольких каталогах в зависимости от решаемой задачи (бинарная классификация в

BinaryClassificationCatalog, рекомендации в RecommendationCatalog и так далее).

На рисунке 2.2 изображена диаграмма деятельности, отображающая процесс построения модели машинного обучения в ML.NET. Диаграмма деятельности — это поведенческая диаграмма, которая отображает поток управления от начальной до конечной точки программы, показывая различные пути принятия решений, которые существуют во время выполнения действия [12].

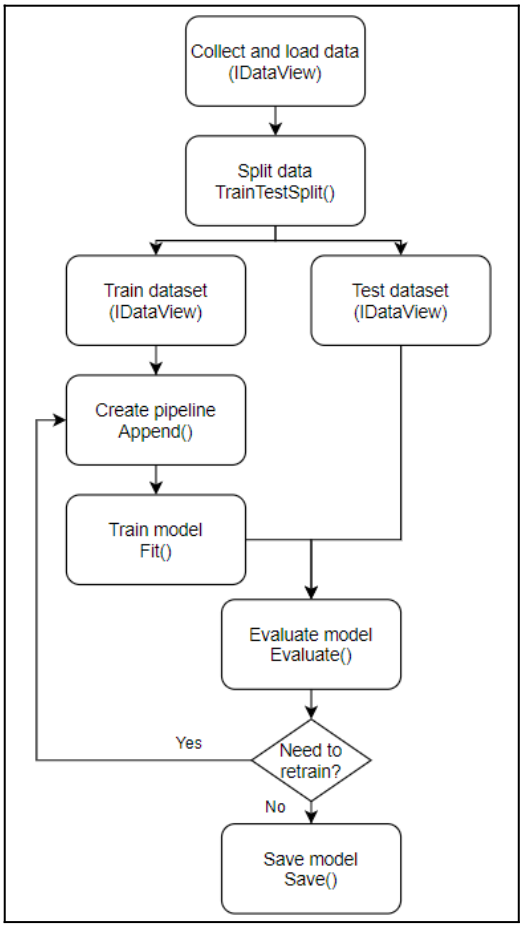


Рисунок 2.2 — Диаграмма деятельности процесса обучения модели

Процесс разработки модели в ML.NET представим в виде последовательности шагов [8]:

1. Сбор и загрузка данных для обучения в объект типа, реализующего интерфейс IDataView.
2. Деление набора данных на обучающий (тренировочный) и тестовый наборы в заданном соотношении.
3. Создание конвейера и включение в него операций (с помощью метода Append) для подготовки тренировочных данных к обучению (преобразование в нужный формат, выбор необходимых признаков, объединение столбцов и так далее), а также выбранного алгоритма машинного обучения.

4. Обучение модели путем вызова метода Fit ранее созданного конвейера.

5. Оценка модели (метод Evaluate) и повторение предыдущих пунктов при необходимости.

6. Сохранение модели в двоичном формате для использования в других приложениях (метод Save).

После сохранения модель может быть использована в других приложениях для создания прогнозов. На рисунке 2.3 изображена диаграмма деятельности процесса использования обученной модели.

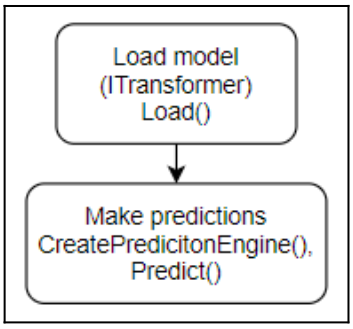


Рисунок 2.3 — Диаграмма деятельности процесса использования модели

Для использования обученной модели выполняются всего два этапа:

1. Загрузка модели в объект типа, реализующего интерфейс ITransformer, с помощью метода Load.

2. Прогнозирование. Включает в себя создание объекта типа PredictionEngine с помощью метода CreatePredictionEngine. PredictionEngine — API, который позволяет выполнять прогнозирование на одном экземпляре тестовых данных, куда в качестве двух универсальных параметров передается тип тестовых данных, для которых будет осуществляться прогноз, и тип самого прогноза. Далее вызывается метод Predict объекта PredictionEngine, куда в качестве аргумента передаются входные данные, для которых будет делаться прогнозирование.

ВЫВОДЫ

1. Рассмотрены различные типы задач машинного обучения, которые позволяет решать платформа ML.NET.

2. Изучен процесс машинного обучения в ML.NET: загрузка и подготовка данных, деление их на тренировочный и тестовый наборы, создание конвейера, обучение и оценка полученной модели.

3. Определено, как использовать модель ML.NET в других приложениях или сервисах: ее нужно сохранить, а затем загрузить в этом приложении — тогда можно делать необходимые прогнозы.

4. Выяснено, как использовать модель на различных устройствах, операционных системах, платформах: для этого применяется экспорт модели в формат ONNX.

ГЛАВА 3 ПОСТРОЕНИЕ И ТЕСТИРОВАНИЕ МОДЕЛИ СИСТЕМЫ РЕКОМЕНДАЦИЙ

3.1 Загрузка данных

3.1.1 Обзор набора данных MovieLens

В качестве набора данных для обучения модели машинного обучения для системы рекомендаций фильмов использовался набор рейтинговых данных о фильмах с веб-сайта MovieLens, которые были собраны и предоставлены исследовательской лабораторией GroupLens. GroupLens — исследовательская группа факультета компьютерных наук и инженерии университета Миннесоты [13]. Со дня своего основания в 1992 году GroupLens реализовала множество различных проектов, которые исследовали такие области как:

- системы рекомендаций;
- онлайн-сообщества;
- мобильные и повсеместные технологии;
- электронные библиотеки;
- локальные географические информационные системы.

Набор данных о фильмах содержит двадцать семь миллионов пятизвездочных рейтингов, которые представляют собой оценку пятидесяти восьми тысячи фильмов двумястами восьмьюдесятью тысячами пользователями, а также сто десять тысяч тегов, проставленных данным фильмам. Данные собирались в период с января 1995 года по сентябрь 2018 года.

Все файлы с данными (genome-scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv, tags.csv) формата CSV, каждый из них содержит одну строку заголовка с названием столбцов. Столбцы содержат запятые в качестве разделителей, а поля, содержащие запятые, заключаются в двойные кавычки — символ «"». В данной работе для обучения модели использовались данные из файлов movies.csv и ratings.csv, поэтому их структура будет рассмотрена подробнее.

Все пользователи оценили по крайней мере один фильм. Каждый пользователь представлен только анонимным идентификатором, то есть никакая другая информация, включая демографическую, не предоставляется.

В набор данных включаются фильмы, у которых есть хотя бы один тег или оценка пользователя. Структура файла movies.csv представлена на рисунке 3.1.

Информация о каждом фильме представляет собой идентификатор (он нужен, так как названия фильмов, даже выпущенных в один год, могут быть одинаковые), название фильма и жанры. Каждый фильм имеет от одного и

более жанра, несколько жанров разделены символом «|». Жанры могут принимать следующие значения [13]: Action (боевик), Adventure (приключения), Animation (анимация), Children (детский), Comedy (комедия), Crime (криминал), Documentary (документальный), Drama (драма), Fantasy (фэнтези), Film-Noir (криминальная драма), Horror (ужасы), Musical (мюзикл), Mystery (мистика), Romance (романтический), Sci-Fi (Science Fiction — научная

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children...
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (199...	Comedy
6	Heat (1995)	Action|Crime|Thriller
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action|Adventure|Thriller

фантастика), Thriller (триллер), War (военный), Western (вестерн), IMAX (максимальное изображение), «(no genres listed)» (без жанра)

Рисунок 3.1 — Структура файла movies.csv

Вся информация о рейтингах содержится в файле ratings.csv. Файл также содержит заголовок, а далее данные в файле представляют собой оценку определенного фильма некоторым пользователем и имеет структуру, представленную на рисунке 3.2.

userId	movieId	rating	timestamp
1	2840	3.0	1256677500
1	2986	2.5	1256677496
1	3020	4.0	1256677260
1	3424	4.5	1256677444
1	3698	3.5	1256677243
1	3826	2.0	1256677210
1	3893	3.5	1256677486
2	170	3.5	1192913581
2	849	3.5	1192913537
2	1186	3.5	1192913611
2	1235	3.0	1192913585
2	1244	3.0	1192913551

Рисунок 3.2 — Структура файла ratings.csv

Таким образом информация о каждой оценке включает в себя идентификатор пользователя, оценившего фильм, представленный также идентификатором, собственно рейтинг — балл по шкале от одной до пяти звезд с шагом ползвезды, и метка времени — количество секунд с полуночи 1 января 1970 года по всемирному координированному времени. Строки внутри этого файла упорядочиваются сначала по идентификатору пользователя, а затем для каждого пользователя — по идентификатору фильма.

Идентификатор фильма можно использовать для ссылки на другие источники с данными о фильмах. Чтобы получить информацию о фильме можно воспользоваться сервисом https://movielens.org. Если идентификатор фильма равен, например, 11, нужно выполнить запрос https://movielens.org/movies/11. Результат запроса представлен на рисунке 3.3.

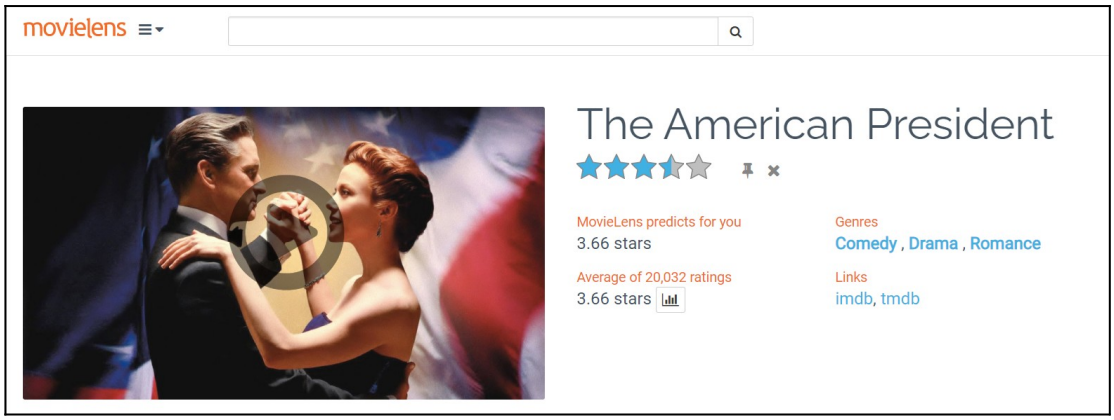


Рисунок 3.3 — Информация о фильме на сайте https://movielens.org

Еще один распространённый сервис с информацией о фильмах представлен по адресу https://www.themoviedb.org. Пример данных о фильме представлен на рисунке 3.4.

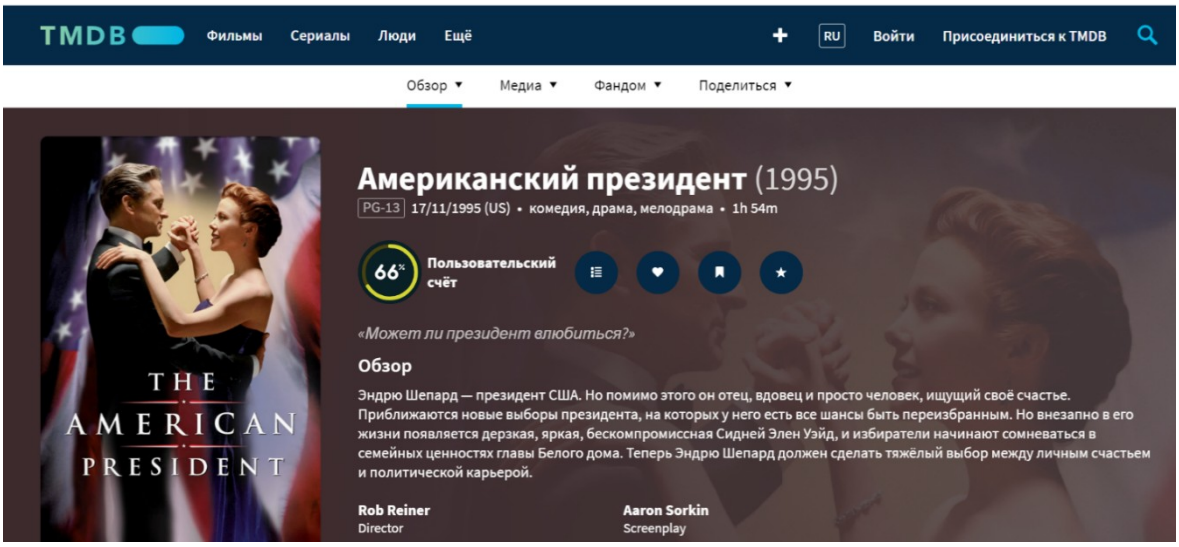


Рисунок 3.4 — Информация о фильме на сайте https://www.themoviedb.org

Оба выше упомянутые сервиса используются далее в данной работе для получения постера фильма по его идентификатору.

3.1.2 Получение и обработка данных

Как ранее упоминалось, в данной работе используются два файла: ratings.csv и movies.csv. Колонка с меткой времени в ratings.csv в данном случае не нужна, так как время выставления оценки пользователем не влияет на то, как он оценивает данный фильм, и поэтому не будет способствовать более точной рекомендации.

Из двух этих файлов данные загружаются и объединяются в один набор данных (рисунок 3.5). Функции загрузки данных из файлов (LoadRatings и LoadMovies соответственно) реализованы в классе DataProcessor и представлены в приложении А. Данные о рейтингах и фильмах загружаются с помощью встроенной функции LoadFromTextFile, в параметрах указывается значение true для аргумента hasHeader (первая строчка файла содержит названия столбцов), а в значении separatorChar указывается символ «,», так как по умолчанию разделитель принимает значение «\t».

userId	movieId	rating	title	genres
1	307	3.5	Three Colors: Blue (Trois coule...	Drama
1	481	3.5	Kalifornia (1993)	Drama|Thriller
1	1091	1.5	Weekend at Bernie's (1989)	Comedy
1	1257	4.5	Better Off Dead... (1985)	Comedy|Romance
1	1449	4.5	Waiting for Guffman (1996)	Comedy
1	1590	2.5	Event Horizon (1997)	Horror|Sci-Fi|Thriller
1	1591	1.5	Spawn (1997)	Action|Adventure|Sci-Fi|Thriller
1	2134	4.5	Weird Science (1985)	Comedy|Fantasy|Sci-Fi
1	2478	4.0	¡Three Amigos! (1986)	Comedy|Western
1	2840	3.0	Stigmata (1999)	Drama|Thriller

Рисунок 3.5 — Информация о фильмах и рейтингах

Для данных о рейтингах дальнейшая обработка не требуется, однако для данных о фильмах требуется решение задачи денормализации жанра. Каждая строка, содержащая список жанров, разбивается на несколько строк, так что в каждой строке становится по одному жанру. Пример денормолизации жанра для фильма с идентификатором 1 представлен на рисунке 3.6.

userId	movieId	rating	title	genres
4	1	4.0	Toy Story (1995)	Adventure
4	1	4.0	Toy Story (1995)	Animation
4	1	4.0	Toy Story (1995)	Children
4	1	4.0	Toy Story (1995)	Comedy
4	1	4.0	Toy Story (1995)	Fantasy

Рисунок 3.6 — Денормализация жанра

3.2 Создание и обучение модели

3.2.1 Преобразование данных для построения модели

После загрузки данные делятся на тренировочный (обучающий) и тестовый наборы в соответствии с заданной долей тестовых данных. В данной работе 80% данных — тренировочные, оставшиеся 20% — тестовые. Деление на наборы осуществляется с помощью встроенной функции TrainTestSplit, ее использование отражено в функции LoadData класса DataProcessor (приложение А).

По умолчанию, когда данные обрабатываются, они передаются потоком отложенным образом, то есть алгоритмы обучения могут загружать данные с диска и проходить по ним несколько раз во время обучения. Поэтому для наборов данных рекомендуется делать кэширование, при этом данные помещаются в память, что уменьшает количество обращений к диску для загрузки данных. Кэширование обучающих данных производится сразу после деления данных на тренировочный и тестовый наборы и осуществляется с помощью встроенной функции Cache.

В машинном обучении столбцы, используемые для прогнозирования, называются признаками, а столбец с возвращенным прогнозом называется меткой. В данном случае идентификатор пользователя, идентификатор, название и жанр фильма — признаки, а рейтинг — метка.

Алгоритмы машинного обучения ML.NET требуют того, чтобы входные данные или объекты находились в одном числовом векторе. Аналогично, значение для метки также должно быть закодировано. Поэтому следующим этапом подготовки данных является получение данных в формате, ожидаемом алгоритмом ML.NET.

Одним из наиболее распространенных типов данных являются категориальные данные. Такие данные представляют собой конечное число категорий. Например, жанры фильмов — категориальные данные. Категориальные данные (как признаки, так и метки), должны быть сопоставлены с некоторым числовым значением для того, чтобы их можно было использовать для создания модели машинного обучения. Существует несколько способов работы с категориальными данными в ML.NET, в данной работе в силу выбранного алгоритма для построения системы рекомендаций (машина факторизации с учетом специфики поля) использовалось прямое унитарное кодирование.

Прямое унитарное кодирование берет конечный набор значений и сопоставляет им целые числа, которые в двоичном представлении имеют только один разряд со значением один, остальные же — нули [14]. Прямое унитарное кодирование удобно использовать, если категориальные данные не должны быть упорядочены (например, порядок жанров не важен).

Таким образом все столбцы с признаками были преобразованы с помощью прямого унитарного кодирования, а затем объединены в один столбец Features. Столбец с рейтингом был преобразован из значения типа float в значение типа bool: если рейтинг больше 3,5, то фильм может быть рекомендован пользователю, если ниже — нет.

Описанные выше преобразования были реализованы с помощью встроенной функции ML.NET OneHotEncoding для прямого унитарного кодирования и CustomMapping — произвольного отображения данных, определяемого пользовательским классом IsRecommendedCustomAction, для преобразования метки к типу bool. При этом так как пользовательское преобразование дальше будет частью сохраненной модели, то для него надо предоставить имя контракта, то есть зарегистрировать.

3.2.2 Обучение модели

После подготовки данных наступает этап обучения модели, который включает в себя выбор алгоритма машинного обучения из каталогов ML.NET и вызова встроенной функции Fit с передачей ей в качестве параметра полученного ранее набора тренировочных (обучающих) данных, представленных типом, реализующим интерфейс IDataView.

Весь процесс преобразования набора данных в обученную модель выполняется в обучающем конвейере. Внутри каждого каталога, например, Transforms, есть специальные расширяющие методы для добавления этапов обработки данных в обучающий конвейер. Ранее в конвейер были добавлены методы для произвольного отображения данных и прямого унитарного кодирования, а также объединения нескольких признаков в один столбец Features. Последним этапом в конвейер добавляется выбранный алгоритм обучения. В данной работе это машина факторизации с учетом специфики поля, которая очень эффективна для построения рекомендаций по множественным характеристикам. Алгоритм для него реализован в классе FieldAwareFactorizationMachine, который находится в каталоге алгоритмов бинарной классификации (BinaryClassification).

После создания объектов в конвейере данные можно использовать для обучения модели. Функция Fit возвращает обученную модель, которая

представляет собой объект типа, реализующего интерфейс ITransformer. То есть модель преобразует входные данные в прогнозы.

3.3 Оценка и сохранение модели

Чтобы построить наиболее эффективную модель, необходимо оценить ее производительность по тестовым данным. Для этого в ML.NET используется функция Evaluate, которая позволяет измерить различные метрики для обученной модели. Полученные метрики зависят от того, какая задача машинного обучения решалась.

Для создания прогнозов на тестовом наборе данных используется обученная модель машинного обучения, для которой вызывается метод Transform. Таким образом перед оценкой тестовые данные сначала преобразовываются, и в результате формируются прогнозы.

Как только прогнозы получены, метод Evaluate оценивает модель, которая сравнивает спрогнозированные значения с фактическими метками (Labels) в тестовом наборе данных и возвращает метрики о том, насколько эффективно работает модель.

Так как машина факторизации с учетом специфики поля (FieldAwareFactorizationMachine) находится в каталоге задач бинарной классификации, то доступные после оценки метрики возвращаются в качестве объекта класса CalibratedBinaryClassificationMetrics и включают в себя свойства, представленные на рисунке 3.7. После обучения модели все метрики для удобства выводятся в файл evaluation_metrics.txt.

Подробное значение каждой метрики было описано в главе 1.

```
Evaluation Metrics
Accuracy: 0,78
Area Under Roc Curve: 0,86
Area Under Precision Recall Curve: 0,86
F1 Score: 0,78
Negative Precision: 0,78
Negative Recall: 0,77
Positive Precision: 0,77
Positive Recall: 0,78
Confusion matrix: TEST POSITIVE RATIO:  0,5006 (7530500,0/(7530500,0+7511742,0))
Confusion table
          ||======================
PREDICTED || positive | negative | Recall
TRUTH     ||======================
 positive || 5 910 246 | 1 620 254 | 0,7848
 negative || 1 741 051 | 5 770 691 | 0,7682
          ||======================
Precision ||   0,7725 |   0,7808 |
```

Рисунок 3.7 — Метрики обученной модели

На протяжении процесса создания и обучения модели она хранится в памяти и доступна во время работы приложения. Однако когда приложение прекращает свое выполнение, модель перестает быть доступна. Обычно модель

используются после обучения в других пользовательских приложениях либо для выполнения прогнозов, либо для повторного обучения, поэтому ее необходимо сохранить. Сохранение модели осуществляется с помощью функции Save. Этот метод сохраняет обученную модель в файл формата zip, который затем может быть использован в других приложениях .NET для выполнения прогнозов. При этом регистрация пользовательского отображения (перевод рейтинга из типа float в тип bool) в сборке — часть процесса сохранения.

ВЫВОДЫ

1. Изучен набор данных о фильмах MovieLens.
2. Загружены и обработаны данные о рейтингах и фильмах из набора данных MovieLens.
3. Данные подготовлены для построения модели.
4. Создана и обучена модель для системы рекомендаций фильмов.
5. Оценена эффективность обученной модели.
6. Модель сохранена для использования в других приложениях.
7.

ГЛАВА 4 РЕШЕНИЕ ПРОБЛЕМЫ ХОЛОДНОГО СТАРТА В СИСТЕМАХ РЕКОМЕНДАЦИЙ

4.1 Определение проблемы холодного старта

Проблема холодного старта или проблема нового пользователя / элемента — невозможность предоставить новому пользователю точные рекомендации или рекомендовать пользователям новинки в товарах / продуктах [15].

В системах рекомендаций «холодный старт» означает, что обстоятельства еще не являются оптимальными для того, чтобы система могла обеспечить наилучшие возможные результаты. Есть две различные категории проблемы холодного старта: холодный старт для продукта и холодный старт для пользователя [16].

4.1.1 Холодный старт для продукта

Согласно основному предположению концепции коллаборативной фильтрации, если продукт уже был популярным среди определенной группы людей, то другие пользователи, имеющие схожие предпочтения, скорее всего, тоже хорошо отреагируют на данный продукт.

Проблема холодного старта для продукта возникает, когда товары, добавленные в каталог, либо имеют мало, либо не имеют вовсе каких-либо взаимодействий с пользователями. Это представляет собой проблему для алгоритма коллаборативной фильтрации из-за того, что он основан на взаимодействиях пользователей и продуктов. Соответственно, если нет таких взаимодействий, то алгоритм не может никому рекомендовать этот товар.

В данном случае действия пользователя очень важны, так как они определяют будущее как продуктов, так и персонализированных рекомендаций, которые составлены специально для данного конкретного пользователя и основаны на его собственной истории взаимодействия с продуктами. При этом, если для определенного продукта не будет достаточного количества взаимодействий с различными пользователями, чтобы заложить основу для точных рекомендаций, система не будет знать, когда предлагать этот продукт в качестве рекомендации. Таким образом, чем больше взаимодействий будет собрано с новым продуктом, тем легче системе составлять точные персонализированные рекомендации.

В качестве систем, проходящих через проблему холодного старта для продукта, выступают новостные и тематические сайты, сайты для аукционов, онлайн-магазины и многие другие. В данном случае эти сайты часто перечисляют продукты на основе даты публикации: сначала самые новые. Тем

самым увеличивается вероятность того, что больше пользователей будут взаимодействовать с новым продуктом. При этом первостепенным фактором, влияющим на генерацию рекомендаций, являются свойства самого продукта, его характеристики.

4.1.2 Холодный старт для пользователя

Холодный старт для пользователя или посетителя означает, что новый пользователь впервые регистрируется в системе. При этом о пользователе нет никаких сведений, в том числе его истории взаимодействий с продуктами, поэтому система рекомендаций не знает о его личных предпочтениях. Знакомство программного продукта (сайта, онлайн-магазина) с новыми посетителями имеет решающее значение для дальнейшего создания для него отличного пользовательского опыта.

В этом случае в течение некоторого периода времени система рекомендаций должна предоставлять пользователю рекомендации, при составлении которых не используются прошлые взаимодействия пользователя с продуктами, так как их еще недостаточно для составления точных прогнозов или же их нет вообще.

Основная стратегия работы с новыми пользователями — попросить предоставить их некоторую информацию о своих предпочтениях в продукте для создания первоначального профиля пользователя: таким образом система будет иметь некоторую отправную точку для работы с этим пользователем. Такая информация обычно собирается во время регистрации или первого посещения пользователя системы, при этом пользователю может предлагаться самому ввести некоторые данные, или же данные собираются извне: например, из его аккаунтов в социальных сетях. При этом важно найти золотую середину между продолжительностью процесса регистрации пользователя, которая теперь включает в себя некоторое анкетирование и, если будет слишком длинной, может привести к тому, что многие пользователи будут отказываться от нее, и объемом начальных данных, необходимых для корректной работы системы рекомендаций.

Самым первым шагом для решения проблемы холодного старта для пользователя является применение стратегии, основанной на популярности. Для этого из всех продуктов определяются самые высоко оцененные другими пользователями. При этом популярность продуктов можно определять в глобальном смысле — например, те продукты, у которых самый высокий рейтинг за все время, либо локально — например, те продукты, что были популярны в последнее время, или в данном регионе. На следующем шаге можно сузить информацию, показываемую новому пользователю. Например,

отображать ему продукты, выбранные на предыдущем шаге, то есть самые рейтинговые (популярные) товары. После этого пользователь наконец может оценить предложенные ему продукты. Оценка может определяться, например, по кликам на определенные товары, или по выставленному ему рейтингу. После таких нескольких оценок во время первого посещения информационного ресурса можно выявить фактическую сферу активного интереса пользователя и составлять новые рекомендации на основе данной собранной информации.

4.2 Задача нахождения пользователя с похожими предпочтениями

4.2.1 Коллаборативная фильтрация относительно пользователей

Коллаборативная фильтрация бывает двух видов: относительно пользователей — заключается в поиске пользователей с похожими предпочтениями, при этом не нужно никакой информации о пользователях, и относительно продуктов — ищутся продукты, похожие на уже понравившиеся пользователю (не нужно никакой информации о продуктах).

Наиболее распространенной является коллаборативная фильтрация относительно пользователей, так как часто информации о пользователях нет, и удобнее использовать только имеющиеся данные о продуктах. Далее не ограничивая общности под коллаборативной фильтрацией имеется в виду коллаборативная фильтрация относительно пользователей.

Для коллаборативной фильтрации в первую очередь необходимо найти в наборе данных наиболее похожего по предпочтениям на исходного пользователя человека. Второй шаг — оценка предпочтений найденного похожего пользователя, чтобы предсказать, какую оценку исходный пользователь поставит некоторому продукту: если оценка предполагаемо высокая — продукт может быть рекомендован пользователю. Отсюда возникает проблема нахождения степени «похожести» одного пользователя на другого.

Изначально для каждого пользователя составляется вектор предпочтений. Если пользователь оценил продукт, в вектор записывается его рейтинг, если нет — ноль. Таким образом получается матрица, где в строках — предпочтения каждого пользователя, а в столбцах — вектора оценок пользователя для каждого продукта.

Наиболее широко используемым методом измерения сходства является косинусное сходство, так как оно хорошо подходит для разреженных матриц.

4.2.2 Косинусное сходство

Косинусное сходство — это показатель, используемый для измерения того, насколько похожи два элемента или документа независимо от их размера [17]. Он измеряет косинус угла между двумя векторами, проецируемыми в многомерное пространство. Это позволяет измерить сходство документов любого типа. Благодаря многомерному массиву можно использовать любое количество переменных (которые рассматриваются как измерения), что, в свою очередь, поддерживает документы большого размера.

Математически косинус угла между двумя векторами получается из скалярного произведения двух векторов, деленного на произведение длин двух векторов.

Косинусное сходство двух векторов приведено на формуле (4.1) [17]

$$similarity(p,q) = \cos\Theta = \frac{p \cdot q}{\|p\|\|q\|} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}}, \quad (4.1)$$

где *similarity (p, q)* — косинусное сходство *n*-мерных векторов *p* и *q* с компонентами p_i и q_i соответственно.

Так как косинусное сходство определяется как косинус между двумя векторами, то результат всегда будет в диапазоне от -1 до 1, где -1 (угол между векторами 180 градусов — они противоположно направлены) показывает, что два элемента не похожи, а 1 (угол между векторами ноль градусов, а значит, они сонаправлены) означает, что два элемента похожи [17].

Таким образом, чтобы найти человека с похожими предпочтениями, нужно найти косинусы между вектором рейтингов исходного пользователя и векторами других пользователей, а затем выбрать максимальный из найденных косинусов.

Вектор, соответствующий этому косинусу (то есть тот, с которым находилось косинусное сходство для исходного вектора предпочтений пользователя), — это вектор предпочтений пользователя, наиболее «похожего» на исходного.

4.3 Хранилище таблиц Azure Table storage

Так как в приложение будут постоянно добавляться новые пользователи и ставить рейтинги фильмам, то необходимо использовать некоторое хранилище для этих данных.

Хранилище таблиц Azure — это сервис, который позволяет хранить нереляционные структурированные данные, или структурированные данные NoSQL, в облачном хранилище ключей/значений без заранее определенной структуры базы данных [18].

Благодаря тому, что табличное хранилище не имеет схемы, данные можно легко адаптировать по мере развития потребностей приложения. Главным преимуществом табличного хранилища перед традиционной базой данных SQL для аналогичных объемов данных является быстрота доступа и экономичность ресурсов.

Azure Table storage позволяет хранить различные гибкие наборы данных, например, пользовательские данные, и другую необходимую для приложения информацию.

Данный сервис позволяет сохранять любое количество сущностей в таблице, а учетная запись Azure позволяет создавать любое количество таблиц. Сущность — это набор свойств, подобных строке базы данных [18], а свойства — пары ключ-значение. У каждой сущности есть три системных свойства: ключ раздела, ключ строки и метка времени. Ключ раздела нужен для того, чтобы объекты, принадлежащие одному разделу, можно было вставлять/обновлять в атомарной операции и запрашивать для чтения быстрее. Ключ строки объекта — это его уникальный идентификатор в разделе. Таблица представляет собой набор сущностей, при этом засчет гибкости хранилища в одной таблице можно хранить совершенно разные сущности, с разным набором свойств [18].

Хранилище таблиц Azure отлично подходит для наборов данных, которые не требуют внешних ключей, сложных соединений с другими таблицами, процедур и могут быть денормализованы для быстрого доступа к нужной информации.

Получить доступ к данным можно с использованием протокола OData и запросов LINQ с помощью библиотек .NET.

Таблицы масштабируются по мере увеличения набора данных, поэтому Azure Table storage хорошо подходит для хранения даже огромного количества информации.

4.4 Проектирование пользовательского приложения

4.4.1 Диаграмма деятельности

Для того чтобы отобразить последовательность взаимодействий системы рекомендаций и пользователя была создана диаграмма деятельности, которая приведена на рисунке 4.1.

При запуске приложения пользователя просят ввести имя. Если такой пользователь есть в системе, то он может ввести одну из трех команд: получить рекомендации, попросить помощь или выйти из приложения. Если пользователь выбирает первый вариант, то первым шагом является загрузка данных из его профиля: по ранее выставленным рейтингам система определяет предпочтения данного пользователя. Далее для него ищется пользователь с похожими предпочтениями из набора данных MovieLens, затем для найденного пользователя подбирается список из пяти фильмов, которые ему наиболее вероятно понравятся. Эти фильмы система просить оценить: новые данные с выставленными рейтингами сохраняются в его профиль, и в следующий раз помогут в составлении более точных рекомендаций. Если выбран второй вариант из м са для получения рекомендаций жения. В случае, когда пользов никаких данных, ему выводитс льмов, которые он может оцени профиль. После такого анке нее упомянутые команды, и ал

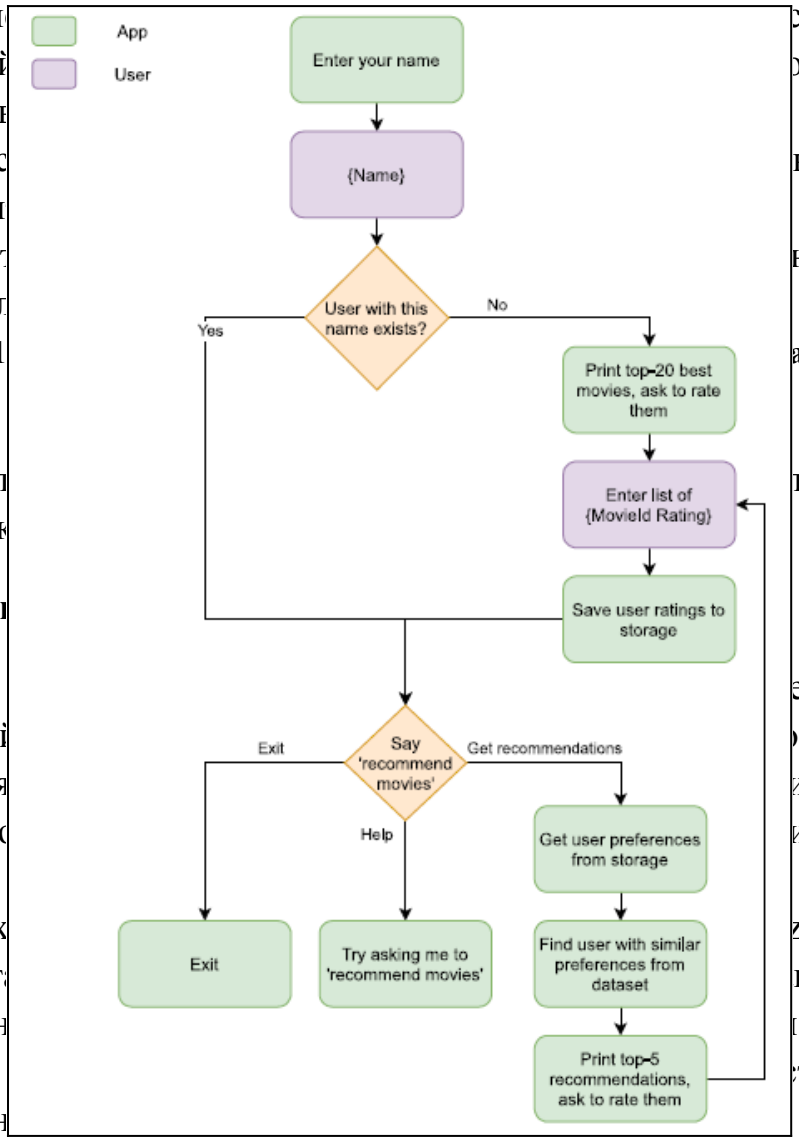


Рисунок 4.1 ателя с системой

Рассмот приложении при выполнении к

4.4.2 Хранени

Чтобы екомендации для пользователей о предыдущих предпочтения и, проставленные фильмам, ко и которые ему понравились.

Для сох zure Table storage. Хранилище та к как для данных не нужны вн таблицами, и в целом Azure ставляет быстрый доступ к данн

Чтобы получить доступ к сервису Azure Table storage необходима подписка Azure и ресурс Storage account, созданный на портале Azure. На рисунке 4.2 представлен созданный на портале Azure ресурс Storage account.

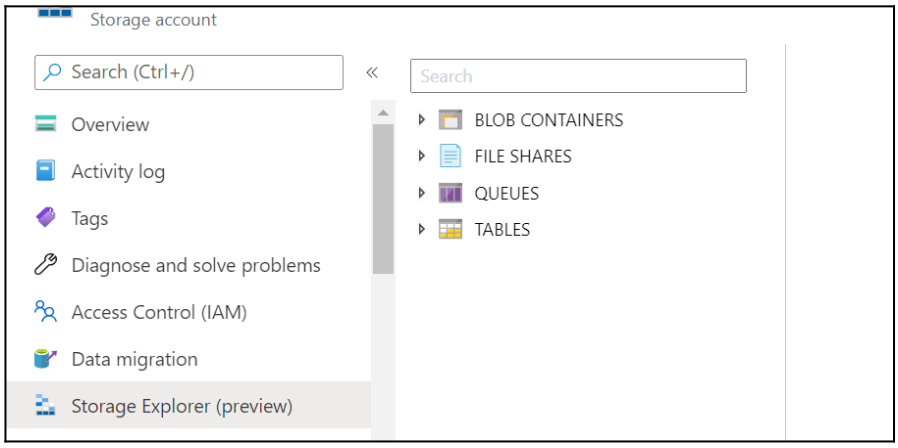


Рисунок 4.2 — Ресурс Storage account на портале Azure

Для начала работы с Azure Table storage в приложении необходимо установить NuGet пакет Microsoft.Azure.Cosmos.Table. Далее на портале Azure в разделе Access keys ранее созданного ресурса берется строка подключения к хранилищу, она записывается в файле приложения и нужна для создания объекта класса CloudStorageAccount, который в свою очередь позволяет создать объект класса CloudTableClient, отвечающий за все операции с таблицами. Он позволяет получать таблицы (а также создавать, если таблицы не существует, с помощью метода CreateIfNotExists) и сущности, хранящиеся в хранилище таблиц.

Реализация подключения к хранилищу и операций с таблицей и данными в ней представлена в классе UserStorage (приложение Б).

Чтобы определить сущность, которая будет представлять объект, хранящийся в таблице, необходимо создать класс, который наследуется от класса TableEntity, при этом класс должен иметь конструктор без параметров. Реализация класса UserRatingEntity также представлена в приложении Б. В качестве ключа строки используется имя (логин) пользователя, а ключ раздела для всех пользователей — «Users». Ключ строки и ключ раздела однозначно идентифицируют каждого пользователя в таблице. При этом объекты с одинаковыми ключами раздела запрашиваются быстрее, чем объекты с разными ключами раздела. Пользовательские свойства сущностей, которые сохраняются в таблице, должны быть общедоступными свойствами класса сущности и поддерживать как чтение (метод get), так и запись (метод set) значений. В классе UserRatingEntity такое свойство одно — RatingsInternal. Данное свойство имеет тип string, и получает свое значение путем JSON сериализации списка объектов класса Rating. Класс Rating представлен двумя

свойствами: MovieId — уникальный идентификатор фильма и RatingValue — значение рейтинга от одного до пяти, выставленного данным пользователем соответствующему фильму.

Все операции с данными осуществляется посредством использования объекта класса CloudTableClient и вызова его метода Execute, которому в качестве параметра передается объект класса TableOperation — табличная операция. Для вставки в таблицу информации о новом пользователе используется метод InsertOrReplace класса TableOperation, для обновления информации о рейтингах, выставленных уже существующим пользователем — Replace, для получения записи по ключу — Retrieve.

4.4.3 Составление матрицы предпочтений пользователей

Так как в процессе составления прогнозов для пользователя, вошедшего в систему, необходимо найти наиболее похожего по предпочтениям на исходного пользователя человека в наборе данных MovieLens, то, как ранее описывалось, для каждого пользователя составлялся вектор предпочтений.

Так как набор данных MovieLens содержит информацию о пятидесяти восьми тысячах фильмах и двухстах восьмидесяти тысячах пользователях, то хранение в матрице предпочтений для каждого пользователя вектора с рейтингами на все фильмы, где, если пользователь оценил фильм, в вектор записывается проставленный им рейтинг, а если нет — ноль, будет ресурсоемко. Во-первых, файл занимает много памяти, во-вторых операции чтения/записи также будут занимать много времени. Несмотря на то, что запись в этот файл будет осуществляться один раз при подготовке данных, чтение всех данных из этого файла будет необходимо при входе пользователя в приложение и запросе им рекомендаций, а долгое ожидание для пользователя недопустимо.

Для решения этой проблемы фильмы были сгруппированы по «схожести». В качестве показателя «схожести» был выбран жанр фильма. Таким образом для каждого пользователя хранится вектор размерностью девятнадцать (количество возможных жанров, перечисленных в главе 3). Если пользователь оценил фильм, то проставленный им рейтинг суммируется в компоненту вектора, соответствующую жанру этого фильма. При этом если фильм имеет несколько жанров, то рейтинг суммируется в каждую из этих компонент.

Таким образом получается матрица, где в строках — предпочтения каждого пользователя, а в столбцах — вектора оценок пользователя для каждого жанра. Так как матрица таким образом возможно получится разреженной, то самым лучшим методом, подходящим для получения степени «похожести» векторов, является косинусное сходство.

Поскольку потенциально могут использоваться разные шкалы рейтингов с разных источников (например, сайтов) для оценки фильмов, а также потому что будет использоваться косинусное сходство, вектора нормализуются. Таким образом получаются вектора с компонентами, значения которых вещественные числа, находящиеся в диапазоне от нуля до единицы.

Полученная матрица предпочтений записывается в файл user_ratings.csv, при этом каждая строчка начинается с идентификатора пользователя, а далее записывается сам вектор предпочтений для данного пользователя.

4.4.4 Решение проблемы холодного старта для пользователя

Если пользователь первый раз зашел в приложение, то о нем еще нет никаких данных: какие ему фильмы нравятся, какие наоборот, что он уже смотрел, а что нет, и так далее. В этом случае система рекомендаций не может сделать прогноз, так как нет начальных данных, на которых должна основываться рекомендация. Основная стратегия работы с новым пользователем в таком случае — попросить предоставить его некоторую информацию о своих предпочтениях в фильмах для создания первоначального профиля пользователя: при этом система будет иметь некоторую отправную точку для работы с этим пользователем. Вся эта информация собирается в процессе регистрации. Диаграмма деятельности данной стратегии приведена на рисунке 4.3. Таким образом после входа нового пользователя в приложение должен быть запрос на предоставления некоторых начальных данных. Для этого ему выводится список наиболее популярных фильмов.

Рисунок 4.3 — Диаграмма деятельности процесса работы с новым пользователем

Список этих фильмов составляется заранее путем прохода по всем рейтингам из файла ratings.csv набора данных MovieLens, группировки этого списка по фильмам, и сортировки по наибольшему суммарному рейтингу для каждого фильма. Полученный список записывается в файл best_movies.csv для быстрого доступа к данным — таким образом для каждого нового пользователя

318	Shawshank Redemption, The (1994)	Crime|Drama
356	Forrest Gump (1994)	Comedy|Drama|Romance|War
296	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller
593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller
2571	Matrix, The (1999)	Action|Sci-Fi|Thriller
260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
527	Schindler's List (1993)	Drama|War
480	Jurassic Park (1993)	Action|Adventure|Sci-Fi|Thriller
2959	Fight Club (1999)	Action|Crime|Drama|Thriller
110	Braveheart (1995)	Action|Drama|War

не придется заново находить самые популярные фильмы. Пример представленных данных в файле best_movies.csv отображен на рисунке 4.4.

Рисунок 4.4 — Информация о лучших фильмах

В первом столбце записан уникальный идентификатор фильма, во втором — название фильма, в третьем — жанры фильма, записанные через разделитель «|».

После того, как пользователю выведен список из двадцати лучших фильмов в системе, ему предлагается оценить какие-нибудь из этих фильмов. Скорее всего пользователь смотрел некоторые из этих фильмов, или слышал что-нибудь о сюжете, и может составить свое мнение о кинокартине. Если же нет, то можно сделать выводы, основываясь на жанре фильма. Таким образом, пользователь выставляет рейтинги по шкале от одного до пяти данным фильмам. При этом совершенно не обязательно оценивать все двадцать фильмов, достаточно пяти, чтобы система рекомендаций могла сделать достаточно эффективный прогноз насчет предпочтений пользователя.

4.4.5 Нахождение пользователя с похожими предпочтениями

Если пользователь раньше использовал приложение, то ему не нужно анкетирование, и нет необходимости выставлять рейтинги фильмам из списка двадцати лучших в системе, так как он уже делал это раньше — при регистрации в системе. Если же пользователь новый, то после того, как ему был выведен список из двадцати лучших фильмов, и он их оценил, система рекомендаций имеет достаточно информации для составления прогнозов, поэтому пользователь может перейти к следующему шагу.

Если пользователь изъявляет желание получить рекомендации, то для начинается процесс прогнозирования. В первую очередь загружается информация о выставленных рейтингах из хранилища Azure Table storage. Для этого используется метод Retrieve класса TableOperation, с помощью которого по ключу (имя или логин пользователя) из таблицы выбирается список рейтингов фильмам, ранее выставленных данным пользователем. Для всех операций с профилем пользователя был создан класс UserProfile, который в своих методах работает с объектом класса UserStorage (приложение Б), который в свою очередь непосредственно работает с Azure Table storage. Полученный список рейтингов передается в метод PredictTop5 класса Predictor (приложение В), созданного для реализации функционала, связанного непосредственно с прогнозированием.

В методе PredictTop5 сначала вызывается метод FindSimilarUserId, в котором для пользователя находится идентификатор пользователя из набора данных MovieLens, наиболее похожего на него по предпочтениям в фильмах.

Для этого рейтинги пользователя преобразуются в вектор предпочтений, так же как ранее это делалось для каждого пользователя из набора данных MovieLens при составлении файла user_ratings.csv: если фильм был оценен пользователем, то в вектор записывается значение рейтинга, если нет — ноль. Затем вектор нормализуется. Далее для полученного вектора и каждого вектора из файла user_ratings.csv находится косинусное сходство.

Для вычисления косинусного сходства использовалась платформа Accord.NET, в частности библиотека Accord.Math. Accord.NET предоставляет методы для статистического анализа, машинного обучения, обработки изображений и компьютерного зрения для .NET приложений. Платформа разделена на библиотеки, доступные для установки как пакеты NuGet. Библиотека Accord.Math предоставляет численные функции и инструменты для работы с матрицами. Для нахождения косинусного сходства двух векторов использовалась структура Cosine и ее метод Similarity. Среди всех полученных значений находится максимальное — пользователь, которому принадлежит соответствующий вектор предпочтений, и есть самый похожий на исходного. После этого для найденного пользователя вычисляется список рекомендаций из пяти фильмов, которые наиболее вероятно ему понравятся, и которые исходный пользователь еще не оценил. Для этого используется ранее обученная с помощью технологий ML.NET модель.

4.4.6 Использование обученной модели

Модель, хранящаяся локально (однако возможно хранение и удаленно), может быть использована в других приложениях. В первую очередь ее надо загрузить: для этого достаточно использования функции Load, куда в качестве параметра передается путь к файлу формата zip с обученной моделью. Затем для вывода более подробной информации о фильмах загружаются данные из файла movies.csv. Наконец, делается прогноз. Для этого используется объект класса PredictionEngine, куда в качестве двух универсальных параметров передается тип тестовых данных, для которых будет осуществляться прогноз (MovieRating), и тип прогноза (MovieRatingPrediction). Далее вызывается метод Predict, куда в качестве аргумента передаются входные данные, для которых будет делаться прогнозирование.

Так как в приложении необходимо вывести лучшие пять фильмов для пользователя, то метод Predict вызывается в цикле: осуществляется перебор всех фильмов из файла movies.csv, для каждого из них создается объект класса MovieRating, инициализированный идентификаторами пользователя и текущего фильма, и для каждого такого объекта делается прогноз. Далее все объекты сортируются по убыванию свойства Score объекта Prediction полученного

прогноза, и выбираются пять первых элементов из тех фильмов, что пользователь еще не смотрел (информации о них нет в его профиле).

Далее пользователю предлагается оценить рекомендованные фильмы по шкале от одного до пяти для того, чтобы данные о его предпочтениях расширялись: чем больше данных о пользователе будет у системы рекомендаций, тем точнее будут дальнейшие прогнозы. Это и есть главное преимущество построенной системы рекомендаций: чем чаще пользователь будет заходить в приложение и запрашивать новые рекомендации, а затем давать обратную связь системе о том, понравились ли ему предложенные фильмы, тем точнее будут рекомендации в следующий раз.

ВЫВОДЫ

1. Изучена и решена проблема холодного старта для пользователя для системы рекомендаций.
2. Исследована коллаборативная фильтрация относительно пользователей.
3. Решена задача нахождения пользователя с похожими предпочтениями с применением косинусного сходства.
4. Изучены и применены возможности хранилища таблиц Azure Table storage для хранения данных о рейтингах, выставленных пользователями.
5. Спроектировано демонстрационное приложение для получения списка из пяти фильмов, которые с наибольшей вероятностью понравятся пользователю, с использованием обученной модели ML.NET.
6.

ГЛАВА 5 ТЕХНОЛОГИИ РАЗРАБОТКИ БОТ-ПРИЛОЖЕНИЯ

5.1 Обзор бот-приложений

Боты — это программы, предназначенные для решения задач пользователя и использующие удобный для них интерфейс. [19]. По мере того, как общение с компьютерами становится всё более привычным и естественным, возрастает роль общения путем диалога на естественном языке. Со стороны пользователя общение с ботом больше напоминает взаимодействие с человеком или умным роботом, чем работу с компьютером. Такое общение может представлять собой как простой диалог с вопросами и немедленными ответами, так и сложный интеллектуальный разговор, в ходе которого пользователю предоставляется доступ к различным сервисам.

Боты представляют собой веб-сервисы и соответственно могут делать то же самое, что и другие типы программного обеспечения: работать с файлами, использовать базы данных, подключаться к различным API и предоставлять свой, решать вычислительные задачи. В то же время уникальными их делает использование механизмов, обычно предназначенных для общения между людьми. Боты полезны в бизнесе, потому что они экономят время и усилия пользователей, автоматизируя различные процессы, которые им необходимо осуществить в ходе использования приложения.

Выделяют следующие основные типы бот-приложений: чат-бот, коммерческий бот, вредоносный бот [19].

Чат-бот — это программа, главная задача которой заключается в ведении диалога с пользователем, отвечая на их вопросы и задавая свои. Это самый распространенный тип ботов, который используется в различных сферах: в службах поддержки, помогая решить простые вопросы, например, такие как смена пароля; для поиска информации, например, прогноза погоды, помощи в выборе подарка; в сфере путешествий, при этом чат-бот предлагает направления путешествия, доступные рейсы, отели; для помощи работодателям и соискателям в процессе поиска работы и подбора кадров; как персональные помощники и так далее.

Коммерческие боты завлекают клиентов и рассылают пакеты информационного контента: новостей и акционных предложений или актуальных товаров.

Вредоносные боты применяются для мошенничества с целью нечестного заработка. Их также могут использовать для координации сетевых атак на компьютеры.

5.2 Создание бота с использованием Azure Bot Service и Azure Bot Framework

Azure — это полноценная облачная платформа, на которой можно разместить существующие приложения и упростить разработку новых приложений [20]. Azure интегрирует облачные сервисы, необходимые для разработки, тестирования, развертывания и управления приложениями, используя при этом все преимущества эффективности облачных вычислений. Для управления всеми сервисами используется портал Azure, который позволяет их настраивать, а также отслеживать состояние ресурсов. Сервисы Azure предоставляют REST API, что позволяет использовать их в любых операционных системах, устройствах и платформах.

Azure Bot Service и Bot Framework предоставляют средства для сборки, тестирования, развертывания и управления интеллектуальными ботами в единой среде. С помощью данной платформы можно создавать ботов, которые используют речь, понимают естественный язык, могут обрабатывать вопросы и давать на них ответы и многое другое. Бот — это веб-сервис, который реализует интерфейс для диалога с пользователем и взаимодействует с Bot Framework Service (один из компонентов Azure Bot Service и Bot Framework) для отправки и получения сообщений и событий [21].

Бот, как и любое другое сложное приложение должно быть протестировано перед публикацией. Для этого есть несколько способов [21]:

- тестирование бота локально с помощью эмулятора — приложения, которое предоставляет интерфейс чата и средства отладки, чтобы помочь понять, как и почему возникает проблема с ботом;
- проверка бота в Интернете: после его развертывания на Azure бот можно протестировать с помощью канала Web Chat, доступного на портале Azure;
- модульное тестирование (англ. unit test).

Когда бот разработан и протестирован, его можно опубликовать в Azure или в собственном веб-сервисе или центре обработки данных: это первый шаг к тому, чтобы бот мог использоваться на сайте или внутри каналов для чата. Бот может быть подключен к таким каналам, как Web Chat, Skype, Telegram и другим. Bot Framework выполняет большую часть работы, необходимой для отправки и получения сообщений со всех этих различных платформ, — бот-приложение получает унифицированный, нормализованный поток сообщений независимо от количества и типа каналов, к которым он подключен.

5.3 Возможности Azure Cloud Services для бот-приложений

Кроме написания основной части бот-приложения, которое представляет собой веб-сервис, реализующий интерфейс для отправки и получения сообщений и событий, для расширения функционала бота можно использовать некоторые дополнительные компоненты, которые предоставляются платформой Azure Cloud Services. Основные сервисы, которые используются при создании бот-приложения:

- Azure Monitor — сбор, анализ и обработка данных телеметрии из облачных и локальных сред;
- Azure Cognitive Services — распознавание речи, поиск в Интернете, принятие решений и другие, в том числе LUIS — обработка естественного языка и распознавание намерений пользователя.

5.3.1 Мониторинг (Azure Monitor)

Azure Monitor обеспечивает максимальную доступность и производительность приложений и сервисов, предоставляя функционал для сбора, анализа и обработки данных телеметрии, что помогает выявлять проблемы в приложениях и зависящих от них ресурсах. Все данные, собранные Azure Monitor, входят в один из двух основных типов — метрики и журналы. Метрики — это числовые значения, описывающие некоторый аспект системы в определенный момент времени [22]. Данные телеметрии, такие как события и трассировки, а также данные о производительности хранятся в виде журналов, чтобы их можно было объединить для анализа.

Сервис Application Insights (компонент Azure Monitor) осуществляет контроль доступности, производительности и использования как локальных так и расположенных в облаке веб-приложений. Он использует мощную платформу анализа данных в Azure Monitor, чтобы предоставить подробные сведения об операциях, выполняемых в приложении, и диагностировать ошибки.

Также Azure Monitor предоставляет функционал для реагирования на критические условия, обнаруженные в собираемых данных. Оповещения в Azure Monitor позволяют заранее уведомлять администратора о критических состояниях в приложении, а также пытаются предпринять корректирующие действия.

5.3.2 Когнитивные сервисы (Azure Cognitive Services)

Azure Cognitive Services — это API и сервисы, предназначенные для использования при создании интеллектуальных приложений без

непосредственного использования искусственного интеллекта [23]. Когнитивные сервисы разделены на различные категории:

- Принятие решений — добавление рекомендаций в приложение для эффективного и обоснованного принятия решения.
- Язык — возможность приложения распознавать естественный язык с помощью предварительно построенных сценариев, и оценивать намерения пользователей, а также перевод текста на поддерживаемые языки.
- Поиск — добавление API поиска Bing в приложение, чтобы иметь возможность просматривать множество веб-страниц, изображений, видео и новостей с помощью одного вызова API.
- Речь — преобразование речи в текст и наоборот, распознавание речи говорящего на основе аудиозаписей.
- Зрение — распознавание и идентификация фотографий, видео и рукописного ввода.

Cognitive Services, как группа сервисов, может не требовать никаких, некоторых или всех пользовательских данных для обученной модели.

Сервис, который предоставляет полностью обученную модель, можно рассматривать как черный ящик, то есть пользователю не надо знать, какие данные были использованы для его обучения. Некоторые сервисы позволяют предоставить свои собственные данные, а затем обучить модель. Это позволяет расширить модель, используя данные сервиса и собственные данные. Выход соответствует потребностям пользователя. Сервис может также потребовать данные для обучения модели. Например, в разрабатываемом бот-приложении использовался сервис распознавания речи LUIS, для которого необходимо предоставить собственные данные для обучения модели.

5.3.3 Распознавание естественного языка (LUIS)

Language Understanding (LUIS) — это облачный сервис, который применяет искусственный интеллект к тексту на естественном языке для прогнозирования общего смысла фразы и извлечения некоторой дополнительной информации [24]. Пользовательское приложение для LUIS — это любое диалоговое приложение, которое взаимодействует с пользователем на естественном языке для выполнения некоторой задачи. Примеры клиентских приложений включают в себя приложения для социальных сетей, настольные приложения с поддержкой речи, а также чат-боты.

Пользовательское приложение отправляет текст в API конечной точки приложения обработки естественного языка LUIS и получает результаты в виде объекта JSON. Взаимодействие пользовательского приложения с приложением LUIS включает следующие шаги [24]:

1. Пользовательское приложение отправляет высказывание пользователя в конечную точку LUIS в качестве HTTP-запроса.

2. LUIS применяет языковые модели машинного обучения к входному тексту и возвращает ответ в формате JSON с распознанным намерением. Ответ в формате JSON содержит текст запроса и наиболее вероятное намерение. Из запроса также могут быть извлечены некоторые дополнительные данные.

3. Пользовательское приложение использует ответ в формате JSON для принятия решений о том, как выполнять запросы пользователя.

Основой модели LUIS являются намерения — категории вводимого текста. Для каждого возможного намерения перед обучением модели необходимо предоставить примеры пользовательских фраз. Высказывания пользователя могут предоставить дополнительные данные, которые приложению LUIS впоследствии необходимо извлечь.

ВЫВОДЫ

1. Azure Bot Service и Bot Framework предоставляют средства для сборки, тестирования, развертывания и управления интеллектуальными ботами в единой среде.

2. Бот-приложение представляет собой веб-сервис, реализующий интерфейс для отправки и получения сообщений и событий.

3. Тестировать бот можно локально с помощью эмулятора Bot Framework, через интерфейс канала Web Chat и с помощью модульного тестирования.

4. Бот может быть подключен к пользовательским каналам, таким как Web Chat, Skype, Telegram и другим.

5. Azure Cloud Services предоставляют сервисы, которые могут быть использованы для расширения функционала бот-приложения: Azure AD B2C, Azure Monitor, Azure Cognitive Services, включая LUIS, и другие.

ГЛАВА 6 РАЗРАБОТКА БОТ-ПРИЛОЖЕНИЯ

6.1 Проектирование бот-приложения

На рисунке 6.1 представлена архитектура бот-приложения. На схеме отображено само бот-приложение, а также его зависимости и связи с внешними сервисами и компонентами.

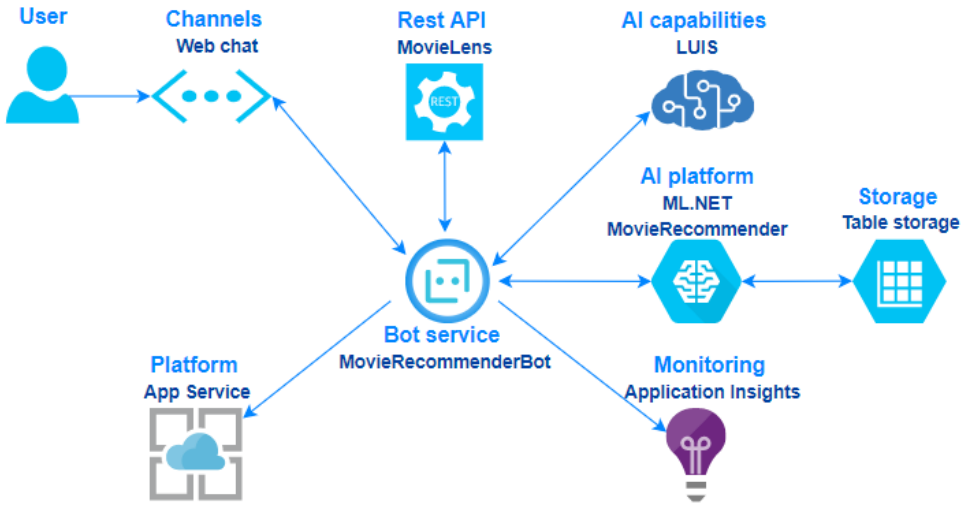


Рисунок 6.1 — Архитектура бот-приложения

Бот-приложение развертывается с помощью Azure App Service и выполняется в Azure. В процессе своей работы оно отправляет данные телеметрии в App Insights, которые в дальнейшем используются для анализа работы приложения.

Рассмотрим последовательность взаимодействия компонентов бота:

1. Пользователь не подключается непосредственно к боту, а взаимодействует с ним через канал, который используется для подключения бота к различным пользовательским приложениям, таким как Web Chat, Email, Skype и так далее.
2. Бот — это веб-сервис, который предоставляет API по /api/messages пути. Этот API используется для получения входящих сообщений от пользователя и ответа на них.
3. С помощью канала и затем бот-приложения пользователь делает запрос на получение рекомендаций.
4. Бот использует сервис распознавания естественного языка LUIS для обработки текстового запроса.
5. Бот интегрируется с ранее обученной с помощью платформы ML.NET системы рекомендаций.
6. Для получения и сохранения информации о предпочтениях (выставленных рейтингах) пользователя используется Azure Table storage.

7. Бот загружает постеры фильмов для интерактивного отображения рекомендаций, используя MovieLens API.

6.2 Принцип работы бота

Бот — это приложение, с которым пользователи взаимодействуют в разговорном режиме, используя текст, графику (например, интерактивные карточки или изображения) или речь. Сервис Bot Framework, компонент Azure Bot Service, отправляет информацию между пользовательскими приложениями, которые также называют каналами, и ботом.

В разговоре люди обычно говорят друг за другом, по очереди осуществляя свой «ход» в течении беседы. В сервисе Bot Framework «ход» представляет собой входящее действие пользователя, инициируемое при обращении к боту, и действие, отправляемое ботом в ответ. Таким образом «ход» — это своего рода обработка, связанная с поступлением какого-либо действия в диалоге. Действие поступает с HTTP запросом и представляется в виде JSON объекта, который затем десериализуется и отправляется для обработки в класс адаптера бота — наследника стандартного класса BotFrameworkHttpAdapter. Код класса адаптера AdapterWithErrorHandler разработанного приложения представлен в приложении Г. Адаптер инициализирует контекст «хода», добавляя в него сведения о самом действии, включая отправителя и получателя, канал и другие данные, необходимые для его корректной обработки, и передает его классу бота.

В классе бота, являющимся наследником стандартного класса ActivityHandler, который в свою очередь реализует интерфейс IBot, определены обработчики действий (реализация класса Bot отображена в приложении Д). Базовым обработчиком является метод OnTurnAsync — так называемый обработчик «хода». Вся обработка действий в диалоге начинается именно там, а затем для каждого полученного действия вызывается метод индивидуальной обработки в зависимости от его типа. Например, если пользователь инициирует действие, представляющее собой отправку сообщения, базовый обработчик перенаправит его обработчику OnMessageActivityAsync. Действие, генерируемое при добавлении новых пользователей в диалог, передается в метод OnMembersAddedAsync. Так как в дальнейшем бот подключается к каналу Web Chat, то удобно также переопределить метод OnConversationUpdateActivityAsync, который вызывается при добавлении или удалении пользователя из диалога, и в первом случае вызывает OnMembersAddedAsync. В разрабатываемом бот-приложении при добавлении нового пользователя, метод SendWelcomeMessageAsync класса Bot,

вызываемый упомянутыми выше обработчиками, выводит приветственное сообщение для пользователя, а также запрос на ввод имени (рисунок 6.2).

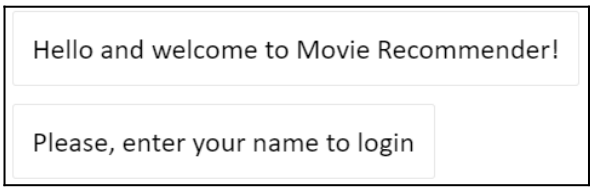


Рисунок 6.2 — Приветственное сообщение бота и запрос ввода имени

6.3 Библиотека диалогов

Диалоги предназначены для того, чтобы выполнять определенные задачи в определенном порядке. Порядок выполнения диалогов позволяет направлять разговор с пользователем. Диалоги можно вызывать различными способами: в ответ на сообщение пользователя или некоторые внешние события, а также из других диалогов. Библиотека диалогов предоставляет ряд встроенных функций: основные из них это каскадные диалоги и диалоги ввода.

Каскадные диалоги объединяют несколько шагов в последовательность, что позволяет боту передавать информацию, полученную на каждом этапе, на следующий шаг. В разработанном бот-приложении каскадный диалог состоит из следующих шагов:

1. Получение имени пользователя, вывод приветствия и проверка, есть ли он в системе: если о пользователе уже есть данные — переход к следующему шагу, если нет — загрузка двадцати наиболее популярных в системе фильмов и вывод их (рисунок 6.3).

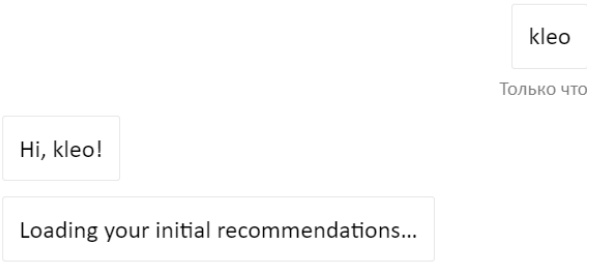


Рисунок 6.3 — Приветствие нового пользователя и вывод сообщения о загрузке начальных рекомендаций

2. Если пользователь новый, то ожидается что он оценит предложенные ему фильмы — для этого вызывается дочерний диалог (рисунок 6.4). Если нет — переход к следующему шагу.

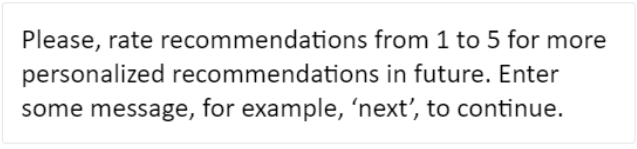


Рисунок 6.4 — Запрос оценить предложенные фильмы

3. . Сохранение выставленных рейтингов в случае нового пользователя, а также вывод сообщения с примером возможного запроса на получение рекомендаций (рисунок 6.5).

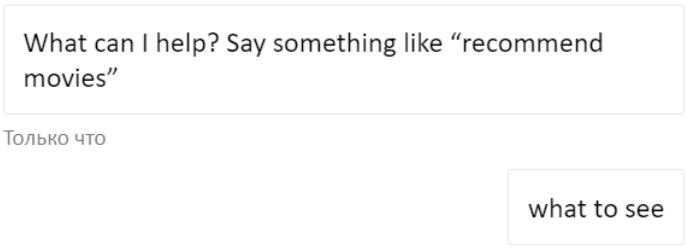


Рисунок 6.5 — Запрос выбора дальнейшего действия

4. Вывод рекомендаций пользователю, если был получен такой запрос (рисунок 6.6), и сообщения с просьбой оценить рекомендованные фильмы (вызывается ранее упомянутый дочерний диалог для сбора рейтингов).

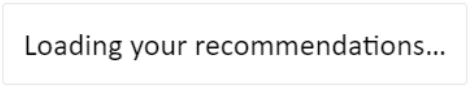


Рисунок 6.6 — Сообщение о загрузке рекомендаций

5. Сохранение выставленных рейтингов, повторное предложение своих услуг ботом.

Диалоги ввода представляют собой методы, запрашивающие у пользователя некоторую информацию, дожидающиеся ее получения и возвращающие ее. Такой диалог повторяется до тех пор, пока не получает ввод или не будет отменен извне. Библиотека диалогов содержит несколько типов диалогов ввода, каждый из которых используется для сбора различных типов ответов. Например, в разработанном приложении используется текстовый запрос TextPrompt совместно с объектом PromptOptions, у которого поле Prompt (собственно запрос на ввод) представляет собой сообщение с вопросом. Ответ пользователя должен представлять собой любое текстовое сообщение (рисунок 6.5).

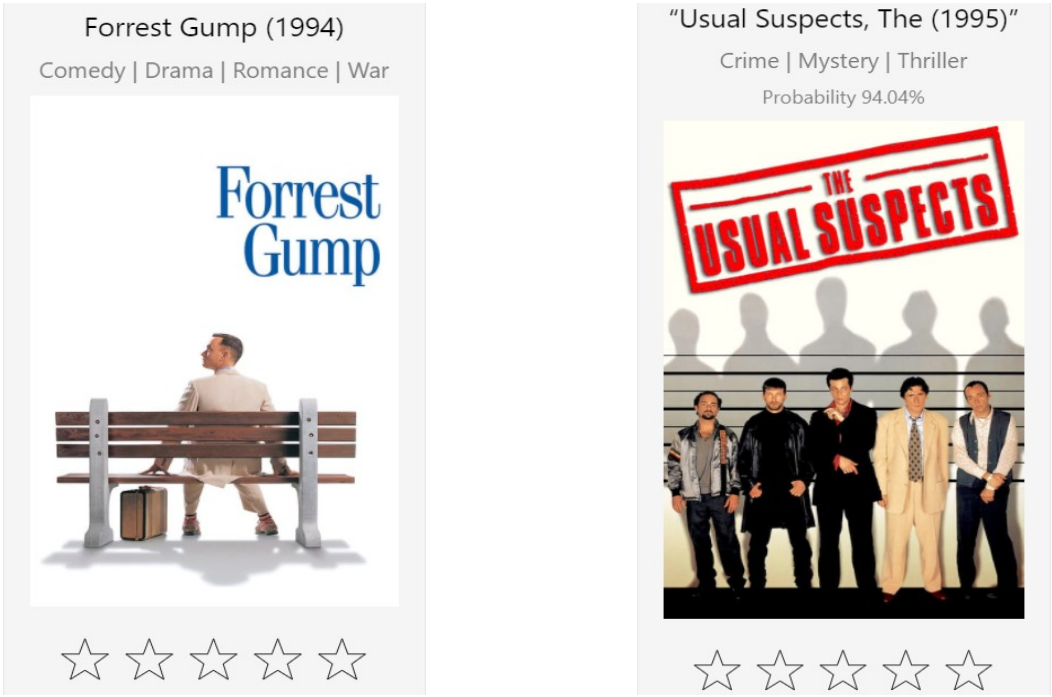
6.4 Форматированные карточки в сообщениях

Обмен сообщениями с пользователем является ключевой функцией любого бота. Форматированные карточки, предоставляемые сервисом Bot Framework, помогают обогатить опыт пользователей, позволяя им получать информацию от бота в виде списка или карусели интерактивных карточек, не ограничиваясь простыми текстовыми сообщениями.

Класс Activity представляет действие в диалоге и включает в себя свойство Attachments, которое содержит массив объектов Attachment — форматированные карточки и файлы мультимедиа внутри сообщения.

Для вывода информации о популярных фильмах для новых пользователей, а также рекомендованных фильмов после запроса пользователя на получение списка рекомендаций были использованы адаптивные карточки. Этот тип карточек может содержать текст, аудиозаписи, изображения, кнопки и поля для ввода. Для работы с адаптивными карточками в C# приложении устанавливается NuGet пакет AdaptiveCards. Адаптивная карточка представлется в виде объекта JSON, соответствующего определенной схеме.

Для вывода информации о фильме была создана схема movieCard.json, содержащая название, список жанров, и изображение постера (рисунок 6.7, а). Схема recommendationCard.json соответствует информации о рекомендованном фильме и представляет аналогичные movieCard.json данные, дополнительно выводится вероятность в процентах того, что фильм понравится пользователю на основе его предыдущих предпочтений (рисунок 6.7, б). Для того, чтобы пользователь мог оценить фильмы по шкале от одного до пяти, был создан компонент для пятизвездочного рейтинга. Обе схемы содержат элемент ColumnSet, который делит область карточки на столбцы (Column), позволяя элементам располагаться рядом с друг другом. Карточка содержит пять столбцов внизу, каждый столбец содержит изображение — звезду, а также объект типа Action.Submit — действие, отправляющее событие боту. В результате нажатия на столбец, боту приходит объект JSON, содержащий идентификатор фильма, соответствующего карточке, а также значение рейтинга, соответствующего номеру колонки: от одного до пяти.



а) **б)**

Рисунок 6.7 — Адаптивная карточка для вывода информации о: а) фильме; б) рекомендации

Чтобы отобразить список форматированных карточек в виде карусели (рисунок 6.8), свойству AttachmentLayout объекта Activity было задано значение Carousel (карусель).

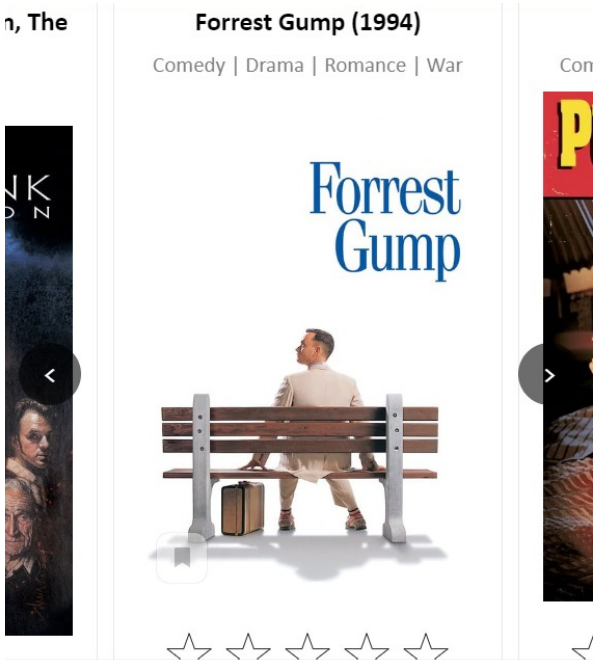


Рисунок 6.8 — Карусель форматированных карточек

6.5 Распознавание естественного языка

6.5.1 Создание приложения LUIS

Способность понимать, что пользователь хочет сказать и какое намерение вкладывает в свою фразу, обеспечивает более естественное ощущение разговора с ботом. LUIS, API распознавания речи, позволяет сделать так, чтобы бот мог понимать смысл пользовательских сообщений, использовать больше естественного языка при диалоге с пользователем и лучше направлять разговор.

Чтобы использовать LUIS, предварительно был создан аккаунт на портале luis.ai.

Далее было необходимо:

1. Нажать кнопку New app для создания нового LUIS приложения.
2. Для созданного приложения выбрать вкладку Manage, раздел Versions.
3. Нажать кнопку Import и выбрать из выпадающего списка Import as JSON.
4. Выбрать ранее подготовленный файл RecommendationLuis.json, расположенный в папке CognitiveModels проекта. Этот файл содержит четыре возможных намерения пользователя: GetRecommendations — забронировать билет, Help — получить помощь, Cancel — отменить текущее действие и

None — ничего. Намерения используются для того, чтобы понять, что имел в виду и что хочет получить пользователь, когда отправлял сообщение боту.

5. Обучить приложение.
6. Опубликовать приложение в рабочей среде.

Скриншот с портала LUIS для пунктов 1-3 приведен на рисунке 6.9.

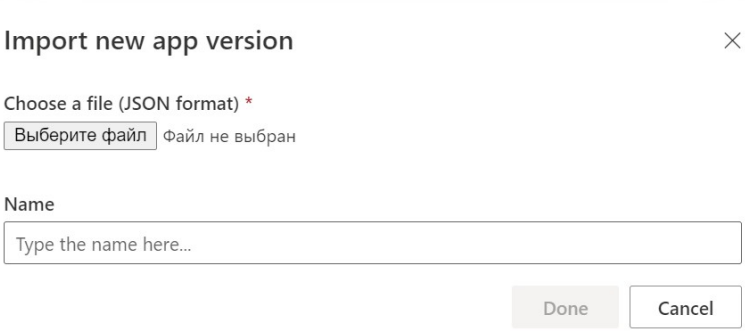


Рисунок 6.9 — Импортирование нового приложения на портал LUIS

6.5.2 Обучение приложения LUIS

Обучение приложения сервиса распознавания речи LUIS — это процесс, который позволяет распознавать естественный язык пользователя. После обучения приложения LUIS его надо протестировать с помощью примеров высказываний, чтобы проверить, правильно ли распознаются намерения и сущности (при их наличии). Если это не так, то приложение LUIS улучшается, дорабатывается, а затем снова обучается и тестируется. Таким образом, обучение и тестирование приложения LUIS — итерационный процесс, который необходимо запустить на портале LUIS. Для этого каждое намерение пользователя должно иметь по крайней мере одно высказывание перед началом обучения. Например, для намерения GetRecommendations в качестве примеров высказываний использовались фразы «get recommendations», «what to see», «make prediction for me» и некоторые другие со схожим смыслом.

Необходимые действия для обучения приложения на портале LUIS:

1. Открыть приложение, выбрав его имя на странице My Apps.
2. В приложении нажать кнопку Train.

При этом по завершении обучения появляется уведомление.

Скриншот с портала LUIS для пункта 2 с уведомлением о завершении обучения приведен на рисунке 6.10.

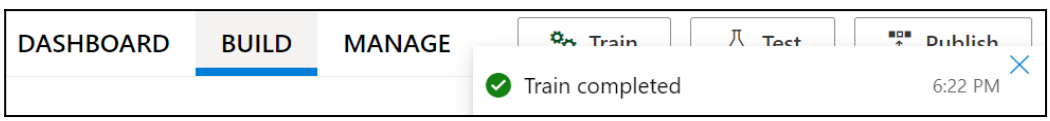


Рисунок 6.10 —Верхняя панель с кнопкой Train на портале LUIS и уведомление об окончании обучения приложения

6.5.3 Публикация обученного приложения в конечной точке

После завершения создания, обучения и тестирования приложения LUIS его необходимо сделать доступным для клиентского приложения, опубликовав в конечной точке.

На портале LUIS выполняются надо выполнить следующие шаги:

1. Нажать кнопку Publish (рисунок 6.11).

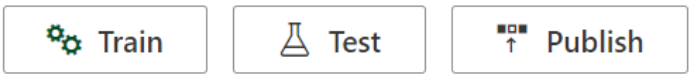


Рисунок 6.11 — Верхняя панель с кнопкой «Publish» на портале LUIS

2. Выбрать параметры для опубликованной конечной точки (рисунок 6.12) и нажать кнопку «Done».

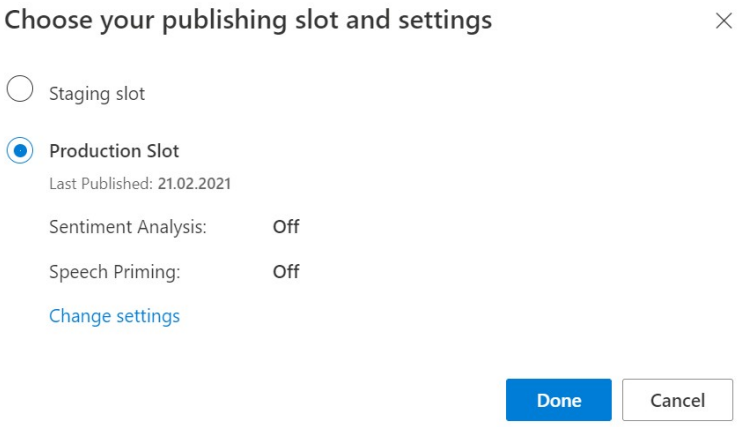


Рисунок 6.12 — Выбор типа публикации приложения LUIS

Когда приложение LUIS опубликовано, к нему можно получить доступ из бот-приложения.

Для этого необходимо:

1. Выбрать опубликованное приложение LUIS на портале https://www.luis.ai.
2. После открытия опубликованного приложения перейти на вкладку Manage в верхней панели.
3. Выбрать вкладку Settings в меню в левой части страницы и скопировать значение App ID.
4. Выбрать вкладку Azure Resources в том же меню и скопировать значения Location и Primary Key.
5. Далее обновляется файл настроек appsettings.json проекта бот-приложения: в секцию Luis добавляются раннее скопированные значения AppId и APIKey. Имя узла APIHostName должно иметь формат <location>.api.cognitive.microsoft.com.

6.5.4 Использование LUIS в бот-приложении

Для использования LUIS в бот-приложении необходимо установить NuGet пакет Microsoft.Bot.Builder.AI.Luis.

Чтобы подключиться к сервису LUIS, бот извлекает информацию, которая была добавлена в файл настроек приложения appsettings.json. Для этого был создан класс RecommendationRecognizer, который также запрашивает сервис LUIS, вызывая RecognizeAsync метод.

LUISGen — это инструмент для создания класса C# для намерений и сущностей, определенных в модели LUIS. Файл RecommendationLuis.cs приложения был сгенерирован с помощью данной утилиты и содержит описание класса RecommendationLuis, который используется для хранения полученных результатов распознавания сообщения пользователя при вызове метода RecognizeAsync<RecommendationLuis> из главного диалога, определенного в классе MainDialog.

Когда сервис LUIS для бота полностью настроен и подключен, его работу необходимо протестировать — для этого был использовано приложение Bot Framework Emulator. Для тестирования сервиса распознавания естественного языка LUIS в эмуляторе вводятся различные сообщения. Если наиболее вероятное намерение, распознанное LUIS, соответствует намерению GetRecommendations — получение рекомендаций, то для данного пользователя производится прогнозирование с использованием ранее обученной модели. Приложение LUIS обучено для различных вероятных запросов пользователя для получения рекомендаций, пример приведен на рисунке 6.13.

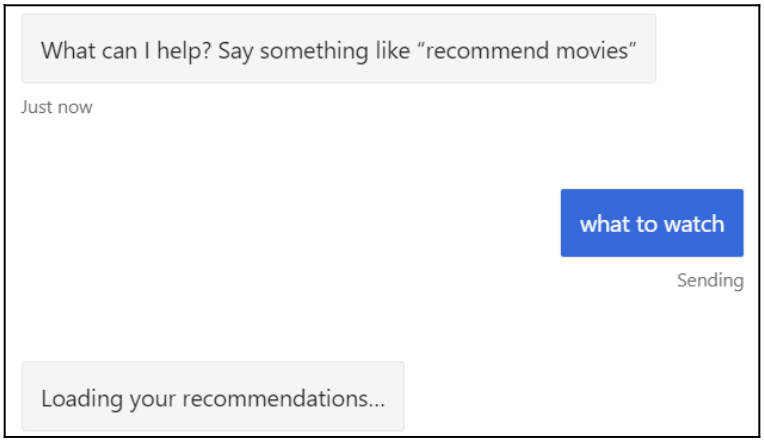


Рисунок 6.13 — Распознавание приложением LUIS намерения пользователя получить рекомендации

После получения рекомендаций, бот повторно предлагает свои услуги, и пользователь может либо получить новые рекомендации, либо закончить диалог — если бот распознает намерение «Cancel» (рисунок 6.14).

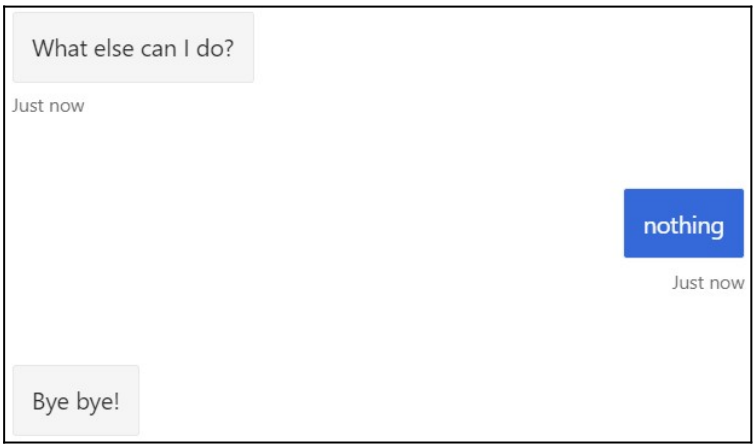


Рисунок 6.14 — Отказ пользователя от повторного использования услуг бота

Также обученная модель LUIS позволяет попросить помощи у бота, в этом случае пользователю выводится подсказка с примером запроса для получения рекомендаций (рисунок 6.15).

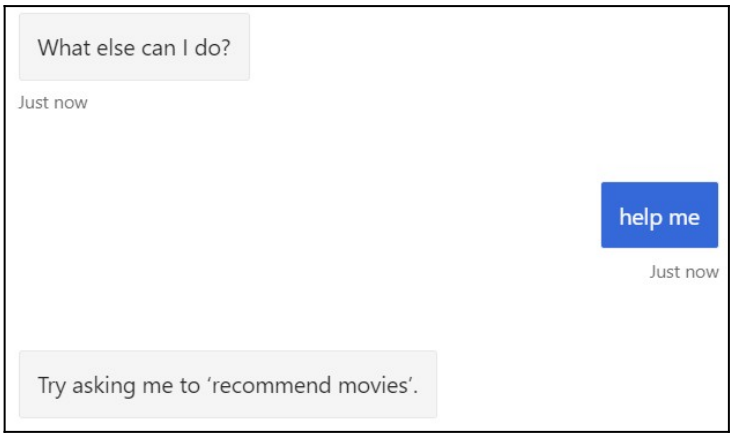


Рисунок 6.15 — Сообщение бота в случае распознавания намерения Help

В остальных же случаях, когда фраза пользователя не подходит ни под одно из трех перечисленных ранее намерений, бот относит ее к намерению «None» и выводит соответствующее сообщение (рисунок 6.16).

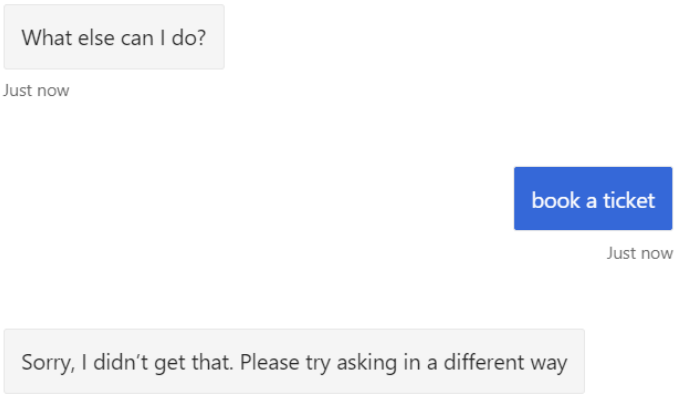


Рисунок 6.16 — Скриншот поведения бота в случае распознавания намерения None

6.6 Взаимодействие приложения с внешними API

Чтобы бот мог выводить фильмы в удобном интерактивном формате, который включает в себя не только название фильма, но и его постер, приложение подключается к MovieLens API.

В проекте создаются интерфейс IMoviePosterService и реализующий его класс MoviePosterService, затем данный сервис регистрируется с помощью процесса внедрения зависимости (англ. dependency injection). Через конструктор в данный класс внедряются сервис IHttpClientFactory — для настройки и создания экземпляра HttpClient, который используется непосредственно для отправки запросов к API, а также IOptions<UriOptions> и IOptions<LoginOptions> — объекты, содержащие необходимые данные, полученные из файла настроек appsettings.json, для подключения к API.

Когда необходимо вывести карточки с популярными фильмами новому пользователю или рекомендации фильмов при распознавании ботом такого намерения пользователя, приложение вызывает метод GetPosterLinkAsync сервиса IMoviePosterService для получения ссылки на изображение постера.

Сначала необходимо получить путь для постера: для этого приложение по идентификатору фильма из MovieLens API получает в виде JSON объекта информацию о фильме, в том числе путь для получения постера. Затем приложение формирует ссылку на постер: она состоит из адреса к популярной базе данных фильмов TMDb, желаемого размера изображения, и ранее полученного пути к постеру.

6.7 Развертывание бота в Azure

Перед развертыванием бота в Azure необходимо собрать проект в режиме выпуска (выбрать «Release» из списка конфигураций решения в Visual Studio). Чтобы развернуть бот из командной строки Windows, был установлен интерфейс командной строки Azure (Azure CLI — Command Line Interface). После входа на портал Azure в командной строке вводятся следующие команды:

1. «az account set --subscription "<azure-subscription>"» — установка подписки для развертывания бота.

2. «az ad app create --display-name " MovieRecommenderAD" --password "password" --available-to-other-tenants» — создание приложения Azure Active Directory, которое позволит боту подключаться к каналам, а также определит

параметры подключения OAuth для аутентификации пользователя и создания токена, который бот использует для доступа к защищенным ресурсам от имени пользователя. Значение name — имя приложения, password — пароль для него, а флаг available-to-other-tenants нужен для работы бота с каналами.

3. «az deployment create --template-file "template-with-new-rg.json" --location "West Europe" --parameters appId="<app-id>" appSecret="<password>" botId="MovieRecommenderBot" "botSku=F0 newAppServicePlanName=" src_plan_movie_recommender_bot" newWebAppName="movie-recommender-bot" groupName="rg_movie_recommender_bot" groupLocation="West Europe" newAppServicePlanLocation="West Europe"» — создание сервиса приложения для бота. Здесь template-with-new-rg.json — полный путь к файлу с таким именем, который находится в папке DeploymentTemplates проекта. Этот файл — готовый шаблон Azure Resource Manager (диспетчера ресурсов Azure). Параметр location — расположение приложения, appId — значение ключа из JSON объекта, выведенного на предыдущем шаге, appSecret — пароль также с предыдущего шага, botId — уникальное имя бота, botSku — ценовая категория (F0 — бесплатно), newAppServicePlanName — имя нового плана сервиса приложения, newWebAppName — имя сервиса приложения, groupName — имя новой группы ресурсов, groupLocation и newAppServicePlanLocation — расположения группы ресурсов и плана сервиса приложения соответственно.

4. «az bot prepare-deploy --lang Csharp --code-dir "." --proj-file-path "MovieRecommenderBot.csproj"» — подготовка файлов C# проекта к развертыванию (команда выполняется из папки проекта), в итоге создается файл .deployment в этой папке.

В результате на портале Azure был создан сервис приложения. Далее можно развертывать бот-приложение прямо из Visual Studio, как любое другое веб-приложение. Для этого надо кликнуть правой кнопкой мыши по проекту, нажать кнопку Publish, далее New, выбрать из списка Azure, затем — Azure App Service (Windows). Из портала Azure загрузятся группы ресурсов и сервисы приложений в них. Необходимо выбрать вновь созданные rg_movie_recommender_bot и movie-recommender-bot и завершить процесс. Откроется окно с настройками развертывания, в нем надо поменять режим развертывания на Self-contained, поскольку бот-приложение использует сторонние библиотеки, в том числе ранее обученную модель системы рекомендаций. Среда выполнения устанавливается в win-x86. После этого остается нажать кнопку Publish и дождаться завершения процесса развертывания. В результате бот-приложение доступно на портале Azure и может быть там протестировано с помощью встроенного канала Web Chat.

6.8 Подключение бота к каналу

Канал — это соединение между ботом и коммуникационными приложениями, такими как Web Chat, Cortana, Skype, Telegram и другими [25]. Бот предварительно надо настроить для подключения к каналам, из которых он должен быть доступен. Сервис Bot Framework, настроенный через портал Azure, подключает бота к этим каналам и служит связующим звеном между ботом и пользователем. Боту ничего не нужно знать о каналах, к которым он подключается, всю работу выполняет данный сервис, в частности он занимается адаптацией сообщений, отправляемых ботом, под конкретный канал, что включает в себя преобразование сообщения из схемы бота в схему канала. При этом если канал не поддерживает все характеристики и/или функционал исходной схемы бота, сервис пытается преобразовать сообщение в поддерживаемый каналом формат. Например, если бот отправляет сообщение, содержащее форматированную карточку с изображением и кнопками, в канал Email, сервис отправит письмо с отдельно прикрепленным изображением и изменит кнопки на ссылки в тексте письма.

Так как бот с системой рекомендаций содержит много карточек различного вида, в том числе с кнопками и картинками, а лишь немногие из каналов поддерживают все эти компоненты без нежелательных видоизменений, то для подключения был выбран канал Web Chat.

Чтобы настроить бот для подключения к каналу, после входа на портал Azure надо выполнить следующие действия:

1. Выбрать бота, который будет подключаться к каналу (ранее созданный MovieRecommenderBot).
2. В разделе Settings нажать Channels.
3. Нажать на кнопку со значком нужного канала.

Web Chat канал автоматически настраивается при создании бота, поэтому он уже есть в списке каналов на вкладке Channels и перечисленные выше шаги для него выполнять не надо. Также бот можно сразу протестировать с помощью этого канала, нажав на Test in Web Chat в разделе Settings.

Процесс настройки отличается для каждого канала. Web Chat канал включает в себя элемент управления веб-чатом, который предоставляет пользователям возможность взаимодействовать с ботом непосредственно на веб-странице. На портале Bot Framework есть все необходимые данные для вставки этого элемента на веб-страницу, включая секретный ключ бота. Чтобы его получить необходимо:

1. Находясь на вкладке Channels, нажать на кнопку Edit для канала Web Chat.
2. В разделе Secret keys выбрать Show для первого ключа.

3. Скопировать отобразившийся секретный ключ и код из раздела Embed cod».

4. Нажать кнопку Done.

Скопированный код из раздела Embed code представляет собой элемент iframe (контейнер для загрузки различных документов) языка HTML, и пока что он не ведет себя как обычный виджет чата, который находится в нижней части окна браузера, всплывает вверх и сворачивается вниз при нажатии.

Для создания виджета чата, в документе WebChat.html (приложение Е) вышеупомянутый iframe «оборачивается» в элемент div (блочный элемент для выделения фрагмента документа), при этом div фиксируется к нижней части окна браузера с помощью стилей CSS, а высота устанавливается небольшой, чтобы изначально не было видно окна чата (код Styles.css представлен в приложении Ж). В iframe в значении атрибута src вставляется ранее скопированный секретный ключ бота. Внутри созданного элемента div создается еще один (с такой же высотой) — он и есть та «полоска» внизу экрана с надписью «Movie Recommender» которая видна пользователю при загрузке веб-сайта (рисунок 6.17).

Рисунок 6.17 — Скриншот виджета чата в свернутом состоянии на странице WebChat.html, открытой в браузере

К данной «полоске», с помощью JavaScript кода, добавляется обработчик события, возникающего при нажатии на него мышкой. В обработчике проверяется высота основного div элемента (в котором находится iframe): если высота равна высоте «полоски», то она увеличивается, таким образом весь чат становится виден (рисунок 6.18), если же наоборот — чат сворачивается вниз.

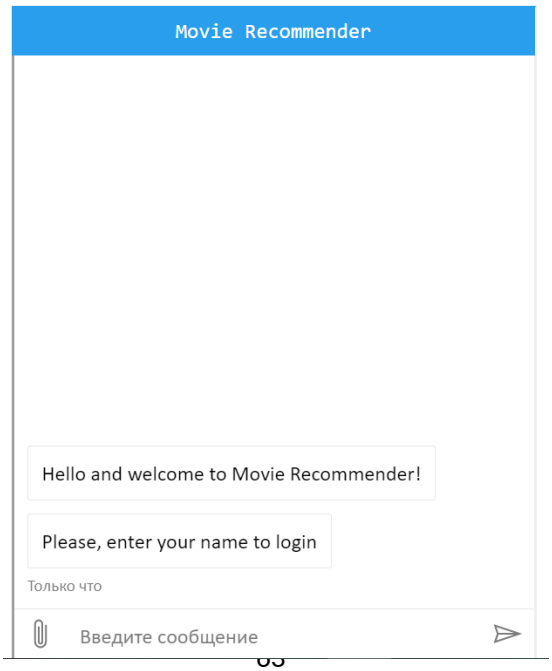


Рисунок 6.18 — Скриншот виджета чата в развернутом состоянии на странице WebChat.html, открытой в браузере

Чтобы виджет появлялся плавно, с помощью CSS для div элемента устанавливается свойство transition для постепенного изменения высоты в течение 0,4 секунды, причем дополнительно задается параметр linear, который обеспечивает эффект перехода с одинаковой скоростью от начала до конца.

Созданный таким образом виджет может быть использован на любом веб-сайте, например, для просмотра фильмов, что позволяет пользователю избежать необходимости листать каталоги и тратить время на сложный выбор: достаточно задать вопрос на естественном языке, и система рекомендаций подберет наиболее подходящие ему кинокартины.

6.9 Добавление телеметрии в бот

Сервис Azure Application Insights отображает данные о приложении в соответствующем ресурсе на портале Azure. Данные телеметрии включают полезные сведения о боте, например, об отправляемых и получаемых запросах, а также помогают обнаружить нежелательное поведение и предоставляют сведения о производительности и использовании на различных каналах.

6.9.1 Подписка на сервис Application Insights на портале Azure

Чтобы добавить телеметрию к боту необходима подписка Azure и ресурс Application Insights, созданный для бота. Из этого ресурса нужно получить ключ инструментирования для настройки бота. Для этого на портале Azure надо выполнить следующие шаги:

1. Нажать на кнопку Monitor.
2. Зайти в раздел Applications, нажать на кнопку Add, ввести необходимые сведения для создания ресурса.
3. Нажать на кнопку Review + create и в списке ресурсов выбрать только что созданный.
4. Далее нажать на созданный ресурс и перейти в раздел Overview.
5. Развернуть блок Essentials, скопировать значение Instrumentation Key и вставить его в файл appsettings.json проекта бот-приложения.

6.9.2 Использование сервиса Application Insights в бот-приложении

Для начала работы с Application Insights в бот-приложении необходимо установить Microsoft.Bot.Builder.Integration.ApplicationInsights.Core — NuGet

пакет. Далее в методе ConfigureServices класса Startup добавляются сервисы для работы с телеметрией:

- IBotTelemetryClient (реализация BotTelemetryClient) — клиент телеметрии;
- ITelemetryInitializer (реализация OperationCorrelationTelemetryInitializer) — инициализатор, который задает контекст корреляции для всех элементов телеметрии;
- ITelemetryInitializer (реализация TelemetryBotIdInitializer) — инициализатор, устанавливающий идентификатор пользователя и идентификатор сеанса;
- TelemetryInitializerMiddleware — промежуточное программное обеспечение для инициализации сбора телеметрии;
- TelemetryLoggerMiddleware — промежуточное программное обеспечение, которое используется инициализатором, и нужно для отслеживания событий разговора бота с пользователем.

В коллекцию промежуточного программного обеспечения адаптера бота AdapterWithErrorHandler (приложение Г) добавляется класс TelemetryInitializerMiddleware.

По умолчанию, при ведении журнала действий некоторые свойства входящих и исходящих действий исключаются из журнала, поскольку они могут содержать личную информацию, например, имя пользователя и текст активности. Однако эти свойства можно включить в журнал, передав в конструктор TelemetryLoggerMiddleware значение true для параметра logPersonalInformation. Это может помочь получить больше информации для выявления причин возникших ошибок.

6.9.3 Анализ данных телеметрии бот-приложения на портале Azure

В результате предыдущих шагов приложение регистрирует данные телеметрии в Application Insights, причем пользователь не видит никаких изменений в поведении бота. Существует несколько способов просмотра телеметрических данных, собранных с помощью Application Insights, один из них — это поиск. Поиск — функция Application Insights, которая используется для изучения отдельных элементов телеметрии, например, исключений и веб-запросов, а также для просмотра трассировки журналов [26]. Диагностический поиск можно открыть в разделе Overview ресурса на портале Azure, нажав на вкладку Search, или в разделе Investigate, пункт Transaction Search. На рисунке 6.19 — диаграмма с количеством событий во времени, которая отображается после открытия диагностического поиска.

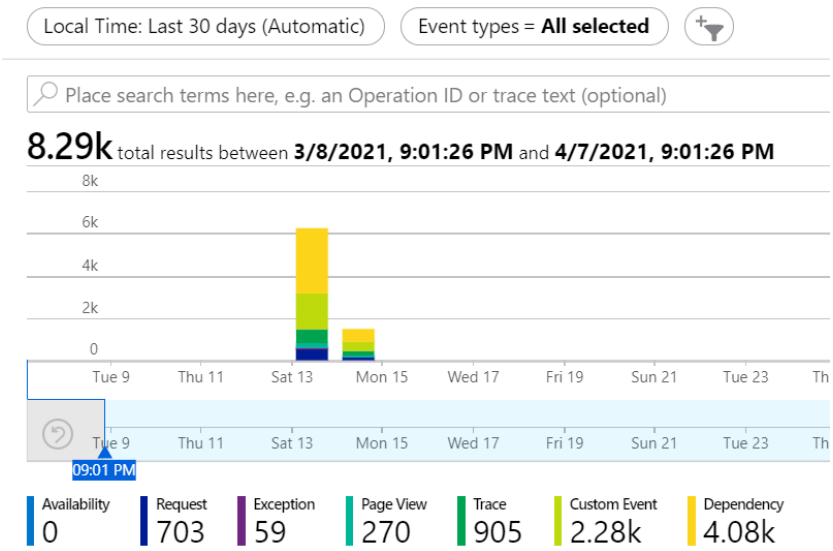


Рисунок 6.19 — Функция поиска Application Insights на портале Azure

Также функция поиска позволяет выбрать тип событий, информация о которых будет отображаться. Для этого есть раскрывающийся список «Event types». Например, часть вывода событий типа «Request», то есть только HTTP-запросов, представлен на рисунке 6.20.

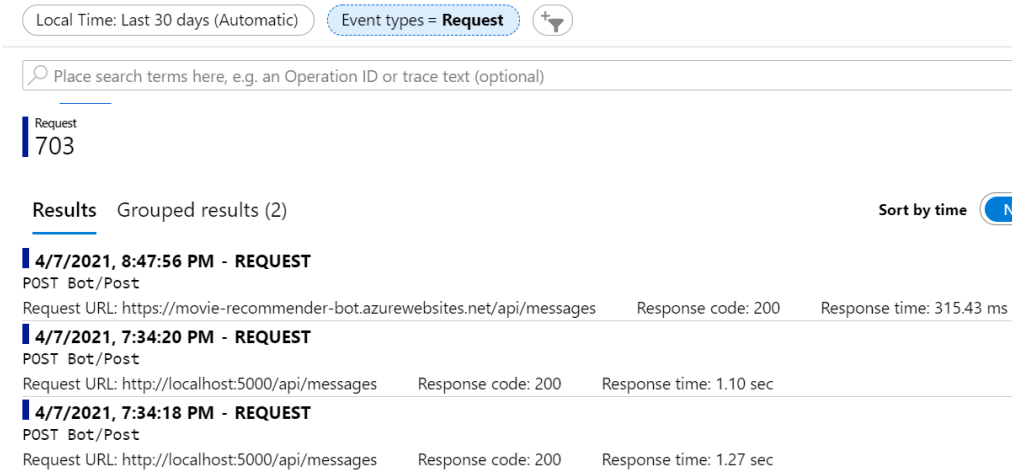


Рисунок 6.20 — Отображение событий типа «Request» на портале Azure

Для любого элемента телеметрии можно просмотреть ключевые поля и связанные с ним элементы, просто выбрав его из списка событий. На рисунке 6.21 отображена часть информации об одном из пользовательских событий: тип действия, канал, с которого пришло данное событие (Web Chat), текст сообщения, локализация и другие. Каждый тип событий имеет как общие для всех характеристики, так и свои собственные.

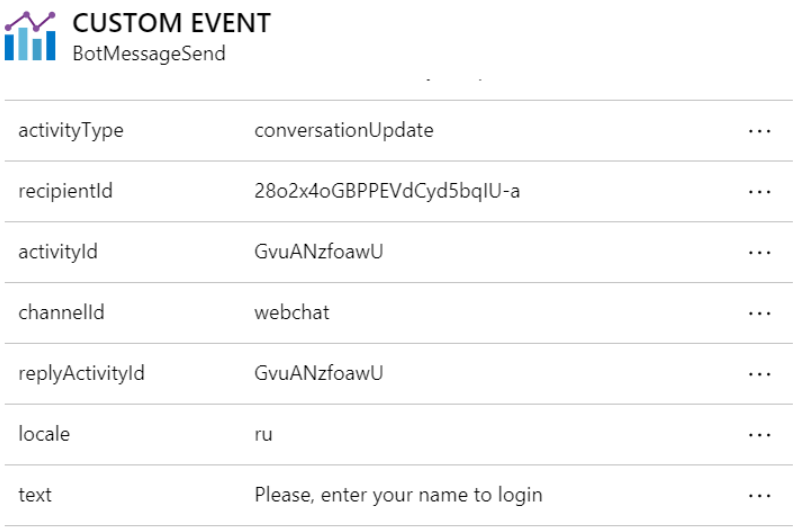


Рисунок 6.21 — Детальная информация о пользовательском событии на портале Azure

Также Application Insights позволяет просмотреть схему приложения — для этого нужно в разделе «Investigate» выбрать «Application map». Схема приложения позволяет выявлять сбои в приложении и «слабые места» производительности во всех компонентах распределенного приложения. Также одной из ее главных задач является получение возможности визуализировать сложные топологии с сотнями компонентов.

Схема приложения находит компоненты, отслеживая HTTP-вызовы их зависимостей. Таким образом каждый узел на схеме представляет компонент приложения или его зависимости, причем по нему можно нажать, чтобы получить более детальные диагностические сведения, например, о событиях Application Insights, а также чтобы просмотреть данные аналитики и затем перейти к рассмотрению производительности и сбоев.

На рисунке 6.22 представлена схема бот-приложения с системой рекомендаций. На ней отображен компонент сервиса Application Insights, компонент localhost — бот-приложение, запущенное локально, компонент бота, развернутого в Azure, а также их зависимости: хранилище таблиц Azure Table storage, MovieLens API, login — для подключения к сервисам Bot Framework и Cognitive Services, LUIS, а также Web Chat — канал, к которому подключен бот.

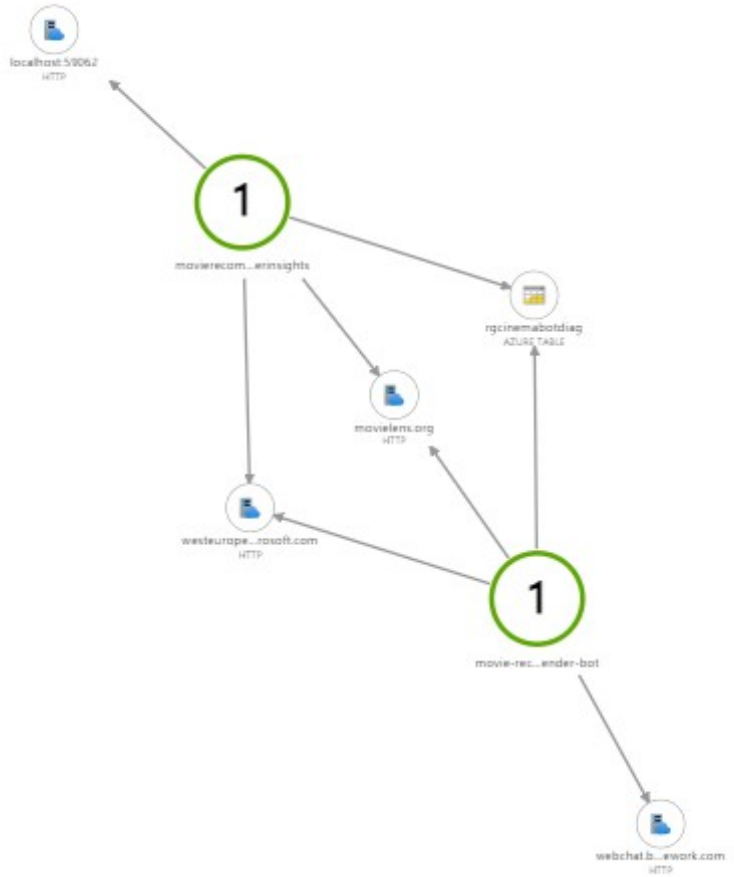


Рисунок 6.22 — Схема бот-приложения на портале Azure

ВЫВОДЫ

1. Рассмотрена работа с библиотекой диалогов.
2. Исследована работа с форматированными карточками в сообщениях.
3. Определено, как создавать LUIS приложение на портале https://www.luis.ai и использовать его в бот-приложении.
4. Разработано бот-приложение для получения рекомендаций фильмов.
5. Бот-приложение интегрировано с ранее разработанной системой рекомендаций на основе машинного обучения.
6. Бот-приложение развернуто на портале Azure.
7. Бот подключен к каналу Web Chat.
8. Создан виджет чата для использования на веб-сайтах.
9. Изучены данные телеметрии бот-приложения, предоставляемые сервисом Azure Application Insights.

ЗАКЛЮЧЕНИЕ

Системы рекомендаций получили широкое распространение во многих онлайн-сферах. Сегодня они используются в различных областях: электронная коммерция и сфера услуг, информационные порталы, стриминговые сервисы, социальные сети и так далее. Системы рекомендаций могут является источником огромного дохода, в случае если они эффективны, а также быть способом значительно выделиться среди конкурентов.

Бот-приложения становятся все более распространенными, поскольку удобны в использовании, а взаимодействие с ними напоминает общение с человеком. Боты предоставляют помощь пользователю в режиме реального времени, ускоряя путь к достижению цели.

Бот-приложение с системой рекомендаций может быть интегрировано в виде виджета чата на веб-сайт для просмотра фильмов, что позволит пользователю избежать необходимости листать каталоги и тратить время на сложный выбор: достаточно задать вопрос на естественном языке, и система рекомендаций подберет наиболее подходящие ему кинокартины.

Результатом данной работы является модель машинного обучения для системы рекомендации фильмов пользователям на основе их предпочтений и бот-приложение, умеющее распознавать естественный язык пользователя и интегрированное с системой рекомендаций фильмов.

Адаптированное под систему рекомендаций тренингов приложение в настоящий момент используется в системе тренингов ИООО «Эксадел» для рекомендации курсов и вебинаров сотрудникам в соответствии с их предпочтениями. Данная разработка позволяет пользователю, использующему систему тренингов, в режиме реального времени получить помощь в выборе курсов или вебинаров, задав вопрос на естественном языке.

В ходе выполнения работы были рассмотрены и решены следующие задачи:

1. Проведен обзор существующих систем рекомендаций, составлены требования к разрабатываемому приложению.

2. Рассмотрены типы систем рекомендаций: фильтрация на основе содержания, коллаборативная фильтрация и гибридная фильтрация, а также алгоритмы коллаборативной фильтрации, основанные на модели: матричное разложение, машины факторизации и машины факторизации с учетом специфики поля.

3. Для создания системы рекомендаций фильмов выбран алгоритм машин факторизации с учетом специфики поля, который предназначен для решения задач бинарной классификации. Рассмотрены метрики качества для класса задач бинарной классификации.

4. Рассмотрены возможности платформы ML.NET для решения различных типов задач машинного обучения, в частности для построения систем рекомендаций. Изучен процесс машинного обучения в ML.NET: загрузка и подготовка данных, деление их на тренировочный и тестовый наборы, создание конвейера, обучение и оценка полученной модели.

5. Определено, как использовать модель ML.NET в других приложениях или сервисах: ее нужно сохранить, а затем загрузить в этом приложении — тогда можно делать необходимые прогнозы, а также определено, как использовать модель на различных устройствах, операционных системах, платформах: для этого применяется экспорт модели в формат ONNX.

6. Изучен набор данных о фильмах MovieLens. Загружены и обработаны данные о рейтингах и фильмах из набора данных MovieLens, осуществлена их полготовка для построения модели в ML.NET.

7. Создана и обучена модель для системы рекомендаций фильмов. Оценена эффективность обученной модели.

8. Изучена и решена проблема холодного старта для пользователя для системы рекомендаций, в том числе решена задача нахождения пользователя с похожими предпочтениями с применением косинусного сходства.

9. Изучены и применены возможности хранилища таблиц Azure Table storage для хранения данных о рейтингах, выставленных пользователями.

10. Спроектировано пользовательское приложение.

11. Рассмотрены основные типы бот-приложений. Изучены возможности Azure Bot Service и Bot Framework — предоставление средств для сборки, тестирования, развертывания и управления интеллектуальными ботами в единой среде.

12. Определены возможности Azure Cloud Services для разработки бот-приложения — предоставление сервисов, которые могут быть использованы для расширения его функционала: Azure AD B2C, Azure Monitor, Azure Cognitive Services, включая LUIS.

13. Спроектировано бот-приложение.

14. Рассмотрена работа с библиотекой диалогов в Azure Bot Framework, в разработке применены диалоги ввода и каскадные диалоги. Изучена работа с форматированными карточками в сообщениях для обогащения пользовательского опыта, в разработке применены адаптивные карточки.

15. На портале lius.ai было создано LUIS приложение для распознавания естественного языка пользователя, и затем использовано в бот-приложении.

16. Организовано получение изображений постеров фильмов через MovieLens API.

17. Разработано бот-приложение для получения рекомендаций фильмов и интегрировано с ранее разработанной системой рекомендаций на основе машинного обучения.

18. Бот-приложение развернуто на портале Azure и подключено к каналу Web Chat, создан виджет чата для использования на веб-сайтах.

19. Включено ведение журнала телеметрии бот-приложения с помощью сервиса Azure Application Insights — компонента Azure Monitor.

Для проектирования диаграмм деятельности процесса обучения и использования модели машинного обучения, схемы построения модели в ML.NET, последовательности взаимодействий системы рекомендаций и пользователя, процесса работы с новым пользователем, а также архитектуры бот-приложения использовался онлайн-сервис для построения диаграмм draw.io.

Для отображения набора данных в формате CSV использовался DB Browser for SQLite — инструмент для работы с файлами баз данных, совместимых с SQLite.

Для разработки приложения с обучением модели ML.NET и создания бот-приложения использовалась среда разработки Visual Studio 2019.

Для тестирования бота использовалось приложение Bot Framework Emulator и канал Web Chat на портале Azure.

Результаты работы представлены на 77-ой и 78-й Научной конференции студентов и аспирантов БГУ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Content-Boosted Collaborative Filtering for Improved Recommendations [Electronic resource] / Prem Melville, Raymond J. Mooney, Ramadass Nagarajan. — Mode of access: https://www.aaai.org/Papers/AAAI/2002/AAAI02-029.pdf. — Date of access: 24.09.2020.

2. Machine Learning [Electronic resource]. — Mode of access: https://reposhub.com/dotnet/machine-learning-and-data-science/dotnet-machinelearning.html. — Date of access: 25.09.2020.

3. Development of recommender systems using ML.NET [Electronic resource] / Bahrudin I Hrnjica, Denis Mušić, Selver Softic. — Mode of access: https://www.researchgate.net/publication/340389519_DEVELOPMENT_OF_RECOMMENDER_SYSTEMS_USING_MLNET. — Date of access: 24.09.2020.

4. Introduction to recommender systems [Electronic resource]. — Mode of access: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada. — Date of access: 28.09.2020.

5. An Intuitive Explanation of Field Aware Factorization Machines [Electronic resource]. — Mode of access: https://towardsdatascience.com/an-intuitive-explanation-of-field-aware-factorization-machines-a8fee92ce29f. — Date of access: 28.09.2020.

6. E-Commerce Item Recommendation Based on Field-aware Factorization Machine [Electronic resource] / Peng Yan, Xiaocong Zhou, Yitao Duan. — Mode of access: https://www.researchgate.net/publication/282846395_E-Commerce_Item_Recommendation_Based_on_Field-aware_Factorization_Machine. — Date of access: 04.10.2020.

7. The ultimate guide to binary classification metrics [Electronic resource]. — Mode of access: https://towardsdatascience.com/the-ultimate-guide-to-binary-classification-metrics-c25c3627dd0a — Date of access: 02.11.2020.

8. What is ML.NET and how does it work? [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/dotnet/machine-learning/how-does-mldotnet-work. — Date of access: 25.09.2020.

9. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems [Electronic resource]. — Mode of access: http://download.tensorflow.org/paper/whitepaper2015.pdf. — Date of access: 18.11.2020.

10. Announcing ML.NET 0.3 [Electronic resource]. — Mode of access: https://devblogs.microsoft.com/dotnet/announcing-ml-net-0-3/#onnx-section. — Date of access: 18.11.2020.

11. ONNX Runtime for inferencing machine learning models [Electronic resource]. — Mode of access: https://azure.microsoft.com/en-us/blog/onnx-runtime-

for-inferencing-machine-learning-models-now-in-preview/. — Date of access: 22.11.2020.

12. Unified Modeling Language (UML) | Activity Diagrams [Electronic resource]. — Mode of access: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/.— Date of access: 16.11.2020.

13. MovieLens Latest Full [Electronic resource]. — Mode of access: https://www.kaggle.com/grouplens/movielens-latest-full. — Date of access: 14.10.2020.

14. Prepare data for building a model [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/dotnet/machine-learning/how-to-guides/prepare-data-ml-net. — Date of access: 16.10.2020.

15. Personality-Based Active Learning for Collaborative Filtering Recommender Systems [Electronic resource] / Mehdi Elahi, Matthias Braunhofer, Francesco Ricci, Marko Tkalcic. — Mode of access: http://www.cp.jku.at/research/papers/elahi2013aixia.pdf. — Date of access: 06.02.2021.

16. The Cold Start Problem for Recommender Systems [Electronic resource]. — Mode of access: https://yuspify.com/blog/cold-start-problem-recommender-systems/. — Date of access: 06.02.2021.

17. MachineX: Cosine Similarity for Item-Based Collaborative Filtering [Electronic resource]. — Mode of access: https://dzone.com/articles/machinex-cosine-similarity-for-item-based-collabor. — Date of access: 06.02.2021.

18. What is Azure Table storage? [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/azure/storage/tables/table-storage-overview. — Date of access: 07.02.2021.

19. 27 Incredible Chatbot Statistics [Electronic resource]. — Mode of access: https://www.netomi.com/chatbot-statistics. — Date of access: 19.03.2021.

20. Get started guide for developers on Azure [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/azure/guides/developer/azure-developer-guide#what-is-azure. — Date of access: 19.03.2021.

21. About Azure Bot Service [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-4.0. — Date of access: 19.03.2021.

22. Azure Monitor overview [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/azure/azure-monitor/overview. — Date of access: 23.03.2021.

23. What are Azure Cognitive Services? [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/azure/cognitive-services/welcome. — Date of access: 25.03.2021.

24. What is Language Understanding (LUIS)? [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/what-is-luis. — Date of access: 25.03.2021.

25. Connect a bot to channels [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-manage-channels?view=azure-bot-service-4.0. — Date of access: 06.04.2021.

26. Using Search in Application Insights [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/azure/azure-monitor/app/diagnostic-search. — Date of access: 07.04.2021.

27.

ПРИЛОЖЕНИЕ А

Исходный код DataProcessor.cs (функции LoadMovies, LoadRatings и LoadData)

```
private List<Movie> LoadMovies(string filePath, bool hasHeader)
    {
        var data = File.ReadAllLines(filePath);
        List<Movie> movies;
        if (hasHeader)
        {
            movies = new List<Movie>(data.Length - 1);
            data = data.RemoveAt(0);
        }
        else
        {
            movies = new List<Movie>(data.Length);
        }
        foreach (var line in data)
        {
            var split = line.Split("");
            if (split.Length == 1)
            {
                split = split[0].Split(',');
            }
            else
            {
                if (split.Length > 3)
                {
                    split[1] = line.Substring(split[0].Length + 1,
                        line.Length - split[0].Length - split[^1].Length - 1);
                }
                split[0] = split[0].Substring(0, split[0].Length - 1);
```

```
            split[2] = split[^1].Substring(1);
        }
        movies.Add(new Movie { Id = float.Parse(split[0]), Title = split[1], Genres =
split[2].Split('|') });
    }
    return movies;
}
private void LoadRatings()
{
    var dataPath = Path.Combine(Environment.CurrentDirectory, "Data", "ratings.csv");
    var ratingsData = _mlContext.Data.LoadFromTextFile<UserRating>(dataPath, hasHeader:
true, separatorChar: ',');
    _ratings = _mlContext.Data.CreateEnumerable<UserRating>(ratingsData, false).ToList();
}

public (IDataView training, IDataView test) LoadData()
{
    var movieRatings = Ratings.Join(Movies, rating => rating.MovieId, movie => movie.Id,
     (rating, movie) => movie.Genres.Select(genre => new MovieRating
     {
        UserId = rating.UserId,
        MovieId = movie.Id,
        MovieTitle = movie.Title,
        MovieGenre = genre,
        Rating = rating.RatingValue
     })).SelectMany(movieRating => movieRating);
    var data = _mlContext.Data.LoadFromEnumerable(movieRatings);
    var split = _mlContext.Data.TrainTestSplit(data, 0.2);
    var training = split.TrainSet;
    training = _mlContext.Data.Cache(training);
    return (training, split.TestSet);
}
```

ПРИЛОЖЕНИЕ Б

Исходный код UserStorage.cs

```
using System.Collections.Generic;
using Microsoft.Azure.Cosmos.Table;
using MovieRecommender.DataModels;
using Newtonsoft.Json;

namespace MovieRecommender
{
   public class UserStorage
   {
      private const string ConnectionString = "<secret>";
      private const string TableName = "Ratings";
      private const string PartitionKey = "Users";
      private readonly CloudTable _tableClient;

      public UserStorage()
      {
         var storageAccount = CloudStorageAccount.Parse(ConnectionString);
         CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new
TableClientConfiguration());
         _tableClient = tableClient.GetTableReference(TableName);
         _tableClient.CreateIfNotExists();
      }

      public void Create(string userId, IList<Rating> ratings)
      {
         var entity = new UserRatingEntity(userId) { Ratings = ratings };
         _tableClient.Execute(TableOperation.InsertOrReplace(entity));
      }

      public void Update(string userId, IList<Rating> ratings)
      {
         var retrieveOperation =
            TableOperation.Retrieve<UserRatingEntity>(PartitionKey, userId);

         var result = _tableClient.Execute(retrieveOperation);
         if (result.Result is UserRatingEntity entity)
```

```
            {
                var currentRatings = (List<Rating>)entity.Ratings;
                currentRatings.AddRange(ratings);
                entity.Ratings = currentRatings;
                _tableClient.Execute(TableOperation.Replace(entity));
            }
        }

        public bool UserExists(string userId)
        {
            var retrieveOperation =
                TableOperation.Retrieve<UserRatingEntity>(PartitionKey, userId);
            var result = _tableClient.Execute(retrieveOperation);
            return result.Result is UserRatingEntity;
        }

        public IEnumerable<Rating> GetUserRatings(string userId)
        {
            var retrieveOperation =
                TableOperation.Retrieve<UserRatingEntity>(PartitionKey, userId);
            var result = _tableClient.Execute(retrieveOperation);
            return (result.Result as UserRatingEntity)?.Ratings;
        }

        private class UserRatingEntity : TableEntity
        {
            public UserRatingEntity(){}
            public UserRatingEntity(string userId)
                :base(UserStorage.PartitionKey, userId) {}
            public string RatingsInternal { get; set; }

            [IgnoreProperty]
            public IList<Rating> Ratings
            {
                get => JsonConvert.DeserializeObject<IList<Rating>>(RatingsInternal);
                set => RatingsInternal = JsonConvert.SerializeObject(value);
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ В

Исходный код Predictor.cs

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Accord.Math.Distances;
using Microsoft.ML;
using MovieRecommender.DataModels;

namespace MovieRecommender.Services
{
   public class Predictor
   {
      private readonly string _userRatingsPath;
      private Dictionary<int, double[]> _userRatings;
      private readonly DataProcessor _dataProcessor;
      private readonly MLContext _mlContext;
      private readonly ITransformer _model;

      public Predictor(MLContext mlContext, ITransformer model, DataProcessor dataProcessor)
      {
         _mlContext = mlContext;
         _model = model;
         _dataProcessor = dataProcessor;
         _userRatingsPath = _dataProcessor.UserRatingsPath;
      }

      public Dictionary<int, double[]> UserRatings
      {
         get
         {
            if (_userRatings == null)
            {
               _userRatings = new Dictionary<int, double[]>();
               var data = File.ReadAllLines(_userRatingsPath);
               foreach (var str in data)
               {
                  var split = str.Split();
```

```
            _userRatings[int.Parse(split[0])] = split.Skip(1).Select(double.Parse).ToArray();
        }
      }
      return _userRatings;
   }
}

private int FindSimilarUserId(double[] currentUserRatings)
 {
    double maxSimilarity = 0.0;
    int userId = 0;
    var cosine = new Cosine();
    foreach (var userRating in UserRatings)
    {
       var similarity = cosine.Similarity(userRating.Value, currentUserRatings);
       if (similarity > maxSimilarity)
       {
          maxSimilarity = similarity;
          userId = userRating.Key;
       }
    }
    return userId;
 }

 public IEnumerable<Recommendation> PredictTop5(IEnumerable<Rating> ratings)
 {
    ratings = ratings.ToList();
    var similarUserId = FindSimilarUserId(_dataProcessor.GetUserAllMoviesRatings(ratings));
    var predictionEngine = _mlContext.Model
       .CreatePredictionEngine<MovieRating, MovieRatingPrediction>(_model);
    var watchedMoviesIds = ratings.Select(r => r.MovieId).ToList();
    return _dataProcessor.Movies.Select(movie => new Recommendation
       {
          Movie = movie,
          Prediction = predictionEngine.Predict(
             new MovieRating
             {
                UserId = similarUserId,
                MovieId = movie.Id
             })
```

```
            })
            .Where(moviePrediction => moviePrediction.Prediction.PredictedLabel
                                && !watchedMoviesIds.Contains((int)moviePrediction.Movie.Id))
            .OrderByDescending(moviePrediction => moviePrediction.Prediction.Score)
            .Take(5);
        }
    }
}
```

ПРИЛОЖЕНИЕ Г

Исходный код AdapterWithErrorHandler.cs

```
using Microsoft.Bot.Builder.Integration.ApplicationInsights.Core;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace MovieRecommenderBot
{
   public class AdapterWithErrorHandler : BotFrameworkHttpAdapter
   {
      public AdapterWithErrorHandler(IConfiguration configuration,
         ILogger<BotFrameworkHttpAdapter> logger,
         TelemetryInitializerMiddleware telemetryInitializerMiddleware)
         : base(configuration, logger)
      {
         Use(telemetryInitializerMiddleware);
         OnTurnError = async (turnContext, exception) =>
         {
            logger.LogError($"Exception caught : {exception.Message}");
            await turnContext.SendActivityAsync("Sorry, it looks like something went wrong.");
         };
      }
   }
}
```

ПРИЛОЖЕНИЕ Д

Исходный код Bot.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Schema;

namespace MovieRecommenderBot.Bots
{
   public class Bot<T> : ActivityHandler where T : Dialog
   {
      protected readonly Dialog Dialog;
      protected readonly BotState ConversationState;
      protected readonly BotState UserState;
      private readonly IStatePropertyAccessor<string> _userNameStateProperty;
      private readonly IStatePropertyAccessor<bool> _needToAskStateProperty;

      public Bot(ConversationState conversationState, UserState userState, T dialog)
      {
         ConversationState = conversationState;
         UserState = userState;
         Dialog = dialog;
         _userNameStateProperty = userState.CreateProperty<string>("UserName");
         _needToAskStateProperty = conversationState.CreateProperty<bool>("NeedToAsk");
      }

      protected override async Task OnMembersAddedAsync(
         IList<ChannelAccount> membersAdded,
         ITurnContext<IConversationUpdateActivity> turnContext,
         CancellationToken cancellationToken)
      {
         if (turnContext.Activity.ChannelId == "webchat")
         {
            return;
         }
```

```
    foreach (var member in membersAdded)
    {
        if (member.Id != turnContext.Activity.Recipient.Id)
        {
            await SendWelcomeMessageAsync(turnContext, cancellationToken);
        }
    }
}

protected override async Task OnConversationUpdateActivityAsync(
    ITurnContext<IConversationUpdateActivity> turnContext,
    CancellationToken cancellationToken)
{
    await base.OnConversationUpdateActivityAsync(turnContext, cancellationToken);

    if (turnContext.Activity.ChannelId == "webchat"
        && turnContext.Activity.MembersAdded?
            .FirstOrDefault(account => account?.Name == "MovieRecommenderBot")!= null)
    {
        await SendWelcomeMessageAsync(turnContext, cancellationToken);
    }
}

protected override async Task OnMessageActivityAsync(
    ITurnContext<IMessageActivity> turnContext,
    CancellationToken cancellationToken)
{
    var message = turnContext.Activity.Text;

    if (await _needToAskStateProperty.GetAsync(turnContext, () => true, cancellationToken))
    {
        await _userNameStateProperty.SetAsync(turnContext, message, cancellationToken);

        await _needToAskStateProperty.SetAsync(turnContext, false, cancellationToken);

        var reply = MessageFactory.Text($"Hi, {message}!");
        await turnContext.SendActivityAsync(reply, cancellationToken);
    }
    await Dialog.RunAsync(turnContext,
```

```
            ConversationState.CreateProperty<DialogState>(nameof(DialogState)),
cancellationToken);
        }

        public override async Task OnTurnAsync(ITurnContext turnContext,
            CancellationToken cancellationToken = default)
        {
            await base.OnTurnAsync(turnContext, cancellationToken);
            await ConversationState.SaveChangesAsync(turnContext, false, cancellationToken);
            await UserState.SaveChangesAsync(turnContext, false, cancellationToken);
        }

        private async Task SendWelcomeMessageAsync(
            ITurnContext turnContext,
            CancellationToken cancellationToken)
        {
            await turnContext.SendActivityAsync(
                MessageFactory.Text("Hello and welcome to Movie Recommender!"),
                cancellationToken);

            await _needToAskStateProperty.SetAsync(turnContext, true, cancellationToken);

            var reply = MessageFactory.Text("Please, enter your name to login");

            await turnContext.SendActivityAsync(reply, cancellationToken);
        }
    }
}
```

ПРИЛОЖЕНИЕ Е

Исходный код WebChat.html

```
<!DOCTYPE html>
<html>
<head>
<link href="styles.css" rel="stylesheet" />
</head>
<body>
<script>
(function () {
    var div = document.createElement("div");
    document.getElementsByTagName('body')[0].appendChild(div);
    div.outerHTML = "<div id='botDiv'>"+
          "<div id='botTitleBar'>" +
          "<p>Movie Recommender</p></div>" +
          "<iframe id='chat' src='https://webchat.botframework.com/embed/MovieRecommenderBot?
s='></iframe></div>";

    document.getElementById('botTitleBar').addEventListener('click', function (e) {
                var botDiv = document.getElementById('botDiv');
                botDiv.style.height = botDiv.style.height == '500px' ? '38px' : '500px';
    });
}());
</script>
</body>
</html>
```

ПРИЛОЖЕНИЕ Ж

Исходный код Styles.css

```
#botDiv {
	background-color: #FFF;
	transition: height 0.4s linear;
	height: 38px;
	bottom: 0;
	position: fixed;
}
#botTitleBar {
	height: 38px;
	width: 400px;
	position:fixed;
	cursor: pointer;
	background-color: #2499EC;
}
p{
	color: #FFF;
	text-align:center;
	margin-top: 10px;
	font-family: consolas;
}

#chat{
	width: 397px;
	height: 500px;
}
```