Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Работа допущена к защите

Директор ВШ ПИ
П.Д. Дробинцев
«_____»_____2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: Создание облаков тегов для фильмов

направление 09.03.04 «Программная инженерия»

образовательная программа

09.03.04_01 «Технология разработки и сопровождения качественного программного продукта»

Выполнил
студент гр. 43504/4 (подпись) С.Р. Астафьев

Руководитель

доцент, к.т.н. (подпись) В.В. Шаляпин

Санкт – Петербург
2019

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа программной инженерии

УТВЕРЖДАЮ

Директор ВШ ПИ
П.Д. Дробинцев
«_____»_____ г.

ЗАДАНИЕ

к работе на соискание степени бакалавра

Студенту группы Астафьеву Сергею Романовичу, 43504/4

1. Тема проекта (работы) Создание облаков тегов

(утверждена распоряжением по ИКНТ от _____№_____)

2. Срок сдачи студентом оконченного проекта (работы)

3. Исходные данные к проекту (работе)

Сервисы Amazon_____

База данных фильмов_____

4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

Введение_____

Анализ предметной области_____

Проектирование архитектуры системы_____

Программная реализация_____

Анализ результатов_____

Заключение_____

5. Перечень графического материала (с точным указанием обязательных чертежей)

_____

6. Консультанты по проекту (с указанием относящихся к ним разделов проекта, работы)

_____

7. Дача выдачи задания

Руководитель Шаляпин Владимир Валентинович_____

Задание принял к исполнению

_____

(подпись студента)

Реферат

С.52. Рис.4. Табл.2. 1 приложение

СЕРВИСЫ, ФИЛЬМЫ, ТЕГИ, ОБЛАКА ТЕГОВ, AMAZON, AMAZON WEB SERVICES

Целью работы является разработка приложения для генерации облаков тегов для фильмов на основе трейлеров.

При создании приложения были использованы сервисы Amazon Web Services, среда IPython с веб-оболочкой Jupyter Notebook.

Были изучены способы использования сервисов от Amazon.

Результатом данной работы является приложение для создания облаков тегов на основе трейлеров для фильмов.

Содержание

Введение

Актуальность

В настоящее время интернет предлагает огромное количество разных способов развлечения для людей. Начиная с прослушивания музыки и заканчивая чтением книг и просмотров фильмов. В данный момент на рынке есть большое количество онлайн-кинотеатров. Каждый из них имеет разную базу данных фильмов. Пользователям важно быстро и легко находить фильмы, которые скорее всего им могут понравиться. Для этого было создано множество рекомендационных алгоритмов машинного обучения. Но для этих алгоритмов нужны данные. Есть множество способов обогатить данные фильмов. Например, учитывать жанр фильма, актеров и режиссера. Одним из способов обогатить данные для улучшения точности предугадывания создание облаков тегов для фильмов.

Цели и задачи

Целью данной работы является разработка приложения для создания облаков тегов для фильмов.

Для достижения цели необходимо выполнить следующие задачи:

1) Исследовать возможности создания облаков тегов
2) Исследовать SaaS сервисы, которые могут помочь в решении данной проблемы
3) Разработать приложение для получения облаков тегов на основе входной информации
4) Отладить приложение

1. Анализ предметной области

1.1 Общие сведения

Тег – ключевой слово для категоризации в фолксономиях. Фолксономия – народная классификация, практика совместной категоризации информации посредством произвольно выбираемых меток, называемых тегами [1]. Облако тегов – это визуальное представление списка категорий или тегов [2]. Теги могут создаваться на основе характеристик фильмов. Такие характеристики включают в себя: синопсис или краткий сюжет, список главных актеров и актрис, список режиссёров и продюсеров, в редких случаях указывают оставшиеся позиции съемочной группы, такие как: монтажер, оператор камеры и звукорежиссёр. В данной работе ключевыми характеристиками фильма являются трейлеры, так как из них можно получить больше информации.

1.2 Способы обработки характеристик фильмов

Есть множество способов получить теги фильмов. Например, взять готовые списки с открытых Интернет-ресурсов, таких как: Kinopoisk [3] и IMDB [4]. Но это не слишком надежно, так как у малоизвестных фильмов, списки тегов буду маленькими или вовсе отсутствовать.

Так, что данный способ является некорректным, так как необходимо получить список тегов для всех фильмов из предоставленной базы данных. Также, приложений с аналогичным разрабатываемым функционалом не было найдено на рынке.

Существует несколько разных способов извлечь данные из трейлеров. Ключевые из них покадровый анализ и анализ звуковой дорожки. Покадровый анализ можно совершать вручную или автоматически. Анализ звуковой дрожки тоже можно совершать вручную или автоматически. Основные способы автоматического звукового анализа:

- Скрытая Марковская модель [5]
- Алгоритм динамической трансформации временной шкалы [6]
- Искусственная нейронная сеть [7]
- Алгоритм глубокого обучения [8]

Далее рассматриваются алгоритмы, используемые для покадрового анализа:

- Faster R-CNN [9]
- YOLO [10]
- SSD [11]

Все вышеперечисленные алгоритмы, являются алгоритмами глубокого обучения, пояснения будут даны в следующей главе.

2. Проектирование приложения

В разрабатываемом приложении было решено использовать покадровый анализ трейлера, а не анализ звуковой дорожки, так как у алгоритмов анализа речи существует большой недостаток. Этим недостатком является множество языков, так как не все фильмы выходят в широкий прокат и получают трейлер с английской звуковой дорожкой. Следовательно, пришлось бы учитывать много языков и настраивать каждый из алгоритмов по многу раз. В покадровом анализе такой проблемы не существует, вследствие чего был выбран именно он.

Как упоминалось ранее, покадровый анализ можно осуществить вручную или создать искусственную нейронную сеть. Для того, чтобы провести анализ вручную, придется нанимать отдельного человека. Работоспособность человека ограничивается 40 часами в неделю, следуя ТК РФ [12] и его минимальная зарплата будет равна 11 280 рублей [13]. Проведя простые расчеты, человек будет тратить 71 рубль в час. Так же исходя из эксперимента, человек может проанализировать 2 трейлера за час. Исходя из вышеперечисленных недостатков, было

решено использовать искусственную нейронную сеть для покадрового анализа. Искусственная нейронная сеть - математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы. Первой такой попыткой были нейронные сети У. Маккалока и У. Питтса [7]. Нейронную сеть будем обучать с помощью алгоритмов глубоко обучения для компьютерного зрения. Глубокое обучение – совокупность методов машинного обучения, основанных на обучении представлениям, а не специализированным алгоритмам под конкретные задачи [8]. Компьютерное зрение – научная область, которая занимается тем, как можно создать компьютеры для получения высокого уровня понимания цифровых изображений или видео [14].

Проектирование приложения начинается с подготовки всех значимых входных данных. Для данного приложения входными данными является каталог или база данных, то

есть некое хранилище с нижеперечисленными характеристиками фильма:

- Название фильма
- Неформальная транслитерация названия
- Уникальный идентификатор
- Год создания
- Ссылка на трейлер

После чего должен быть настроен канал передачи информации из хранилища в разрабатываемое приложение. Следующим шагом, в действие приводится искусственная нейронная сеть для анализа трейлера. В конце приложение должно получить ответ от нейронной сети и преобразовать в удобный для нас вид. Ниже представлен рисунок предварительной архитектуры приложения.

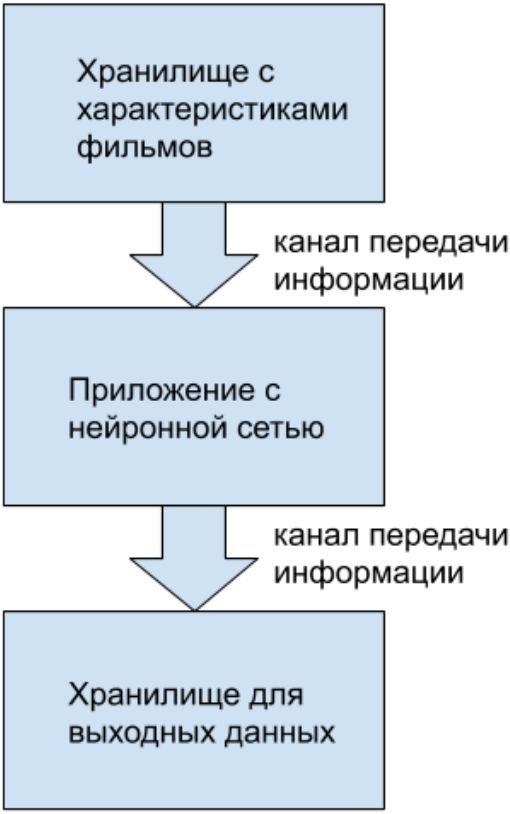

Рисунок 1 Предварительная архитектура приложения

По итогам преобразования на выходе приложения получается список из структур, которые хранят: уникальный идентификатор, название фильма, неформальная транслитерация названия, список тегов, полученных от искусственной нейронной сети и веса для

каждого уникального тега. Вес тега – это нормализованное количество данного уникального тега относительно количества всех тегов, обнаруженных искусственной нейронной сетью. Нормализация проводилась, следуя данной формуле:

$$W = m / N$$

, где m – количество уникального тега, встречающегося за данный трейлер, а N – общее количество всех тегов, обнаруженных при анализе трейлера.

Так же для разрабатываемого приложения поднимался вопрос о способе деплоя данного приложения. Существует два основных способа деплоя: на локальной собственной машине и аренда удаленного сервера в виде облачного сервиса. У каждого из этих способов есть преимущества и недостатки. Рассмотрим их далее. Главным недостатком традиционного деплоя, то есть деплоя на собственной локальной машине, является наличие данной машины, ее настройка и администрирование. Главным плюсом является то, что в чрезвычайных ситуациях сохраняется полный доступ к машине. У облачного деплоя, то есть аренды удаленного сервера, все наоборот. Ниже приведена сводная

таблица из преимуществ и недостатков каждого из способов.

Таблица 1 Преимущества и недостатки способов деплоя

Способ деплоя	Традиционный деплой	Облачный деплой
Ответственность за проблемы физического характера	Присутствует	Отсутствует
Взимаемая плата	За электричество и охлаждение	Только арендная плата
Недоступность сервера связанная с проблемами у третьих сторон	Отсутствует	Присутствует

Для данного проекта было решено выбрать не традиционный деплой как stand-alone приложения на собственном локальном сервере, а использовать сервисы SaaS для облачного деплоя. Данное решение было вызвано отсутствием собственного сервера и сжатыми сроками

разработки приложения. Также плюсом является проверка выгодности сервисов для деплоя и их работоспособности. Если будут получены выгодные результаты, использование данных сервисов в других проектах.

3. Разработка приложения

3.1 Архитектура приложения

Все существующие функции необходимые для разрабатываемого приложения, присутствуют на стеке сервисов Amazon. Также настройка сервисов Amazon проста, а документация данных сервисов подробна. Поэтому для данного приложения были выбраны сервисы от Amazon. Также предоставленный каталог со всей нужной информацией о фильмах уже находится в ранее созданном облачном хранилище Amazon Simple Storage Service. Список используемых сервисов:

- Rekognition [15]
- Simple Storage Service [16]
- Simple Notification Service [17]
- Simple Queue Service [18]
- Lambda [19]

Для экономии времени, так как большинство используемых сервисов, оплачиваются по времени использования, и облегчения отладки всего приложение было решено поделить на 3 модуля-функции, которые будут исполняться сервисом Lambda по специальному событию-уведомлению. Первый модуль отвечает за

создание очереди в ранее созданной SQS queue из уникальных id фильмов из базы данных. Queue – это очередь сообщений от сервиса SQS, при помощи которого происходит передача информации между модулями. Второй модуль отвечает за загрузку трейлера в S3, так как сервис Rekognition может анализировать только те видеофайлы, которые хранятся в S3, и после загрузки делает запрос к Rekognition на начало анализа данного трейлера. Последний из трех модулей отвечает за получение ответа от Rekognition и сохранение его в S3. Все модули передают информацию друг-другу через настроенный ранее канал из SQS queue и SNS topic. Ниже представлена архитектура приложения.

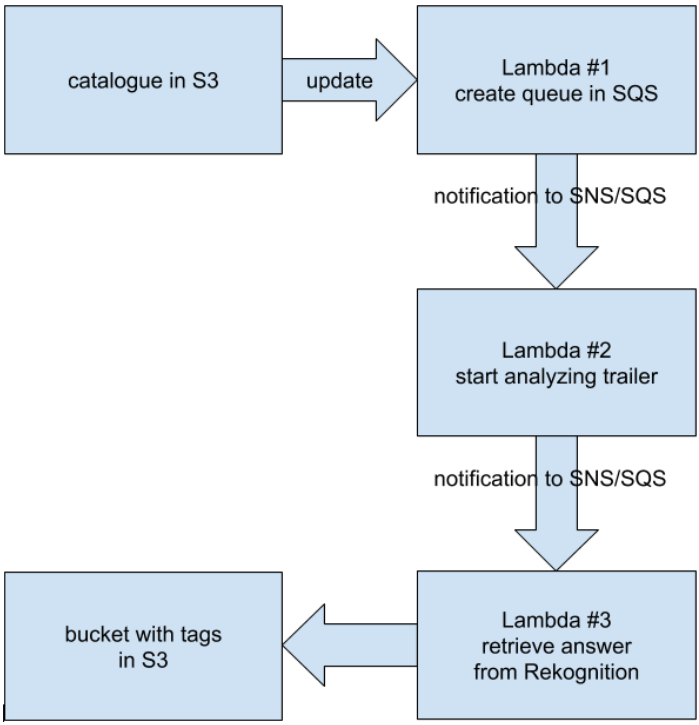

Рисунок 2Архитектура приложения

3.2 Используемые сервисы SaaS

Rekognition - это сервис для анализа изображений и видео на базе глубокого обучения. Сервис умеет распознавать объекты, людей, текст, сцены и действия. Ниже приведены основные преимущества и возможности данного сервиса. Непрерывное обучение: сервис постоянно обучается с использованием новых данных, благодаря чему

расширяется его способность распознавать объекты, сцены и действия и повышается точность распознавания. Полное управление: время отклика Amazon Rekognition стабильно при любых объемах запросов. Задержка приложения останется неизменной даже при росте количества запросов до десятков миллионов. Пакетный анализ и анализ в режиме реального времени: сервис позволяет анализировать видео из Amazon Kinesis Video Streams в режиме реального времени и изображения по мере загрузки в Amazon S3. Низкая стоимость: при использовании Amazon Rekognition оплачивается только количество проанализированных изображений или минут видео, а также хранение данных о лицах для последующего распознавания.

S3 – это сервис хранения объектов, предоставляющий функции архивирования и резервного копирования данных, а также функции администрирования и анализа объектов.

Lambda – это сервис бессерверных вычислений, который запускает программный код в ответ на определенные события и отвечает за автоматическое выделение необходимых вычисленных ресурсов.

SNS - это высокодоступный, надежный, безопасный, полностью управляемый сервис отправки сообщений по модели «издатель – подписчик» (Pub/Sub), с помощью которого можно изолировать микросервисы, распределенные системы и бессерверные приложения.

SQS - это полностью управляемый сервис очередей сообщений, с помощью которого можно изолировать и масштабировать микросервисы, распределенные системы и бессерверные приложения.

3.3 Подготовка сервисов

В самом начале написания кода приложения, необходимо астроить сервисы. В S3 необходимо создать два bucket – виртуальное облачное хранилище. В одном из них будут храниться трейлеры фильмов, а в другом выходные данные – теги к фильмам. Трейлеры фильмов должны храниться в S3, так как этого требует Rekognition, данный сервис может анализировать только те файлы, которые находятся в S3.

После нужно настроить SNS и SQS. Для настройки SNS необходимо создать topic, который служит для передачи информации между функциями нашего приложения. Для создания topic, необходимо зайти в Amazon SNS Console, и

там выбрать пункт «Create topic». Задав имя нашему новому topic, мы получим его Amazon Resource Name (ARN) – уникальный идентификатор для сервисов Amazon, который понадобится нам для созданий запросов через API. Для настройки SQS, мы должны создать queue, который служит для хранения сообщений, который транспортируются при помощи SNS. Так же, как и для SNS, вначале мы должны зайти в Amazon SQS Console и выбрать там пункт «Create New Queue». При создании queue можно изменять параметры, такие как максимальный размер сообщений, длительность хранения сообщений, длительность задержки доставки и длительность видимости сообщения. Для разрабатываемого приложения, достаточно стандартных параметров, которые предлагает сам Amazon. После создания queue и topic, необходимо связать их. Для этого требуется зайти в Amazon SQS Console, выбрать уже созданный queue и в пункте «Queue Actions» подписать через «Subscribe Queue to SNS Topic». После этого необходимо указать ARN нашего topic, и выбрать «Subscribe». После чего необходимо установить разрешение на queue для получения сообщений из SNS topic. Для этого мы должны создать уникальное policy для нашего queue, в котором мы должны указать, что мы

разрешаем нашему topic с указанным ARN, действие «sqs:SendMessage». Теперь все сообщения, передаваемые через SNS topic, будут сохраняться в SQS queue. Теперь у нас есть собственный канал передачи данных, который используется для общения межу сервисами Amazon и нашими функциями. Далее нужно настроить Rekognition и Lambda.

Для настройки сервиса Rekognition нужно создать обслуживающую роль, у которой будут данные разрешения: «AmazonSQSFullAccess», «AmazonRekognitionFullAccess» и «AmazonS3ReadOnlyAccess». Также данная роль должна позволять Rekognition общаться с ранее созданными SNS topic. Теперь сервис Rekognition готов к работе, перейдем к настройке Lambda. Для Lambda нужна роль, которая позволит данному сервису исполнять наш код и получать доступ к S3. После настройки Lambda, все сервисы готовы к работе.

3.4 Модуль создания очереди

Первый модуль отвечает за создание очереди из фильмов, трейлеры которых должны быть загружены в S3 и проанализированы сервисом Rekognition. Данная функция

начинает свое выполнение, когда база данных, хранящаяся в S3 обновляется. Обновление базы данных происходит 3 раза в день, и в каждое обновление новые фильмы как могут быть добавлены, так и не могут. Для предотвращения повторного анализа уже проанализированных трейлеров, нам нужно сопоставлять список уже загруженных трейлеров и фильмов из базы данных по их индивидуальному id, заранее сгенерированному. Так как стандартная функция пакета botocore s3.list_objects_v2() не поддерживает возвращение полного списка объектов в bucket S3, то нужно создать собственную функцию. Наша функция использует параметр «ContinutationToken», что позволяет извлечь все объекты, находящиеся в нужном bucket, а не только первые 100 объектов. После того, как у нас есть полный список уже загруженных трейлеров, нам нужно обработать всю базу данных фильмов. База данных «дампится» в json файл и загружается в S3. Данный файл называется каталог, и мы будем использовать его как реплику базы данных, для уменьшения вычислительной нагрузки и сохранения конфиденциальности нашей базы данных с фильмами. Так как каталог создается в .json.gz формате, то для упрощенния навигации по нему, воспользуемся стандартными возможностями языка Python.

Воспользуемся пакетами BytesIO и GzipFile, чтобы разархивировать каталог, используя функции BytesIO().read() и GzipFile().read().decode(). В итоге получаем обычный json. Чтобы обращаться к нему будем использовать стандартный пакет json, и функцию из него json.loads(). Проходя поэлементно по каталогу, мы найдем все фильмы, трейлеры которых еще не скачены и не проанализированы. После того как, такой фильм находится, используя функцию sns.publish() из пакета botocore, отправляем данные по этому фильму в нашу очередь. Из нее следующий модуль возьмет информацию по фильму и использует по назначению.

3.5 Модуль анализа трейлера

Второй модуль отвечает за загрузку трейлера фильма в S3 и отправление запроса на анализ этого трейлера. Событие из-за которого выполняется данная функция - появление в очереди сообщения с данными по трейлеру. А именно, это url трейлера с локального сервера и его id. Так как пакет botocore не имеет функции загрузки файла в S3 из url, нам придется написать ее самим. Используя стандартный пакет requests и функцию requests.get().raw.read(), мы можем извлечь трейлер фильма

из url и загрузить во временную память. После чего загрузить в S3 используя функцию пакета botocore s3.put_object() в формате mp4. Название трейлера в S3 образуется из id фильма для упрощения нахождения уже загруженных трейлеров фильмов. После того как загрузка произошла, используя функцию пакета botocore recognition.start_label_detection(), отправляем сервису Rekognition запрос на анализ данного трейлера. Данная функция возвращает стандартный словарь Python, из которого мы можем извлечь JobID – индивидуальный номер запроса на анализ. Мы сохраняем связку из JobID и id фильма в отдельный файл, так как они понадобится нам для получения ответа от Rekognition и распознавания какой трейлер фильма был проанализирован. На этом работа второго модуля заканчивается.

3.6 Модуль сбора ответов

Третий модуль ответственен за сбор ответа от Rekognition и преобразования его в нужный нам вид. Событие-уведомление, по которому выполняется данная функция, - это автоматическое сообщение о конце анализа от Rekognition, отправленное в ранее созданный нами SNS topic. Так как ответ от Rekognition не содержит имя

проанализированного файла, мы не знаем какому трейлеру данный ответ принадлежит. Чтобы решить данную проблему, мы заранее сохраняли id фильма и JobID анализа, и сопоставив JobID в сохраненом файле с JobID из ответа, мы сможем узнать id фильма. Ответ от Rekognition получаем используя функцию из пакета botocore recognition.get_label_detection(). Эта функция возвращает стандартный словарь Python, в котором находятся список объектов, которые смог распознать алгоритм Rekognition. Пройдя поэлементно по этому списку, мы найдем все не повторяющиеся теги, подсчитаем их общее количество и индивидуальное количество. Индивидуально количество тегов нам нужно для вычисления весов – нормализованного коэффициента количества данного тега относительно общего количества тегов. Также мы опять воспользуемся каталогом, так как нам нужны такие данные как id фильма, его оригинальное название и название в неформальной транслитерации. В конце сохраним все в файл в формате json. После этого наше приложение заканчивает работу.

4. Результаты

4.1 Графическое представление данных

После того, как разработанное приложение полностью проанализировало все фильмы из изначальной базы данных, получается большой список из особых структур данных, описанных во второй главе. Так как изначально в базе данных хранилось более 10000 фильмов, то ручной анализ результатов представляется затруднительным, следовательно, эти данные нужно интерпретировать. Для этого было решено использовать графовую систему управления базами данных Neo4j. Она считается одной из наиболее распространённых систем такого рода. Эта система хранит данные в собственном формате, специализированно приспособленном для представления графовой информации, такой подход в сравнении с моделированием графовой базы данных средствами реляционной СУБД позволяет применять дополнительную оптимизацию в случае данных с более сложной структурой. Также утверждается о наличии специальных оптимизаций для SSD-накопителей, при этом для обработки графа не требуется его помещение целиком в оперативную память вычислительного узла, таким образом, возможна обработка достаточно больших графов. Так же отличительной чертой

является то, что данный проект open source. Исходя из выше перечисленных преимуществ, была выбрана именно эта система. У данной системы свой собственный язык запросов – Cypher, при помощи которого все наши данные загружались в граф с построением связей фильм-тег. Так как отображение всего графа является затруднительным, так как рендеринг графа происходит за счет вычислительных характеристик клиента. Поэтому было решено отрендерить граф только для 100 случайных фильмов. Ниже представлены рисунки графа, на которых красные окружности - это теги, а зеленые - это фильмы.

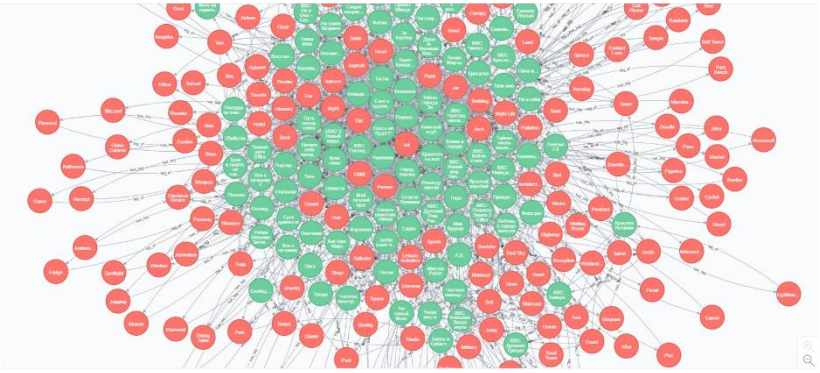

Рисунок 3 Графическое представление облаков тегов

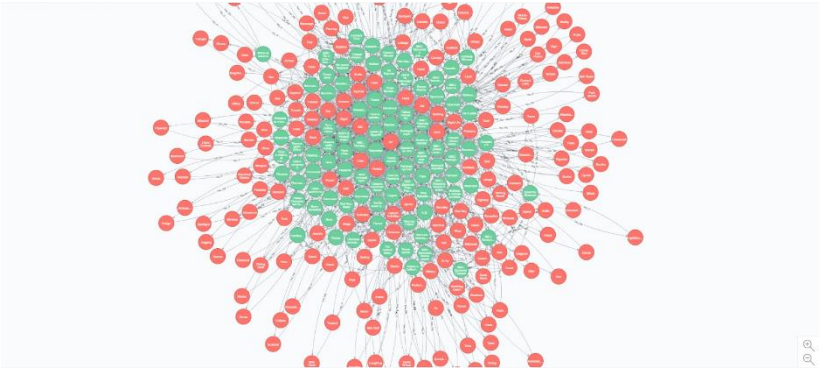

Рисунок 4 Графическое представление облаков тегов

На скриншотах видно, что большинство фильмов находятся близко, и есть теги, присутствующие почти у всех фильмов. А также присутствуют фильмы, находящиеся поодаль. Данное разбиение на облака тегов является хорошим, так как присутствуют и близкие, и дальние связи тегов и фильмов.

4.2 Экономическая выгода

Стоит отметить экономическую эффективность данного приложения. Сервис Lambda позволяет работать приложению постоянно за небольшую плату, так как, как только обновляется каталог с фильмами, приложение начинает работать, в остальных случаях плата не взымается. Так же плюсом является то, что данное приложение не выполняется пока ждет ответа от сервиса Rekognition,

иногда анализ трейлера может доходить до 10 минут. Длительность анализа зависит от длительности трейлера и как часто там появляются новые кадры. Исходя из выше перечисленного и цен на сервисы Amazon, стоимость анализа 1 минуты трейлера будет обходится мене чем в 0.03 USD. Если сравнивать экономическую эффективность с обычным приложением, то данное решение выигрывает, так как кроме затрат на сервисы Amazon, хоть и вычитаются затраты на Lambda, пришлось бы иметь машину, которая постоянно работает, а это дополнительные затраты на электричество. А если такой машины нет, то пришлось бы ее арендовать. Для сравнения у Amazon есть сервис для аренды обычных облачных вычислительных ресурсов Amazon EC2. Если подсчитать его затраты, то выходит 0.05 USD за анализ 1 минуты трейлера. Если проводить сравнение с человеческими ресурсами, то данное приложение так же имеет преимущества. Ниже приведена таблица сравнения.

Таблица 2 Сравнение эффективности

	человек	приложение
Количество	100	До 1000

тэгов (шт.) для одного трейлера		
Затраченное время (мин.) на один трейлер	30	10
Работоспособность (час/нед)	40	168
Оплата (USD/час)	1	1.8

Как видно из таблицы данное приложение имеет все преимущества перед обычным человеком, не смотря на повышенную оплату. Только в час разработанное приложение обрабатывает в 3 раза больше трейлеров чем человек, и при этом не имеет ограничений по времени работы.

Выводы

В результате данной исследовательской работы были получены навыки использования и настройки сервисов Amazon Web Services и навыки применения API AWS. Спроектированное приложение полностью удовлетворяет нашим изначальным требованиям. Так же важно отметить общие преимущества использования сервисов. Нам не нужно хранить все трейлеры у себя локально, а это примерно 1.5 Тб информации. Нам не нужно было собственноручно создавать, обучать и тестировать нейронную сеть для распознавания объектов в видеофайлах. Amazon гарантирует, что их сервис Rekognition постоянно обучается, что является огромным плюсом, так как самостоятельно поддерживать нейронную сеть трудно. Сервисы SNS и SQS предоставляют безопасный способ передачи информации. Для каждого сервиса Amazon есть подробная документация, в которой описано как его настраивать и как им пользоваться. В итоге можно сказать, что разработка приложений с использованием сторонних сервисов SaaS, выгодно как с

экономической точки зрения, так и с точки зрения эффективности затраченного времени на разработку.

Библиографический список

1. Isabella Peters. Folksonomies. Indexing and Retrieval in Web 2.0

2. Martin Halvey, Mark T. Keane. An Assessment of Tag Presentation Techniques

3. Интернет-ресурс Кинопоиск: https://kinopoisk.ru (дата обращения 05.08.2018)

4. Интернет-ресурс IMDB: https://imdb.com (дата обращения 05.08.2018)

5. Baum, Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains.

6. Silva, Batista. Speeding Up All-Pairwise Dynamic Time Warping Matrix Calculation.

7. Мак-Каллок У. С., Питтс В. Логическое исчисление идей, относящихся к нервной активности (перевод английской статьи 1943 г.)

8. Rina Detcher. Learning while searching in constraint-satisfaction problems

9. Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region

Proposal Networks: https://arxiv.org/abs/1506.01497 (дата обращения 06.04.2019)

10. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection: https://arxiv.org/abs/1506.02640 (дата обращения 07.04.2019)

11. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector: https://arxiv.org/abs/1512.02325 (дата обращения 08.04.2019)

12. Статья 91 Трудового Кодекса Российской Федерации

13. Федеральный закон от 28.12.2017 №421-ФЗ

14. Dana H. Ballard, Christopher M. Brown. Computer Vision

15. Amazon Rekognition Developer Guide: https://docs.aws.amazon.com/en_us/rekognition/latest/dg/what-is.html

16. Amazon Simple Storage Service Developer Guide: https://docs.aws.amazon.com/en_us/AmazonS3/latest/dev/Welcome.html

17. Amazon Simple Notification Service Developer Guide: https://docs.aws.amazon.com/en_us/sns/latest/dg/welcome.html

18. Amazon Simple Queue Service Developer Guide: https://docs.aws.amazon.com/en_us/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html

19. Amazon Lambda Developer Guide: https://docs.aws.amazon.com/en_us/lambda/latest/dg/welcome.html

Приложение

Текст программы.

```
import boto3

import json

from io import BytesIO

from gzip import GzipFile

s3 = boto3.client('s3')

def get_all_s3_keys(bucket):

    """Get a list of all keys in an S3 bucket."""

    keys = []


    kwargs = {'Bucket': bucket}

    while True:

        resp = s3.list_objects_v2(**kwargs)

        for obj in resp['Contents']:

            keys.append(obj['Key'][:-4])
```

```
        try:
            kwargs['ContinuationToken'] =
resp['NextContinuationToken']
        except KeyError:
            break

    return keys

def lambda_handler(event, context):
    topicArn = 'exampleTopicArn'
    sns = boto3.client('sns')
    numberOfTrailers = 6
    i = 0
    j = 0

    finishedTrailers = get_all_s3_keys('bucket_name')
```

```
    retr = s3.get_object(Bucket='bucket_name',
Key='catalogue.json.gz')

    bytestream = BytesIO(retr['Body'].read())

    catalogue = GzipFile(None, 'rb',
fileobj=bytestream).read().decode('utf-8')

    cat_json = json.loads(catalogue)


    while (i < numberOfTrailers) and (j <
len(cat_json['movies'])):

        if cat_json['movies'][j]['id'] not in finishedTrailers:

            sns.publish(TargetArn=topicArn,

            Message=json.dumps(cat_json['movies'][j]['id'] + ' ' +
cat_json['movies'][j]['trailerUrlSd']) +

                    ' need to upload')

            i += 1

        j += 1
```

```
    i = 0
    j = 0

    while (i < numberOfTrailers) and (j <
len(cat_json['serials'])):
        if cat_json['serials'][j]['id'] not in finishedTrailers:
            sns.publish(TargetArn=topicArn,
            Message=json.dumps(cat_json['serials'][j]['id'] + ' ' +
cat_json['serials'][j]['trailerUrlSd']) +
                    ' need to upload')
            i += 1
        j += 1

    i = 0
    j = 0
```

```
    while (i < numberOfTrailers) and (j <
len(cat_json['multipartMovies'])):
        if cat_json['multipartMovies'][j]['id'] not in
finishedTrailers:
            sns.publish(TargetArn=topicArn,

Message=json.dumps(cat_json['multipartMovies'][j]['id'] + ' ' +
                cat_json['multipartMovies'][j]['trailerUrlSd']) +
                ' need to upload')
            i += 1
        j += 1
import json
import boto3
from io import BytesIO
from gzip import GzipFile

dataTags = {}
```

```
s3 = boto3.client('s3')

rek = boto3.client('rekognition', region_name='eu-west-1')



def getIndex(uid, catalogue)->[str,int]:

    for i in range(len(catalogue['movies'])):

        if uid == catalogue['movies'][i]['id']:

            return ['movies', i]

    for jk in range(len(catalogue['serials'])):

        if uid == catalogue['serials'][jk]['id']:

            return ['serials', jk]

    for kk in range(len(catalogue['multipartMovies'])):

        if uid == catalogue['multipartMovies'][kk]['id']:

            return ['multipartMovies', kk]



def getUid(jobId, catalogue) -> str:
```

```
    for i in range(len(catalogue)):
        if jobId == catalogue[i]["jobId"]:
            return catalogue[i]["uid"]


def lambda_handler(event, context):
    message = event['Records'][0]['Sns']['Message']
    RekAnswr = json.loads(message)
    if RekAnswr["Status"] == 'SUCCEEDED':

        retr = s3.get_object(Bucket='bucket_name',
Key='catalogue.json.gz')
        bytestream = BytesIO(retr['Body'].read())
        catalogue = GzipFile(None, 'rb',
fileobj=bytestream).read().decode('utf-8')
        cat_json = json.loads(catalogue)
```

```
        retr = s3.get_object(Bucket='bucket_name',
Key='jobIds.json')

        bytestream = BytesIO(retr['Body'].read())

        catalogue = GzipFile(None, 'rb',
fileobj=bytestream).read().decode('utf-8')

        jobIds = json.loads(catalogue)


        resp = rek.get_label_detection(JobId=RekAnswr['JobId'],
MaxResults=1000, NextToken='', SortBy='TIMESTAMP')

        tags = []

        weights = {}

        all_labels = 0


        uid = getUid(RekAnswr['JobId'], jobIds)

        _type, index = getIndex(uid, cat_json)


        for label in resp['Labels']:
```

```
    all_labels += 1
    if label['Label']['Name'] not in tags:
        tags.append(label['Label']['Name'])
        weights[label['Label']['Name']] = 1
        continue
    else:
        weights[label['Label']['Name']] += 1
for k, v in weights.items():
    weights[k] = float(weights[k] / all_labels)
dataTags = {
    'uid': uid,
    'name': cat_json[_type][index]['name'],
    'alias': cat_json[_type][index]['alias'],
    'tags': tags,
    'weights': weights,
    }
```

```
        json_str = json.dumps(dataTags, ensure_ascii=False) +
"\n"
        json_bytes = json_str.encode('utf-8')
        gz_body = BytesIO()
        gz = GzipFile(None, 'wb', 9, gz_body)
        gz.write(json_bytes)
        gz.close()
        s3.put_object(
            Bucket='bucket_name',
            Key='tags/tags_'+uid+'.json',
            ContentType='application/json',
            ContentEncoding='gzip',
            Body=gz_body.getvalue()
        )
import json
import boto3
from botocore.vendored import requests
```

```
from io import BytesIO

from gzip import GzipFile


s3 = boto3.client('s3')

topicArn = "exmapleTopicArn"

roleArn = 'exampleRoleArn'

rek = boto3.client('rekognition', region_name='eu-west-1')

trigger = ['need', 'to', 'upload']


def lambda_handler(event, context):

    trailerData =  event['Records'][0]['Sns']['Message'].split()

    if all(x in trailerData for x in trigger):

        req_for_video = requests.get(trailerData[1][:-1],
stream=True)

        fo_from_req = req_for_video.raw

        req_data = fo_from_req.read()
```

```
        s3.put_object(Bucket='trailers_bucket_name',
Key=trailerData[0][1:]+'.mp4', Body=req_data)

        response = rek.start_label_detection(Video={'S3Object':
{'Bucket': 'trailers_bucket_name',

                                                'Name': trailerData[0][1:]
+ '.mp4'}},

                                    NotificationChannel={'RoleArn':
roleArn, 'SNSTopicArn': topicArn})


        retr = s3.get_object(Bucket='bucket_name',
Key='jobIds.json')

        bytestream = BytesIO(retr['Body'].read())

        catalogue = GzipFile(None, 'rb',
fileobj=bytestream).read().decode('utf-8')

        jobIds = json.loads(catalogue)

        jobIds.append({'uid': trailerData[0][1:], 'jobId':
response['JobId']})

        json_str = json.dumps(jobIds, ensure_ascii=False) + "\n"

        json_bytes = json_str.encode('utf-8')
```

```
gz_body = BytesIO()
gz = GzipFile(None, 'wb', 9, gz_body)
gz.write(json_bytes)
gz.close()
s3.put_object(
    Bucket='bucket_name',
    Key='jobIds.json',
    ContentType='application/json',
    ContentEncoding='gzip',
    Body=gz_body.getvalue()
)
```