Министерство науки и высшего образования Российской Федерации

МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ (государственный университет)

ФАКУЛЬТЕТ УПРАВЛЕНИЯ И ПРИКЛАДНОЙ МАТЕМАТИКИ

КАФЕДРА ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Специализация 011600 «Прикладные физика и математика»

“СЕРВИС ОПТИМИЗАЦИИ ГИПЕРПАРАМЕТРОВ АЛГОРИТМОВ КЛАССИФИКАЦИИ”

Выпускная квалификационная работа
(магистерская диссертация)

Выполнила:
студент 273 группы _____Мусинов И.И.

Научный руководитель:
к.ф.-м.н., доцент _____Устюжанин А.Е.

Москва – 2018

Содержание

1.Введение.....3
2.Постановка проблемы.....4
3. Сервис по оптимизации гиперпараметров алгоритмов классификации.....4
4. Анализ существующих программных средств.....5
4.1. Библиотеки оптимизации.....6
4.2. Облачные платформы.....7
4.3. Существующие решения.....7
4.4 Преимущества разработки по сравнению с аналогами.....8
5. Технические характеристики.....8
5.1. Описание архитектуры сервиса.....8
5.2. Описание алгоритма и функционирования программы.....9
5.2.1. Описание функционирования серверной части программы.....9
5.2.2. Описание функционирования клиентской части программы.....13
5.3. Описание и обоснование выбора состава технических и программных средств 14
5.3.1. Описание выбора состава технических средств.....14
5.3.4. Обоснование выбора состава программных средств.....15
5.4. Метод организации входных и выходных данных.....16
6. Вычислительный эксперимент.....16
6.1. Оптимизация гиперпараметров.....16
.....21
7.Заключение.....22
8. Список использованных источников.....23

1. ВВЕДЕНИЕ

В данный момент области науки такие как машинное обучение и анализ данных стремительно развиваются. Появляется все больше и больше задач, требующих навыков специалистов в данных областях. Один из важных навыков в данной сфере является правильная настройка алгоритмов классификации, которая состоит в подборе гиперпараметров алгоритма. Гиперпараметры - это параметры которые не меняются в процессе обучения модели. Набор гиперпараметров считается наилучшим, если на этом наборе метрики модели достигают экстремумов. Каждый алгоритм машинного обучения обладает некоторым набором гиперпараметров. Необходимо подбирать такой набор гиперпараметров, чтобы не допускать переобучения модели, а также слишком большого потребления ресурсов во время обучения модели. Не существует строгого алгоритма подбора гиперпараметров, поэтому в каждом конкретном случае исследователи вынуждены перебирать некоторое количество комбинаций гиперпараметров для получения наилучшей модели.

С целью ускорения процесса поиска наилучшей комбинации гиперпараметров, создаются библиотеки оптимизации (hyperopt, skopt, spearmint). Процесс обучения модели и ее кросс-проверка очень затратные по времени и вычислительным ресурсам операции, поэтому в основном применяются алгоритмы Байесовской оптимизации, учитывающие значения метрик всех полученных моделей.

Кроме проблем оптимизации, существует проблема вычислительных ресурсов. Для получения модели с хорошими метриками за приемлемое время не всегда достаточно ресурсов персонального компьютера или выделенной виртуальной машины. В таких случаях исследователи используют кластер или облачную инфраструктуру для распределения вычислительной нагрузки между несколькими машинами.

Несмотря на разнообразие библиотек оптимизации и облачных провайдеров, проблема поиска сервиса, совмещающего в себе удобный интерфейс для

исследователей, необходимые и настраиваемые алгоритмы оптимизации, возможность проведения вычислений на кластере, для эффективного поиска набора гиперпараметров остается открытой.

2. ПОСТАНОВКА ПРОБЛЕМЫ.

Для исследователей в области машинного обучения необходим инструмент эффективного поиска наилучшей комбинации гиперпараметров алгоритмов классификации. Для этого необходимо создать программный комплекс со следующими требованиями:

1. Удобный пользовательский интерфейс.
2. Реализация алгоритмов оптимизации, оптимальных для ресурсоемких функций.
3. Возможность выполнения вычислений на кластере.

3. СЕРВИС ПО ОПТИМИЗАЦИИ ГИПЕРПАРАМЕТРОВ АЛГОРИТМОВ КЛАССИФИКАЦИИ

В некоторых областях науки, в частности в области машинного обучения и анализа данных, существует проблема воспроизводимости научных исследований. Под воспроизводимостью исследования здесь понимается совместное хранение компьютерных программ, использующихся для исследования, входных данных и информации об окружении (информации о программных средствах, необходимых для проведения исследования, в том числе номера их версий). Эта проблема сопряжена с трудностями при настройке окружения среды в которой происходит эксперимент. Данная проблема может возникать и при поиске оптимального набора гиперпараметров модели. Когда результаты метрик качества, полученные при обучении моделей, сильно варьируются или когда окружение для проведения эксперимента имеет конфликты для разных моделей. Эта проблема решается с помощью контейнеризации приложения (например с помощью Docker [21]).

Разработанный сервис «Wonderland» подходит для любых вычислений представимых в виде графов и решает проблему воспроизводимости этих вычислений, с помощью использования Docker-контейнеризации в своей архитектуре. Тем самым данная система может применяется в самых разных областях науки: биоинформатика, анализ данных, машинное обучение, вычислительная математика. Так как задачи из данных областей требуют больших вычислительных ресурсов, то программа разработана для эксплуатации на облачной платформе Azure.

Серверная часть разработанной программной системы представляет собой удобный инструмент для запуска и управления воспроизводимыми научными исследованиям. Она может использоваться в любых направлениях современной науки, в которых применяются вычислительные эксперименты, в особенности если данные вычисления параллелизуемые и представимы в виде графов.

Клиентская часть разработанной программной системы упрощает выполнение научных исследований конкретного вида, а именно выполнение задачи оптимизации гиперпараметров алгоритмов классификации.

4. АНАЛИЗ СУЩЕСТВУЮЩИХ ПРОГРАММНЫХ СРЕДСТВ

В начале рассмотрим существующие библиотеки оптимизации и облачные платформы, которые по отдельности удовлетворяют требованиям, предъявляемым к программному комплексу. В конце проанализируем существующие готовые решения.

4.1. Библиотеки оптимизации.

Исследователи, работающие в области анализа данных и машинного обучения, в основном используют язык программирования python. Поэтому на этом языке можно найти необходимые для нашей задачи библиотеки оптимизации. В основном используются следующие библиотеки:

1. skopt

2. hyperopt

3. spearmint

Это open-source библиотеки, разрабатываемые небольшими группами людей. Алгоритмы и интерфейсы данных библиотек не всегда полностью удовлетворяет потребностям пользователей. И что самое главное, не все библиотеки продолжают поддерживаться сообществом разработчиков и пользователей.

Библиотека hyperopt разрабатывается несколько лет, содержит пару алгоритмов оптимизации: Random Search и TPE(Tree of Parzen Estimators), но не имеет реализаций байесовских алгоритмов, которые наиболее эффективны в задачах с вычислением ресурсоёмкой функции, а интерфейс библиотеки затрудняет асинхронную оптимизацию. Еще один минус данной библиотеки – небольшое количество человек, участвующих в разработке и поддержке в данный момент времени. Из-за этого проблемы, возникающие в процессе эксплуатирования библиотеки, могут оставаться нерешаемыми. Из плюсов этой библиотеки можно отметить хорошую документацию и возможность использования локального хранилища для сохранения результатов вычислительных экспериментов.

Библиотека skopt наиболее подходящий инструмент для поиска гиперпараметров. Она обладает удобным интерфейсом с возможностью асинхронной оптимизации и набором байесовских алгоритмов оптимизации таких как [18].

4.**2. Облачные платформы.**

От облачных платформ требуется возможность создания кластера, наличия системы оркестровки данного кластера, а также возможность масштабирования кластера. Рассмотрим коммерческие облачные платформы:

1. AWS

2. Azure

3. Google Cloud

Все облачные платформы из данного списка предоставляют данный функционал пользователю.

4.**3. Существующие решения.**

Существующие программы, которые в той или иной мере решают поставленную задачу:

1. vowpal wabbit ()

2. Облачный сервис «Студия Машинного обучения Azure» (://azure.microsoft.com/ru-ru/services/machine-learning-studio)

Vowpal wabbit open-source программа, имеющая высокую скорость работы, так как в основе использует библиотеки на языке C++. Имеет функционал по работе с кластером Azure. Но данная программа не решает проблему управления облачным кластером, а также доработка данной программы для нужд пользователей может потребовать больших затрат.

Облачный сервис «Студия Машинного обучения Azure» - это коммерческий продукт с ограниченным набором моделей и алгоритмов оптимизаций, что накладывает ограничение на решаемые задачи и дополнительно расходует ресурсы исследователей.

Таким образом, встает необходимость создания собственного сервиса, удовлетворяющего описанным выше требованиям.

4.4 Преимущества разработки по сравнению с аналогами

Сервис wonderland предоставляет исследователям удобный интерфейс на базе библиотеки modelgym и дополняет его возможностью проводить вычисления в кластере на облачной платформе Azure. В сервисе возможно использовать любые алгоритмы машинного обучения и алгоритмы оптимизации для эффективного поиска гиперпараметров.

5. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

5.1. Описание архитектуры сервиса

Программа состоит из клиентской (wonderclient) и серверной части (wonderland). Серверная часть представляет собой приложение для исполнения научных исследований на кластере Kubernetes [4]. Она выполняет следующие функции:

1) Инициализация запуска графа вычислений (исследования) на доступных аппаратных ресурсах (распределенном кластере Kubernetes).

2) Получения результатов выполнения графа вычислений.

3) Предоставление пользователю доступа к вышеприведенным возможностям через фреймворк Google gRPC [5].

Клиентская часть приложения выполняет следующие функции:

1) Создание сообщения в котором хранится сериализованный в формате protobuff граф вычислений. Сам граф вычислений формируется на основе переданного данной программе описания набора гиперпараметров алгоритма.

2) Отправка сообщения серверной части.

3) Получение результатов обучения модели.

Клиентская часть написана на языке Python 3.6 [6], а также интегрирована с библиотеками оптимизации skopt [22], hyperopt [7] и modelgym [2] (библиотека по обучению моделей использующуюся в лаборатории LAMBDA).

5.2. Описание алгоритма и функционирования программы

5.2.1. Описание функционирования серверной части программы

Серверная часть программы написана на языке программирования Go. Сервер общается с клиентской частью посредством протока удаленных вызовов gRPC. Описание сообщений есть в файле wonderland.proto. Данный файл возможно

скомпилировать для языка программирования golang для использования интерфейса удаленных вызовов внутри кода серверного приложения. Серверная часть получает сообщения с описанием графов вычислений в формате cwl и запускается метод связанный с данным удаленным вызовом. Данный граф состоит из нескольких нод, каждая из которых запускает отдельно в kubernetes кластере. Перед запуском ноды сервер подсоединяется к kubernetes кластеру с помощью конфигурационного файла ~/.kube/config и создает kubernetes-спецификацию, которую отправляет в master-ноду kubernetes-кластера.

Kubernetes кластер принимает спецификацию ноды и создает докер-контейнер по ней. Данные для обучения и конфигурация модели копируются в контейнер из удаленного файлового хранилища AFS. В данном докер-контейнере происходит обучение модели и ее кросс-проверка. В конце вычисляются метрики качества на полученной модели. Результаты отправляются на удаленное файловое хранилище в виде файла в формате json.

Wonderland использует реляционную систему управления базами данных с открытым исходным кодом PostgreSQL для хранения информации о графах вычислений, отправляемых пользователями. В базе данных содержится две связанные таблицы pipeline_launch и kubernetes_job, описание которых приведено ниже в таблицах. Под pipeline подразумевается граф вычислений, состоящий из одной или нескольких вершин (нод). При каждом запуске нового pipeline информация о нем заносится в таблицу pipeline_launch. В одном pipeline может находиться несколько нод, каждая нода будет запущена на кластера. Информация о каждом запуске ноды графа заносится в таблицу kubernetes_job.

Описание таблицы pipeline_launch.

Столбец	Тип данных	Описание
id	BIGSERIAL PRIMARY KEY	Индекс записи
protobuf	BYTEA NOT NULL	Сообщение protobuff

status	INTEGER NOT NULL	Статус запущенного pipeline
created_at	TIMESTAMP NOT NULL	Дата запуска
created_by	VARCHAR(100)	Пользователь, запустивший pipeline

Описание таблицы kubernetes_job.

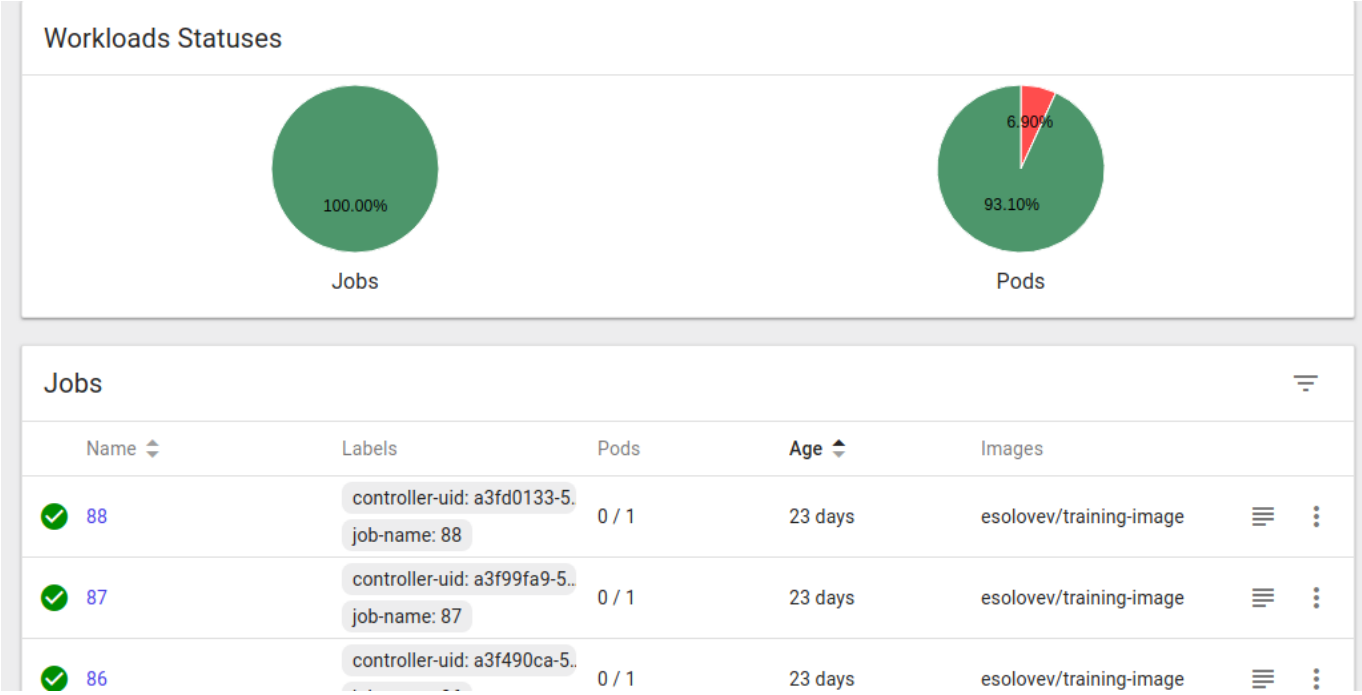
Столбец	Тип данных	Описание
id	BIGSERIAL PRIMARY KEY	Индекс записи
pipeline_launch_id	BIGSERIAL	Индекс соответствующего запуска pipeline
status	INTEGER NOT NULL	Статус запущенного pipeline
created_at	TIMESTAMP NOT NULL	Дата запуска
pipeline_node_name	VARCHAR(100)	Название ноды pipeline

При запуске сервера wonderland происходит подсоединение к база данных посредством сокета, перебрасываемого внутрь докер-контейнера. Также к базе данных возможно подсоединиться посредством URI. С помощью данной базы данных, возможен мониторинг в режиме реального времени использования кластера и запущенных на нем графов вычислений. Это позволяет отслеживает несанкционированный доступ к кластеру и определять сбои в системе вычислений. СУБД Postgres была выбрана в качестве базы данных, потому что является базой данных с открытым кодом и распространяется свободно. Данная база очень

популярна и поддерживается большой группой разработчиков, а также успешно зарекомендовала себя во многих проектах.

Авторизация производится посредством TLS-сертификатов [12]. Пользователи создают запрос на создание сертификата в формате csr и отправляют его администратору сервиса. Администратор сервиса подписывает данный запрос и получившийся сертификат отправляет обратно клиенту. После чего клиент может подсоединиться к серверу с помощью данного сертификата и своего ключа и передавать данные в зашифрованном виде.

Провайдером Kubernetes-кластера в данном случае выступает Microsoft Azure, предоставляющий соответствующий сервис. Для хранения входных и выходных данных используется сервис облачного файлового хранилища - Azure Files Storage. В качестве мониторинга kubernetes кластера может использоваться программа kubernetes dashboard.



Развертка серверной компоненты осуществляется посредством ansible-скриптов. Процесс развертки:

1) Создание докер-контейнера на виртуальной машине. В контейнер пробрасываются все необходимые файлы (скомпилированный код сервера, файл конфигурации kubernetes, сокет для базы данных postgres) , а также порт контейнера 50051 соединяется с портом хост-машины.

2) В контейнере настраиваются переменные окружения необходимые для работы сервера, описанные в json файле.

3) Запускается сервер. При запуске он подсоединяется к базе данных и начинает прослушивать входящие grpc сообщения от клиентов.

4) На хост машине присоединяется удаленное хранилище данных Aure File Storage.

Среди плюсов использования ansible-развертки следуют отметь возможность обновления сервера до новой версии.

5.2.2. Описание функционирования клиентской части программы

Клиентская часть представляет собой библиотеку wonderclient на языке программирования Python 3.6 и модификацию библиотеки modelgym.

Modelgym предоставляет интерфейс для обучения моделей машинного обучения. Она использует библиотеки оптимизации hyperopt (http://hyperopt.github.io/hyperopt/) и skopt (https://scikit-optimize.github.io/).

Библиотека wonderclient написана на языке программирования Python 3.6. Она интегрирована с библиотекой modelgym, позволяющей пользователю решать задачу оптимизации гиперпараметров. Она содержит класс ModelGymClient, который может быть передан библиотеке modelgym через параметр client с целью делегирования задач, возникающих во время процесса оптимизации гиперпараметров (обучения моделей машинного обучения и оценки их качества), на кластер Kubernetes, управляемый серверном wonderland.

```
%%time
models = ModelSpace(CtBClassifier,
                    space={'iterations': scope.int(hp.quniform('iterations', 100, 500, 100)),
                           'depth': scope.int(hp.quniform('depth', 1, 11, 1)),
                           'learning_rate': hp.loguniform('learning_rate', -5, -1),
                           'rsm': hp.uniform('rsm', 0, 1),
                           'leaf_estimation_method': hp.choice('leaf_estimation_method', ['Newton', 'Gradient']),
                           'l2_leaf_reg': scope.int(hp.quniform('l2_leaf_reg', 1, 10, 1)),
                           'bagging_temperature': hp.uniform('bagging_temperature', 0, 2),
                    space_update=True)
trainer = TpeTrainer(models)

##Cluster optimization
trainer.crossval_optimize_params(Accuracy(),
                                 path_to_data,
                                 metrics=[Accuracy()], client=client, opt_evals=2)
```

```
CPU times: user 96 ms, sys: 52 ms, total: 148 ms
Wall time: 49.6 s
```

В данном примере, в качестве алгоритма машинного обучения был выбран CtBClassifier, а также определенные области гиперпараметров, описанные в формате библиотеки hyperopt (http://hyperopt.github.io/hyperopt/). Алгоритм оптимизации TPE, реализованный в библиотеке hyperopt (http://hyperopt.github.io/hyperopt/), находит наилучшую комбинацию гиперпараметров.

При помощи библиотеки wonderclient создаётся графы вычислений из одной ноды (каждая нода-шаг заключается в обучении нескольких моделей машинного обучения и оценки их качества), данные для обучения и параметры алгоритмов в формате json отправляются в Azure File Storage. Итак, в графе содержится следующая информация:

1) Тип алгоритма машинного обучения

2) Ссылка на набор гиперпараметров алгоритма в AFS (Azure File Storage)

3) Ссылка на данные для обучения в AFS (Azure File Storage)

4) Докер-образ, который будет развернут в контейнере внутри kubernetes кластера, и необходимые для его успешного запуска параметры.

Данная информация сериализуется в сообщение wonderland_pb2.Pipeline и отправляеются по протоколу gRPC (https://grpc.io/) на сервер wonderland в ПОРВН. Обратно wonderclient получает значение метрик на обученной модели, которые

передает modelgym. Внутри modelgym алгоритм оптимизации, получив значения метрик, определяет следующий набор гиперпараметров для обучения модели.

5.3. Описание и обоснование выбора состава технических и программных средств

5.3.1. Описание выбора состава технических средств

Для бесперебойной работы серверной части приложения необходима виртуальная машине на облачной платформе Azure со следующими характеристиками:

1) 64-битный процессор Intel с частотой 1 ГГц или более быстрый.
2) >1 ГБ оперативной памяти (рекомендуется не менее 8 ГБ).
3) >2 ГБ свободного места на жестком диске
4) Бесперебойный доступ в сеть Интернет, статический IP-адрес

Также для работы системы вычислений необходимо наличие Kubernetes-кластера созданного с помощью acs-engine сервиса. Данный сервис обеспечивает автоматическое масштабирование кластера при увеличении вычислительной нагрузки на него.

Для исследователей, использующих данных сервис необходим персональный компьютер со следующими характеристиками:

1) 64-битный процессор Intel с частотой 1 ГГц или более быстрый.
2) >1 ГБ оперативной памяти (рекомендуется не менее 8 ГБ).
3) >2 ГБ свободного места на жестком диске (для установки всего необходимого окружения)

Требования к серверной виртуальной машине, обусловлены минимальными требованиями ОС Ubuntu 16.04 и программного обеспечения Docker. На данной виртуальной машине с помощью докер-контейнера развертывается серверная часть сервиса wonderland.

Требования к клиентскому оборудованию обусловлены требованиями бибилотек modelgym и wonderclient. Обе библиотеки используют интерпретатор Python 3.6.

5.3.4. Обоснование выбора состава программных средств

Требования к составу и параметрам программных средств для серверной части приложения обусловлены:

- широкой популярностью дистрибутива Linux Ubuntu Server 16.04
- увеличенным сроком поддержки ОС Ubuntu Server 16.04

Требования к составу и параметрам программных средств для клиентской части приложения обусловлены требованиями потенциальных пользователей сервиса (исследователи Научно-учебной лаборатории метода анализа больших данных используют либо Ubuntu 16.04, либо macOS последней версии, а также язык программирования Python последней версии), а также требованиями библиотеки modelgym к окружению.

5.4. Метод организации входных и выходных данных

Входными данными является сообщение сериализованное в формате Google Protocol Buffers. В данном сообщении содержится следующая информация:

1) Ссылка на тип и набор гиперпараметров алгоритма в AFS (Azure File Storage)

2) Ссылка на данные для обучения в AFS (Azure File Storage)

3) Докер-образ, который будет развернут в контейнере внутри kubernetes кластера, и необходимые для его успешного запуска параметры.

Данные для обучения и файлы json с конфигурацией алгоритмов берутся из Azure Files Storage. Сервис Azure File Storage был выбран по причине распределенности кластера и необходимости монтировать в контейнеры,

располагающиеся на каждой ноде (виртуальной машине) кластера, необходимые данные для проведения вычислений.

Выходными данными является сообщение в формате Google Protocol Buffers, а также файлы в формате json с результатами метрик, расположенные в Azure File Storage.

Формат Google Protocol Buffers имеет удобное описание в protobuff-виде, которое можно компилировать для многих языков программирования, что позволяет разрабытвать клиентскую и серверную части на разных языках, использующих единое пространство сообщений.

6. ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ

6.1. Оптимизация гиперпараметров

Дана «рабочая тетрадь» Jupyter Notebook (на языке Python 3.6), в которой используется библиотека modelgym с поддержкой wonderland. Покажем интерфейс для работы с modelgym и wonderclient, а также проведем несколько шагов оптимизации на кластере и на локальной машине.

Импортируем необходимые пакеты и создаем экземпляр клиента:

```
In [2]: from modelgym.models import CtBClassifier
        from modelgym.utils import XYCDataset
        from modelgym.trainers import TpeTrainer
        from modelgym.metrics import Accuracy
        from modelgym.report import Report
        from modelgym.utils import ModelSpace
        from hyperopt import hp
        import numpy as np
        from sklearn.datasets import make_classificati
        from pathlib import Path
```

Wonderland optimization

```
In [3]: from wonderclient import ModelGymClient
```

```
In [4]: client = ModelGymClient()
```

Рисунок 1.

Создаём случайный набор данных для построения моделей и сохраняем его в директорию, которая связана с AFS:

```
In [5]: n_features=20
        X, y = make_classification(n_samples=100, n_features=n_features, n_informative=10, n_classes=2)
        dataset = XYCDataset(X, y)
        y = np.array([y])
        data = np.concatenate((X, y.T), axis=1)
        path_to_data = Path.home() / "repo-storage/test/DATA/tmp_data12345678.csv"
        np.savetxt(path_to_data, data,
                   fmt='%.2f',
                   header=','.join([str(x) for x in range(n_features)]+['y']),
                   delimiter=',')
```

Рисунок 2.

Выполняем оптимизацию гиперпараметра learning rate (скорость обучения) для модели CatBoostClassifier двумя способами с помощью кластера и на локальной машине:

```
In [7]: %%time
        models = ModelSpace(CtBClassifier,
                            space={'learning_rate': hp.loguniform('learning_rate', -5, -1)},
                            space_update=True)
        trainer = TpeTrainer(models)

        ##Cluster optimization
        trainer.crossval_optimize_params(Accuracy(), path_to_data, metrics=[Accuracy()], client=client, opt_evals=2)

        CPU times: user 96 ms, sys: 52 ms, total: 148 ms
        Wall time: 49.6 s
```

Рисунок 3а. Кластерная оптимизация

```
In [8]: %%time
        ##Non-cluster optimization
        models = ModelSpace(CtBClassifier,
                            space={'learning_rate': hp.loguniform('learning_rate', -5, -1)},
                            space_update=True)
        trainer = TpeTrainer(models)

        trainer.crossval_optimize_params(Accuracy(), dataset, metrics=[Accuracy()], opt_evals=2)

        CPU times: user 4min 25s, sys: 46.3 s, total: 5min 11s
        Wall time: 2min 23s
```

Рисунок 3б. Оптимизация на локальной машине

В результате Wall time кластерной оптимизации 49.6с почти в 3 раза меньше оптимизации на локальной машине (2мин 23с), при условии почти одинаковых характеристик виртуальных машин и локальной машины. Затраченное процессорное

время в случае кластерной оптимизации очень мало в сравнении с не-кластерной оптимизацией, так как вся вычислительная нагрузка приходится на кластер. В reporter хранится результат оптимизации. В пользовательской панели k8s кластера (Kubernetes dashboard) появилась запись об успешно отработавших задачах (jobs). В базе данных PostgreSQL в соответствующих таблицах находятся строчки, соответствующие отправленным на сервер пайплайнам и их Kubernetes-задачам (jobs). Таким образом, серверная часть программы удовлетворяет функциональным требованиям по части исполнения вычислительных графов на Kubernetes-кластере, а клиентская – требованиям по наличию возможности решения задачи оптимизации гиперпараметров.

Рассмотрим интерфейс вызова процесса оптимизации:

```
In [11]: ##Non-cluster optimization
         trainer.crossval_optimize_params(Accuracy(), dataset, metrics=[Accuracy()], opt_evals=1)

In [12]: reporter = Report(trainer.get_best_results(), dataset, [Accuracy()])
         reporter.print_all_metric_results()
```

Рисунок 4.

Отличие вызова процесса оптимизации на локальной машине или на удаленном кластере заключается в передаваемом параметре client. Что дает удобный интерфейс для пользователей библиотеки modelgym, а также для тех кто использует эту библиотеку впервые.

В другом эксперименте было измерено 3 различные метрики для 4 алгоритмов классификации после оптимизации гиперпараметров с алгоритмом gp_minimize на одной и той же выборке. В таблице приведены значения <на локальной машине>/<в сервисе>.

	Accuracy	Roc_auc	F1-Score
LGBMClassifier	0.762 / 0.789	0.834 / 0.864	0.801 / 0.835
XGBClassifier	0.821 / 0.859	0.899 / 0.901	0.841 / 0.888

RFClassifier	0.795 / 0.832	0.827 / 0.852	0.784 / 0.801
CtBClassifier	0.831 / 0.868	0.914 / 0.947	0.847 / 0.889

Сравним различные алгоритмы оптимизации на конкретной выборке больных раком легких из библиотеки sklearn. Параметры задачи классификации:

1) Выборка о 569 наблюдениях

2) 30 признаков и 2 класса

3) Используем алгоритм LightGBM

4) Оптимизируемая метрика ROC_AUC

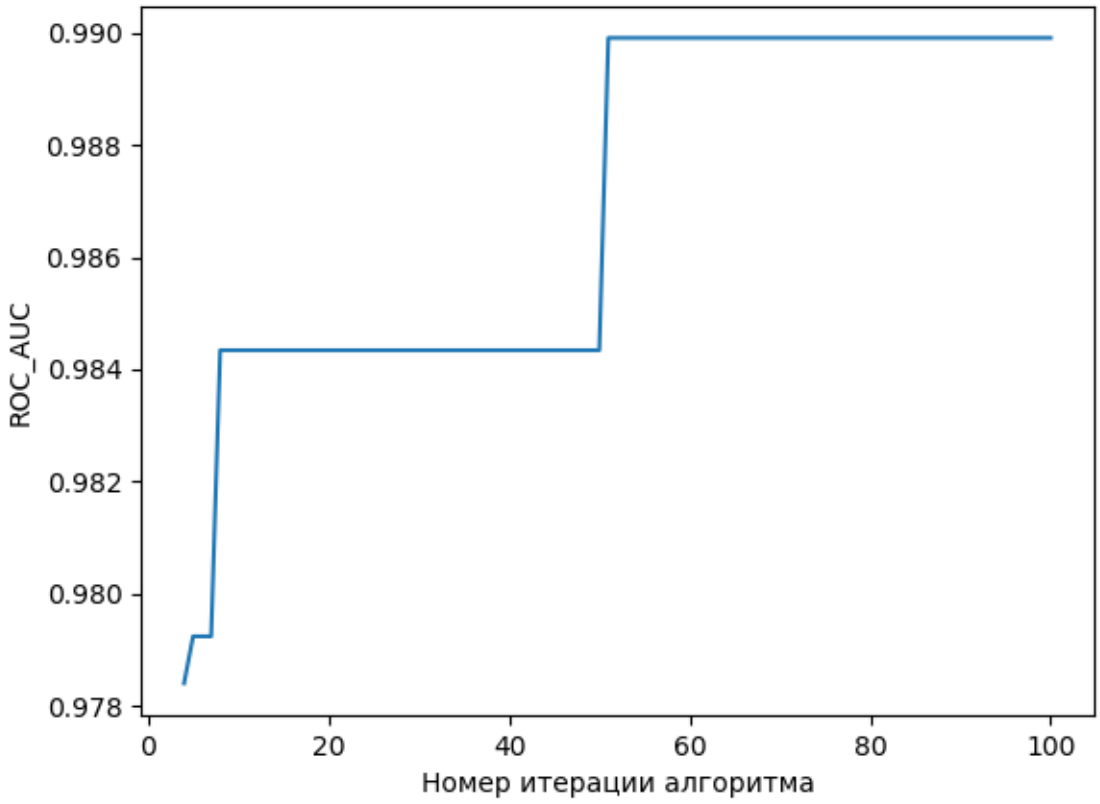
Рассмотрим 3 алгоритма оптимизации:

1) forest_minimize (skopt) — алгоритм последовательной оптимизации с использованием решающих деревьев

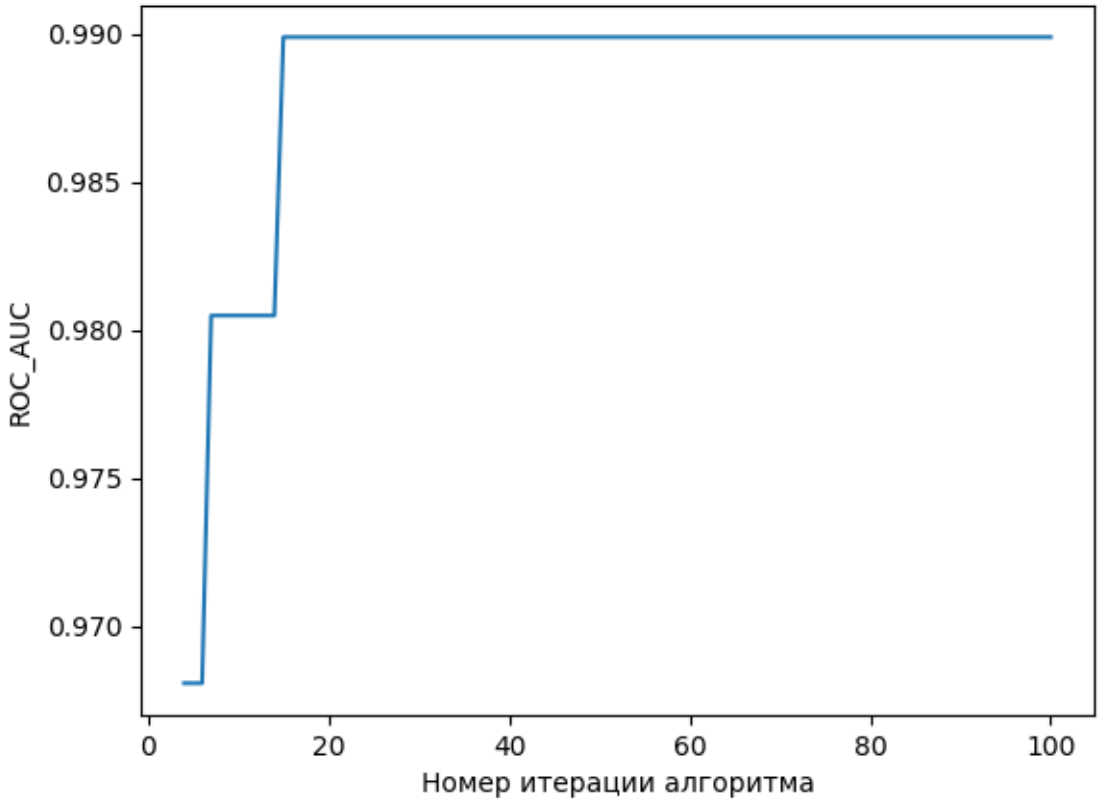
2) gp_minimize (skopt) — алгоритм байесовской оптимизации, использующий гауссовский процесс

3) tpe (hyperopt) — Tree-structured Paren Estimator

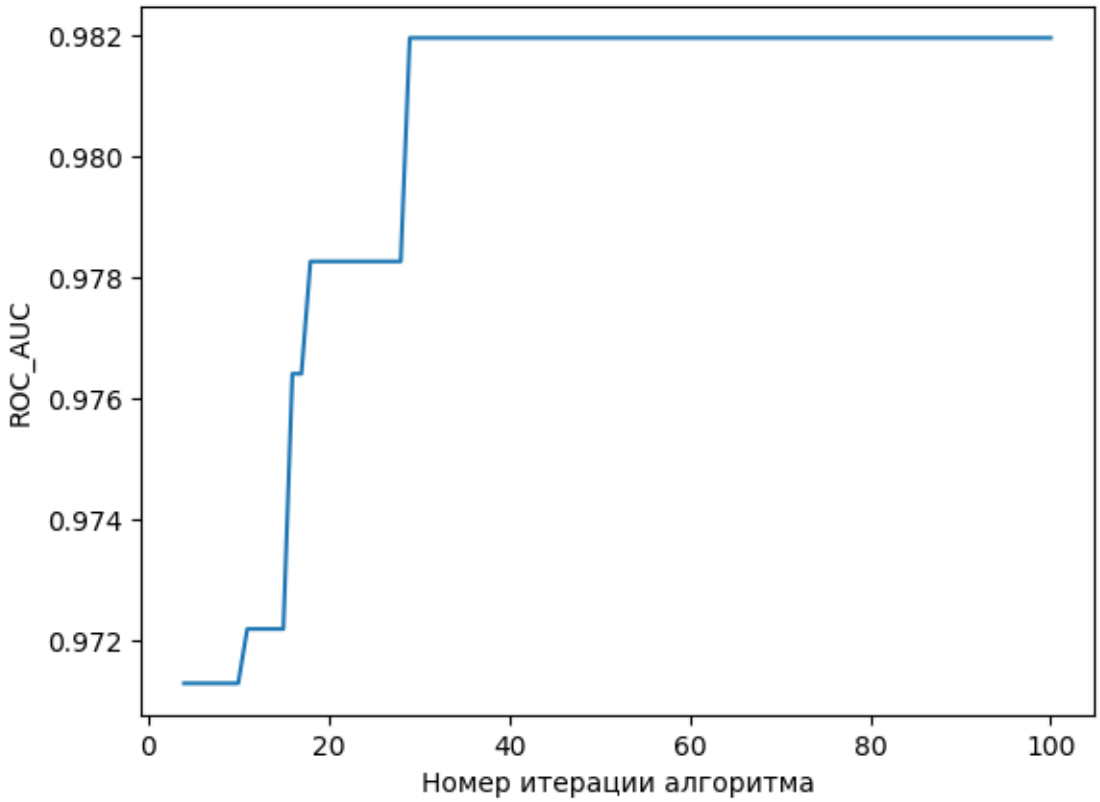
Каждый алгоритм выполнит цикл из 100 итераций. На каждой итерации будет обучена модель и выполнена кросс-проверка при разбиении данных на 3 части. На каждой из частей будет вычислена метрика ROC_AUC. Находится набор гиперпараметров на котором среднее значение данной метрики на 3 частях данных. На графиках представлены максимальные значения усредненной метрики к началу n-ой итерации алгоритма.



Процесс оптимизации алгоритма forest_minimize (skopt)



Процесс оптимизации алгоритма gp_minimize (skopt)



Процесс алгоритма оптимизации TPE (hyperopt)

Алгоритм оптимизации	Значение метрики ROC_AUC после 100 итераций
forest_minimize (skopt)	0,989915
gp_minimize (skopt)	0,989883
TPE(hyperopt)	0,981966

7.ЗАКЛЮЧЕНИЕ

Сервис «Wonderland» решает проблему поиска гиперпараметров эффективнее и в некоторых случаях способен находить оптимальную комбинацию гиперпараметров в разы быстрее программных средств, использующих только локальную машину. Он почти не использует ресурсов локальной машины,

перекладывая все вычисления на удаленный кластер и не замедляет компьютер исследователя. Алгоритмы используемые в оптимизации, эффективны для задач поиска оптимального набора гиперпараметров. Интерфейс «Wonderland» дополняет интерфейс библиотеки modelgym, что облегчает использование этого сервиса.

Среди возможных улучшений сервиса стоит выделить следующие:

1) Разработка своих алгоритмов оптимизации под конкретные задачи машинного обучение и анализа данных.

2) Добавление своего планировщика в систему kubernetes для эффективной утилизации кластерных ресурсов.

3) Добавление возможности использования сервиса на других облачных ресурсах.

8. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Библиотека оптимизации hyperopt // https://github.com/hyperopt/hyperopt (дата обращения: 10.02.2018, режим доступа: свободный).

2. yandexdataschool/modelgym: Gym for predictive models // Github URL:
3. Project Jupyter // URL: http://jupyter.org (дата обращения: 20.02.2015, режим доступа: свободный).

4. Kubernetes | Production-Grade Container Orchestration [Электронный ресурс] // URL: https://kubernetes.io/ (дата обращения: 12.07.2017, режим доступа: свободный).
5. A high performance, open-source universal RPC framework [Электронный ресурс] // URL: https://grpc.io (дата обращения: 12.07.2017, режим доступа: свободный).
6. Python 3.6.5 documentation // URL: https://docs.python.org/3/ (дата обращения: 30.03.2018, режим доступа: свободный).

7. Distributed Asynchronous Hyperparameter Optimization in Python // Github URL: https://github.com/hyperopt/hyperopt (дата обращения: 30.03.2018, режим доступа: свободный)
8. The Go Programming Language // URL: https://golang.org (дата обращения: 30.03.2018, режим доступа: свободный).
9. Protocol Buffers | Google Developers // URL: https://developers.google.com/protocol-buffers/ (дата обращения: 30.03.2018, режим доступа: свободный).
10. PostgreSQL: The world's most advanced open source database // URL: https://www.postgresql.org/ (дата обращения: 10.04.2018, режим доступа: свободный).
11. kubernetes/client-go: Go client for Kubernetes // GitHub URL: https://github.com/kubernetes/client-go (дата обращения: 10.04.2018, режим доступа: свободный).
12. TLS // Википедия URL: https://ru.wikipedia.org/wiki/TLS (дата обращения: 10.04.2018, режим доступа: свободный).
13. Installation/SystemRequirements [Электронный ресурс] // URL: https://help.ubuntu.com/community/Installation/SystemRequirements (дата обращения: 10.11.2017, режим доступа: свободный).
14. Minimum hardware requirement to run Docker - General Discussions - Docker Forums [Электронный ресурс] // URL: https://forums.docker.com/t/minimum-hardware-requirement-to-run-docker/28072 (дата обращения: 10.11.2017, режим доступа: свободный).
15. Pachyderm - Scalable, Reproducible Data Science // URL: http://www.pachyderm.io/ (дата обращения: 10.04.2018, режим доступа: свободный).
16. Ansible Documentation // Ansible Documentation URL: http://docs.ansible.com/ansible/latest/index.html (дата обращения: 10.04.2018, режим доступа: свободный).
17. Azure Container Service Documentation - Tutorials, API Reference // URL: https://docs.microsoft.com/en-us/azure/container-service/ (дата обращения: 30.03.2018, режим доступа: свободный).
18. Алгоритм оптимизации, используемый в skopt // URL:

19. Azure File Storage // URL: https://azure.microsoft.com/en-us/services/storage/files/ (дата обращения: 12.12.2017, режим доступа: свободный).
20. CWL // URL: https://www.commonwl.org/ (дата обращения: 12.10.2017, режим доступа: свободный).
21. Документация Docker // URL: https://docs.docker.com/ (дата обращения: 12.09.2017, режим доступа: свободный).
22. Документация библиотеки skopt // URL: / (дата обращения: 14.02.2018, режим доступа: свободный).