

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

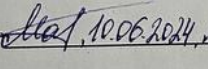
на тему

ИИФРОВОЙ ПОМОЩНИК ПЕДАГОГА

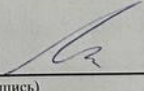
Студент  27.05.2024 Семенов Андрей Сергеевич
(подпись, дата) (ФИО полностью)

Руководитель  27.05.2024 Синев М. П.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр  27.05.2024 Бычков А. С.
(подпись, дата) (фамилия, инициалы)

Репензент  10.06.2024 Мартыненко А.М. зав. кафе. Программирования
(подпись, дата) (фамилия, инициалы) (должность, место работы) *Левисту*

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 № 11)

Заведующий кафедрой  Митрохин М. А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой *отлично* (протокол заседания ГЭК от 14.06.24 № 16)

Секретарь ГЭК  Гурин Е. И.
(подпись) (фамилия, инициалы)

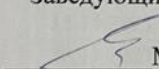
Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой

 М. А. Митрохин
«13» сентября 2022г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

1. Студент Семенов Андрей Сергеевич гр. 22ВВИм1 факультета
вычислительной техники, направления 09.04.01.

2. Тема работы «Цифровой помощник педагога».

Тема утверждена приказом ПГУ № 765/о от "13" сентября 2022 г.

3. Руководитель магистерской диссертации к.т.н., доцент кафедры ВТ ПГУ
Синев М. .

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: создание интеллектуальной системы
предсказания успеваемости у

Разработать автоматизированное клиент-серверное программное
обеспечение с использованием искусственного интеллекта для предоставления
информации по заданным датасетам.

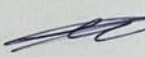
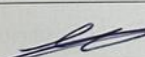
5. Перечень подлежащих разработке вопросов:

1) Анализ уже действующих решений из сферы цифровых помощников
педагога.

2) Проектирование и обучение нейронной сети.

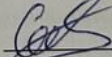
- 3) Разработка микросервиса для интеграции модели нейронной сети.
- 4) Тестирование согласно ранее описанному техническому заданию на разработку.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучить теоретические основы	10%	27.09.2022-15.11.2023	
2.	Анализ технических заданий, анализ действующих решений из сферы цифровых помощников педагога	10%	16.11.2023-22.03.2023	
3.	Выбор инструментов для разработки и утверждение функционала системы	5%	23.03.2023-14.06.2023	
4.	Спроектировать и обучить нейронную сеть	30%	15.06.2023-10.10.2023	
5.	Разработать серверную часть	20%	11.10.2023-05.01.2024	
6.	Интегрировать модель в разработанный микросервис	15%	06.01.2024-17.04.2024	
7.	Апробация разработанного серверного приложения в форме тестирования	5%	18.04.2024-9.05.2024	
8.	Оформление выпускной работы	5%	10.05.2024-28.05.2024	

Дата выдачи задания "13" сентября 2022 г.

Руководитель магистерской диссертации  13.09.2022 Синева М.П.
(подпись, дата)

Задание к исполнению принял студент  13.09.2022 Семенов А. С.
(подпись, дата)

Работу к защите допустить
Декан факультета  11.06.2024 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 96 листов, 27 рисунков, 5 источников и 2 приложения.

НЕЙРОННЫЕ СЕТИ, PYTORCH, PYTHON, DJANGO.

Цель работы – проектирование и обучение модели нейронной сети для предсказания успеваемости обучающихся с последующей интеграцией в микросервис для предоставления возможности взаимодействия с моделью.

Необходимыми задачами для достижения цели являются: анализ актуальности проекта; анализ потребностей пользователей; анализ аналогов; анализ типов нейронных сетей; изучение современных инструменты разработки нейронных сетей; проектирование и обучение нейронную сети; разработка кодовой базы микросервиса, который будет использовать обученную модель нейронной сети; тестирование работы созданного микросервиса.

Объектами разработки являются: нейронная сеть для предсказания успеваемости обучающихся и микросервис для предоставления способа взаимодействия с моделью.

В процессе работы были выполнены все поставленные задачи. Были проанализированы: актуальность проекта; потребности пользователей, аналоги, типы нейронных сетей. Были изучены современные инструменты разработки нейронных сетей и для создания микросервисов. В результате на языке Python использованием фреймворка PyTorch была разработана и обучена нейронная сеть, которая способна выдавать предсказание с точностью в 84%. Данная нейросеть была интегрирована в микросервис, написанный с помощью фреймворка Django.

					ПГУ 1.090401.32.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Цифровой помощник педагога Пояснительная записка	Лит.	Лист	Листов
Разраб.		Семенов А.С.		28.05.23				
Провер.		Синев М.П.		28.05.23			4	96
Н. Контр		Бычков А.С.		28.05.23		ФВТ, каф. ВТ		
Утверд.		Митрохин М.А.		28.05.23		гр. 22ВВим1		

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	6
ВВЕДЕНИЕ	7
1 Анализ предметной области.....	9
1.1 Анализ актуальности проекта	9
1.2 Анализ потребностей пользователей	10
1.3 Анализ аналогов	11
1.4 Анализ типов нейронных сетей	13
2 Реализация.....	16
2.1 Проектирование и обучение нейронной сети	17
2.1.1 Постановка задачи.....	17
2.1.2 Выбор инструментов разработки	18
2.1.3 Подготовка данных, проектирование и обучение нейронной сети	20
2.2 Разработка серверной части	35
2.2.1 Постановка задачи.....	35
2.2.2 Выбор инструментов разработки	38
2.2.3 Реализация серверной части	44
3 Описание результатов.....	52
ЗАКЛЮЧЕНИЕ	64
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	66

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

МЭШ- московская электронная школа. Интернет-платформа, которая состоит из электронного дневника и журнала, библиотеки, а также портфолио

API (Application Programming Interface) – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой

CNN (Convolutional Neural Network) – сверточная нейронная сеть. Архитектура нейронной сети, которая нацеленна на эффективное распознавание образов

JSON (JavaScript Object Notation) – текстовый формат обмена данными

ReLU (Rectified Linear Unit) – широко используемая функция активации нейронов в нейронных сетях

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

RNN (Recurrent Neural Network) – рекуррентная нейронная сеть. Архитектура нейронной сети, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки

UML (Unified Modeling Language) – унифицированный язык моделирования

ВВЕДЕНИЕ

Нейронные сети — это мощный инструмент искусственного интеллекта, способный анализировать и обрабатывать большие объемы данных, выявлять закономерности и делать прогнозы. В последние годы они стали неотъемлемой частью повседневной жизни человека. Нейронные сети нашли широкое применение в различных ее сферах, начиная от медицины и технологий, и заканчивая образованием.

Нейронные сети используются в образовании для улучшения качества обучения и повышения эффективности учебного процесса. Они помогают учителям проверять работы учеников, планировать учебную программу и многое другое.

Нейронные сети имеют огромный потенциал, как инструмент облегчения повседневных задач, которые ставятся перед преподавателем.

Целями данной диссертации является:

- изучение современных инструментов для создания и обучения моделей нейронных сетей,
- проектирование и обучение модели нейронной сети,
- изучение современных инструментов для создания микросервисов,
- создание микросервиса, который будет использовать обученную модель для предсказания успеваемости обучающихся.

Для достижения поставленных целей необходимо:

- произвести анализ актуальности работы,
- произвести анализ потребностей пользователей,
- произвести анализ аналогов,
- провести анализ типов нейронных сетей,
- изучить современные инструменты разработки нейронных сетей,
- спроектировать нейронную сеть и обучить ее,

- разработать кодовую базу микросервиса, который будет использовать обученную модель нейронной сети,
- протестировать работу созданного микросервиса.

1 Анализ предметной области

1.1 Анализ актуальности работы

В современных реалиях все школы России так или иначе пользуются различными электронными ресурсами в процессе ведения учебной деятельности, например электронными журналами. Учитель ежедневно вынужден ставить перед собой множество задач, обрабатывать огромное количество различной информации, связанной учениками и учебным процессом в целом. Это невероятно сложно, потому наблюдается тенденция развития информационных технологий, которые позволяют снизить нагрузку преподавателей.

Например, с сентября 2018 года все Московские школы активно пользуются высокотехнологичной образовательной средой “Московская электронная школа” (МЭШ), которая позволяет учителям смотреть расписание занятий, вести учет успеваемости обучающихся, задавать домашнее задание и многое другое.

Также уже с первого января 2023 года все школы России обязаны использовать в образовательном процессе федеральную государственную информационную систему “Моя школа”, которая схожа по функционалу с МЭШ [1]. Это регламентировано Федеральным законом от 30.12.2021 г. № 472-ФЗ О внесении изменений в Федеральный закон «Об образовании в Российской Федерации», а именно пунктом 3, подпунктом б, в котором сказано, что в Федеральном законе от 29 декабря 2012 года № 273-ФЗ "Об образовании в Российской Федерации" необходимо статью 16 дополнить частью 31 следующего содержания: "31. При реализации основных общеобразовательных программ и образовательных программ среднего профессионального образования с применением электронного обучения, дистанционных образовательных технологий, предусматривающих обработку персональных данных обучающихся, организация, осуществляющая образовательную деятельность, должна использовать государственные информационные

системы, создаваемые, модернизируемые и эксплуатируемые для реализации указанных образовательных программ." [2].

На этапе знакомства с новыми учениками, учителю сложно оценить потенциал обучающихся. Это необходимо для того, чтобы уделить больше внимания “слабым” ученикам в процессе обучения. Было бы полезно иметь инструмент, который на основании известных признаков мог бы предположить итоговую успеваемость обучающихся.

На основании всего написанного ранее можно сделать вывод, что данная диссертации является актуальной.

1.2 Анализ потребностей пользователей

Целевой аудиторией предполагаемой работы являются школьные учителя. У них много обязанностей связанных, как непосредственно с учебной деятельностью, так и с различными организационными моментами. К стандартным задачам учителя относится:

- проведение уроков,
- проверка различных работ обучающихся и выставление оценок.

Но, помимо этого, хороший учитель должен следить за успеваемостью учеников. Учителя люди, а не компьютеры, им сложно все удержать в голове, потому есть риск забыть что-то важное или не заметить “просадку” в знаниях у ученика или класса и впоследствии испортить статистику успеваемости в школе.

Предполагаемая работа «Цифровой помощник преподавателя» должен помочь справиться с описанной ранее проблемой. Он должен на основании доступных фактов об учениках предсказывать их итоговую успеваемость. Это позволит учителям планировать учебную деятельность таким образом, чтобы уделять больше внимание «слабым» ученикам.

Современный учитель активно использует в своей профессиональной деятельности различные электронные устройства (смартфон и компьютер),

потому для удобства необходимо обеспечить доступ к цифровому помощнику преподавателя со всех перечисленных устройств.

1.3 Анализ аналогов

В рамках работы над данной научно-исследовательской работой было рассмотрено несколько существующих проектов, которые позиционируют себя, как цифровой помощник учителя.

Одним из самых популярных приложений на площадке Google Play Маркет является «Помощник учителя Журнал оценок». Снимок экрана данного приложения представлен на рисунке 1.

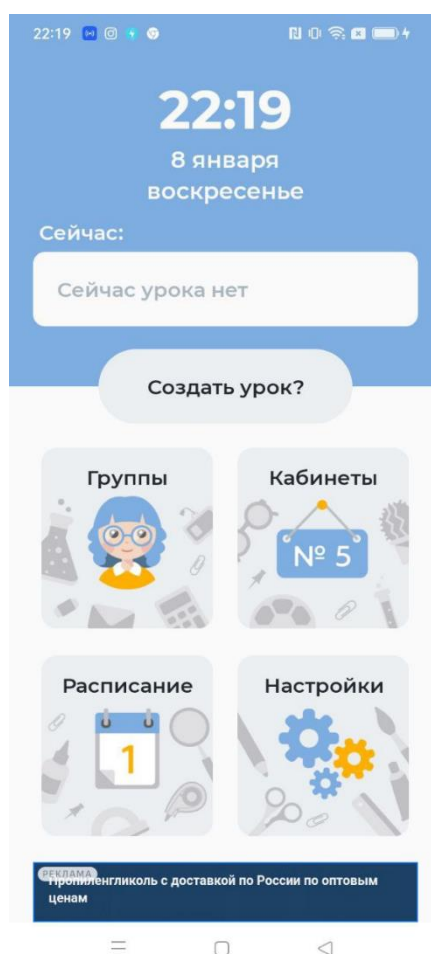


Рисунок 1 – Приложение «Помощник учителя Журнал оценок»

Проанализировав данное приложение, были выделены следующие преимущества и недостатки.

Преимущества:

– интуитивно понятный графический интерфейс, который позволяет легко взаимодействовать с приложением,

– оригинальный способ выставления оценок: помимо стандартного экрана с журналом урока есть экран, на котором абстрактно изображены парты и ученики, которые занимают места за ними. При нажатии на имя ученика увеличивается его оценка за урок (от единицы до пятерки).

Недостатком является скудный функционал. С практической точки зрения он состоит из просмотра расписания, выставления оценок и вычисления среднего балла ученика.

Далее было рассмотрено приложение «Журнал МЭШ». Его используют во всех общеобразовательных заведениях города Москвы. Снимок экрана данного приложения представлен на рисунке 2.

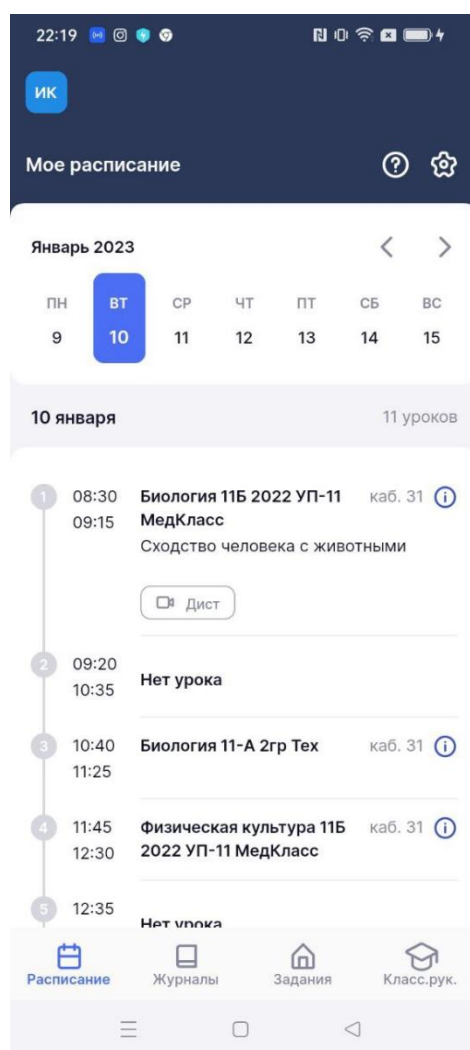


Рисунок 2 – Приложение «Журнал МЭШ»

В ходе анализа данного приложения были выявлены следующие преимущества и недостатки.

Преимущества:

- интуитивно понятный графический интерфейс, который позволяет легко взаимодействовать с приложением,
- богатый функционал, который позволяет просматривать расписание, вести учет успеваемости студентов, выдавать домашнее задание. Есть отдельный раздел для классного руководителя, где можно просматривать расписание подопечного класса и вести журнал пропусков учениками занятий,
- доступ к приложению защищается паролем или биометрией (сканер отпечатка пальца или face id).

Недостатков не выявлено.

Рассмотренные ранее приложения не обладают функционалом, который позволил бы предсказывать успеваемость учеников.

1.4 Анализ типов нейронных сетей

Существует огромное множество типов нейронных сетей. Самыми популярными являются:

- нейронные сети прямого распределения,
- сверточные нейронные сети,
- рекуррентные нейронные сети.

Сети прямого распределения или многослойные перцептроны (MLP)- это один из самых простых типов нейронных сетей. Эти нейросети состоят из одного входного слоя, одного или нескольких скрытых слоев и выходного слоя. Каждый нейрон в одном слое соединен с каждым нейроном в следующем слое. Сети прямого распространения обучаются с использованием методов градиентного спуска, таких как обратное распространение ошибки (backpropagation). Во время обучения сеть корректирует веса связей между нейронами, чтобы минимизировать ошибку прогнозирования. В основном

данный тип нейронных сетей используется для решения задач классификации и регрессии [3].

При обучении нейронной сети прямого распределения частыми проблемами является:

- переобучение— это когда нейронная сеть хорошо классифицирует данные их обучающей выборки, но теряет способность к обобщению (плохо классифицирует другие данные),

- затухание градиентов— это явление, при котором градиенты ошибки, применяемые для корректировки весов сети, становятся малы настолько, что процесс обучения становится крайне медленным или останавливается вовсе.

Сверточные нейронные сети (CNN) — это тип нейронных сетей, специально разработанных для обработки и анализа данных с пространственной структурой, таких как изображения и видео. В сверточных нейронных сетях используются сверточные слои, которые применяют фильтры (ядра) к небольшим областям входных данных для извлечения признаков. Это позволяет сети автоматически изучать локальные шаблоны и характеристики во входных данных. После сверточных слоев часто добавляют пулинг слои, которые уменьшают размер данных путем объединения регионов или усреднения значений. Это помогает уменьшить вычислительную сложность и сделать признаки более инвариантными к масштабу и смещению. Также включают слои активации, такие как ReLU (Rectified Linear Unit), которые помогают введении нелинейности в модель и улучшают скорость обучения. В основном данный тип нейронных сетей применяются в задачах компьютерного зрения, распознавания образов и анализа изображений.

При обучении сверточных нейронных сетей частыми проблемами является:

- переобучение,
- чувствительность к шуму.

Рекуррентные нейронные сети (RNN)– это тип нейронных сетей специально разработанный для работы с последовательными или временными данными, такими как текст. В рекуррентных нейронных сетях есть обратные связи, которые направлены от выхода нейронов к их входам. Они позволяют передавать информацию о предыдущих состояниях сети в текущий момент времени. Это помогает учитывать контекст и последовательность данных. Обучение данного типа нейронной сети происходит с помощью алгоритма обратного распределения по времени (back propagation through time).

При обучении рекуррентных нейронных сетей частыми проблемами является:

- затухание градиентов,
- взрыв градиентов – это явление, при котором градиенты ошибки, применяемые для корректировки весов сети, становятся велики на столько, что теряется стабильность в процессе обучения.

2 Реализация

Реализация данной работы будет поделена на две части:

- 1) проектирование и обучение нейронной сети,
- 2) разработка серверной части (микросервиса).

Предполагается, что пользователь с помощью клиента должен иметь возможность запросить у сервера список учебных классов и учеников по каждому из классов с предсказанием их успеваемости. Предсказание должна делать обученная нейронная сеть. Наглядно предполагаемое взаимодействие пользователя, клиента и сервера представлено на UML- диаграмме последовательности на рисунке 3.

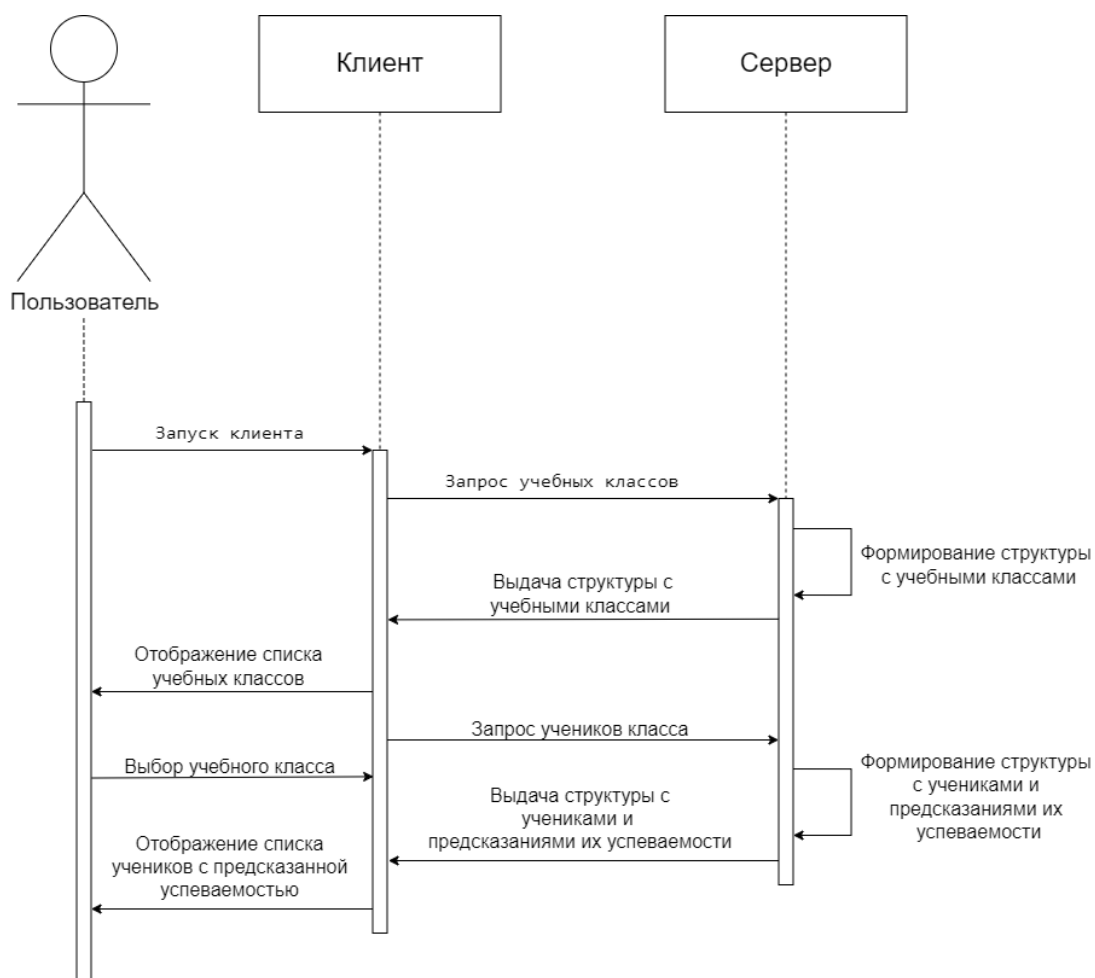


Рисунок 3 – UML-диаграмма последовательности взаимодействия клиента и сервера

2.1 Проектирование и обучение нейронной сети

Главной целью данной диссертации является разработка и обучение нейронной сети для предсказания итоговой успеваемости обучающихся. Она будет интегрирована в микросервис, разработку стоит начать именно с нее.

2.1.1 Постановка задачи

Для проектирования нейронной сети необходимо подобрать данные. Дата сет должен предоставить обширный набор наблюдений и признаков для наиболее качественного обучения нейронной сети.

Данные должны быть подвергнуты обработке для повышения качества обучения. А именно следует:

- закодировать все атрибуты типа Object, как номера класса,
- закодировать все небинарные категориальные переменные с помощью алгоритма частотно кодировки,
- не категориальные переменные нормализовать методом Мин-макс нормализация,
- устранить несбалансированность данных методом *upsampling data*.

На основании анализа типов нейронных сетей было принято решение разрабатывать и обучать нейронную сеть прямого распределения. Этому способствовали следующие причины:

- простота реализации: сети прямого распространения являются одним из самых простых типов нейронных сетей и легко реализуются на практике,
- эффективность: сети прямого распространения показывают хорошие результаты в классификации объектов по признакам, особенно когда речь идет о больших объемах данных,
- простой анализ результатов: сети прямого распространения позволяют легко интерпретировать результаты классификации и понять, какой признак оказался решающим для принятия решения,
- возможность использования различных функций активации: с помощью сетей прямого распространения можно использовать различные

функции активации, что позволяет лучше моделировать сложные зависимости между признаками объектов.

Для обучения нейронной сети необходимо разработать алгоритм подбора гипер- параметров без участия человека, чтобы процесс обучения не требовал постоянного наблюдения и вмешательства. Наиболее точная модель должна сохраняться в файл до последующего использования в микросервисе.

2.1.2 Выбор инструментов разработки

Изначально необходимо выбрать язык программирования для разработки нейронной сети.

Чаще всего для разработки нейронных сетей используется язык программирования Python. Этому способствуют следующие причины:

- python является высокоуровневым языком программирования,
- простой и лаконичный синтаксис,
- большой набор библиотек и инструментов для работы с нейронными сетями,
- широкое сообщество разработчиков.

Для разработки данной работы так же будет использоваться язык программирования Python.

После необходимо выбрать фреймворк для разработки нейронной сети.

Современные инструменты для разработки нейронных сетей обладают множеством функций и возможностей, которые помогают ускорить и упростить процесс создания и обучения нейронных сетей.

Наиболее популярными фреймворками являются:

- tensorflow,
- keras,
- caffe,
- pyTorch.

Каждый из них имеет свои преимущества и недостатки.

TensorFlow— это один из самых популярных фреймворков для машинного обучения, который разработан командой Google Brain. Возможности данного фреймворка позволяют гибко настраивать модель и эффективно обучать ее. Многие пользователи данного фреймворка отмечают сложность его API, что можно отнести к минусам данного решения.

Keras — это фреймворк, построенный поверх TensorFlow. Главным его преимуществом является высокоуровневое API которое делает разработку нейронных сетей простой и интуитивно понятной. Но главное преимущество данного фреймворка так же является его главным недостатком, ведь высокоуровневость API данного фреймворка ограничивает гибкость настройки модели.

Caffe — это библиотека для разработки нейронных сетей, особенно популярная в области компьютерного зрения. Она обладает достаточно высокой скоростью обучения и наиболее эффективна в работе с изображениями. Недостатком данного фреймворка является его наименьшая популярность в сравнении с остальными.

PyTorch — это еще один популярный фреймворк для разработки нейронных сетей, разработанная командой Facebook AI Research. Он предлагает гибкий и удобный интерфейс для создания и обучения моделей глубокого обучения. Данный фреймворк использует динамический вычислительный граф, что позволяет легко изменять структуру и параметры модели в процессе обучения, в то время как TensorFlow использует статический вычислительный граф, который может быть менее гибким. При всем этом PyTorch обладает простым и интуитивно понятным API.

На основании всего вышеперечисленного для разработки данной нейронной сети будет использоваться фреймворк PyTorch .

Так же для обработки данных необходимы дополнительные библиотеки:

– numpy — пакет для выполнения научных вычислений,

– pandas – инструмент для эффективной работы с таблицами данных, который может быть использован для организации процесса обучения искусственных нейронных сетей,

– sklearn – библиотека машинного обучения, включающая в себя реализацию различных алгоритмов обучения с учителем.

2.1.3 Подготовка данных, проектирование и обучение нейронной сети

Для обучения модели был выбран датасет Student Performance. Эти данные отражают успеваемость учащихся в средней школе двух португальских школ. Атрибуты данных включают оценки учащихся, демографические данные, социальные и школьные характеристики) и были собраны с использованием школьных отчетов и анкет.

Датасет содержит следующие атрибуты:

- school– школа студента (в данном случае признак бинарный, так как данные собирались с двух школы. В будущем количество можно увеличить),
- sex– пол,
- age– возраст студента (от 15 до 22),
- adress– адрес студента городской или сельский,
- famsize– размер семьи (бинарный признак: «LE3» — меньше или равно 3 или «GT3» — больше 3),
- pstatus– статус совместного проживания родителей (бинарный признак: вместе или отдельно),
- meddu– образование матери (0 – нет, 1 – начальное образование (4 класс), 2 – с 5 по 9 классы, 3 – среднее образование, 4 – высшее образование),
- fedu– образование отца (0 – нет, 1 – начальное образование (4 класс), 2 – с 5 по 9 классы, 3 – среднее образование, 4 – высшее образование),
- mjob– профессия матери,
- fjob– профессия отца,
- reason– причина выбрать эту школу (близкая к «дому», «репутация» школы, предпочтение «курса» или «другое»),

- guardian– опекун учащегося («мать», «отец» или «другое»),
- traveltime– время в пути от дома до школы (числовое: 1 – <15 минут, 2 – 15 – 30 минут, 3 – 30 минут – 1 час или 4 – >1 часа),
- studytime– еженедельное время обучения (числовое: 1–<2 часов, 2–2–5 часов, 3–5–10 часов или 4–>10 часов),
- failures– количество не сданных в прошлом экзаменов по предметам (n если $1 \leq n < 3$, иначе 4),
- schoolsup– дополнительная образовательная поддержка (да или нет),
- famsu– помощь семьи в обучении (да или нет),
- paid– дополнительные платные занятия (да или нет),,
- activities– внеклассная деятельность (да или нет)
- nursery– посещал ли ученик детский сад (да или нет),
- higher– хочет ли получить высшее образование (да или нет),
- internet– есть ли доступ к интернету дома (да или нет),
- romantic– состоит ли в романтических отношениях (да или нет),
- famrel– отношения в семье (оценка от 1 до 5),
- freetime– сколько свободного времени после школы (оценка от 1 до 5),
- goout– как часто гуляет с друзьями (оценка от 1 до 5),
- dalc– частота употребления алкоголя в рабочие дни (оценка от 1 до 5),
- walc– частота употребления алкоголя в выходные дни (оценка от 1 до 5),
- helth– состояние здоровья (оценка от 1 до 5),
- absence– количество пропусков занятий,
- G1– оценка за первый период,
- G2– оценка за второй период,
- G3– итоговая годовая оценка.

Целевым атрибутом является G3. В описании датасета сказано, что атрибуты G1 и G2 имеют высокую коллинеарность с целевым атрибутом. С

ними модель было бы обучить проще, но для задачи классификации учеников по успеваемости на этапе знакомства мне это не подходит, поэтому я не буду их учитывать.

Перед обучением данные нужно подготовить. Изначальное состояние датасета представлено на рисунке 4.

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	4	0	11	11
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	2	9	11	11
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	6	12	13	12
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	0	14	14	14
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	0	11	13	13

Рисунок 4 – Исходный датасет

Делю столбец G3 на классы:

```
df['performance'] = pd.cut(df['G3'], bins=[-1, 5, 10, 15, 20],
labels=False)
```

- 0– плохая успеваемость (итоговый балл от 0 до 5)
- 1– удовлетворительная успеваемость (итоговый балл от 5 до 10)
- 2– хорошая успеваемость (итоговый балл от 10 до 15)
- 3– отличная успеваемость (итоговый балл от 15 до 20)

Далее я закодировал все атрибуты типа object, как номера класса

```
object_cols = df.select_dtypes(include='object').columns
df[object_cols] = df[object_cols].apply(lambda x:
pd.Categorical(x).codes)
```

Итоговый вид датасета представлен на рисунке 5.

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3	performance
0	0	0	18	1	0	0	4	4	0	4	...	3	4	1	1	3	4	0	11	11	2
1	0	0	17	1	0	1	1	1	0	2	...	3	3	1	1	3	2	9	11	11	2
2	0	0	15	1	1	1	1	1	0	2	...	3	2	2	3	3	6	12	13	12	2
3	0	0	15	1	0	1	4	2	1	3	...	2	2	1	1	5	0	14	14	14	2
4	0	0	16	1	0	1	3	3	2	2	...	3	2	1	2	5	0	11	13	13	2

Рисунок 5 – Датасет после кодирования атрибутов типа object

Все атрибуты имеют тип integer, что можно увидеть на рисунке 6.

Далее я взял все не бинарные категориальные признаки и закодировал их по алгоритму частотной кодировки (Frequency Encoding). Данный метод

заключается в том, что каждый категориальный признак кодируется числом, которое равно количеству этого признака в датасете:

```
not_binary_object_cols = [col for col in object_cols if
len(df[col].unique()) > 2]
not_binary_object_cols.extend(['Medu', 'Fedu', 'traveltime',
'studytime', 'failures', 'famrel',
                                'freetime', 'goout',
'Dalc', 'Walc', 'health'])
not_binary_object_cols.extend(['Medu', 'Fedu', 'traveltime',
'studytime', 'failures', 'famrel',
                                'freetime', 'goout',
'Dalc', 'Walc', 'health'])
df.loc[:, not_binary_object_cols] =
__frequency_encoding(df.loc[:, not_binary_object_cols])
```

Чтобы убедиться, что все категориальные признаки были закодированы, был вызван метод `df.info`, где можно увидеть, что все переменные имеют тип `integer`. Результат выполнения метода представлен на рисунке 6.

Реализация алгоритма метода частотной кодировки выглядит следующим образом:

```
def __frequency_encoding(df):
    new_df = df.copy()
    for c in df.columns:
        mapa = df[c].value_counts().to_dict()
        new_df[c] = df[c].map(mapa)
    return new_df
```

Он принимает на вход дата фрейм с не бинарными категориальными признаками. Затем в каждом столбце вычисляется количество каждого уникального элемента (частота) и сохраняется в словарь. После каждое значение в столбце заменяется на соответствующую ему частоту.

В результате дата фрейм имеет вид, который представлен на рисунке 7.

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 651 entries, 0 to 650
Data columns (total 34 columns):
#   Column                Non-Null Count  Dtype
---  -
0   school                651 non-null    int8
1   sex                   651 non-null    int8
2   age                   651 non-null    int64
3   address               651 non-null    int8
4   famsize               651 non-null    int8
5   Pstatus              651 non-null    int8
6   Medu                 651 non-null    int64
7   Fedu                 651 non-null    int64
8   Mjob                 651 non-null    int8
9   Fjob                 651 non-null    int8
10  reason               651 non-null    int8
11  guardian             651 non-null    int8
12  traveltime           651 non-null    int64
13  studytime            651 non-null    int64
14  failures             651 non-null    int64
15  schoolsup            651 non-null    int8
16  famsup              651 non-null    int8
17  paid                 651 non-null    int8
18  activities           651 non-null    int8
19  nursery             651 non-null    int8
20  higher               651 non-null    int8
21  internet             651 non-null    int8
22  romantic             651 non-null    int8
23  famrel              651 non-null    int64
24  freetime            651 non-null    int64
25  goout               651 non-null    int64
26  Dalc                651 non-null    int64
27  Walc                651 non-null    int64
28  health              651 non-null    int64
29  absences            651 non-null    int64
30  G1                  651 non-null    int64
31  G2                  651 non-null    int64
32  G3                  651 non-null    int64
33  performance         651 non-null    int64

```

Рисунок 6 – Вывод результата метода df.info

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3	performance
0	0	0	18	1	0	0	176	129	136	37	...	252	143	453	249	125	4	0	11	11	2
1	0	0	17	1	0	1	143	174	136	368	...	252	205	453	249	125	2	9	11	11	2
2	0	0	15	1	1	1	143	174	136	368	...	252	145	121	120	125	6	12	13	12	2
3	0	0	15	1	0	1	176	210	48	181	...	107	145	453	249	250	0	14	14	14	2
4	0	0	16	1	0	1	139	131	259	368	...	252	145	453	150	250	0	11	13	13	2

Рисунок 7 – Датасет после кодирования не бинарных категориальных признаков методом Frequency Encoding

Далее я нормализую не категориальные атрибуты типа integer методом Мин-макс нормализация (MinMax Scaling):

```

num_cols = df.select_dtypes(include='int64').columns
num_cols = num_cols.drop(labels=['Medu', 'Fedu',
'traveltime', 'studytime', 'failures', 'famrel', 'freetime',
'goout', 'Dalc', 'Walc', 'health', 'performance'])
for c in num_cols:
    df[c] = (df[c] - df[c].min()) / (df[c].max() - df[c].min())

```

В результате датасет имеет вид, который представлен на рисунке 8.

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Walc	health	absences	G1	G2
0	0	0	0.428571	1	0	0	176	129	0.417062	0.040580	...	252	143	453	249	125	0.1250	0.000000	0.578947
1	0	0	0.285714	1	0	1	143	174	0.417062	1.000000	...	252	205	453	249	125	0.0625	0.473684	0.578947
2	0	0	0.000000	1	1	1	143	174	0.417062	1.000000	...	252	145	121	120	125	0.1875	0.631579	0.684211
3	0	0	0.000000	1	0	1	176	210	0.000000	0.457971	...	107	145	453	249	250	0.0000	0.736842	0.736842
4	0	0	0.142857	1	0	1	139	131	1.000000	1.000000	...	252	145	453	150	250	0.0000	0.578947	0.684211

Рисунок 8 – Датасет после нормализации не категориальных атрибутов типа integer методом Мин-макс

С помощью метода `pandas.value_counts()`, можно понять, что количество учеников с плохой успеваемостью равно 17, с удовлетворительной 180, хорошей 372 и отличной 82. Результат функции представлен на рисунке 9.

```

2    372
1    180
3     82
0     17
Name: performance, dtype: int64

```

Рисунок 9 – Вывод результата метода `value_counts`

Это свидетельствует о несбалансированности данных. Для устранения несбалансированности был использован метод `upsampling data` [4]. Данный метод заключается в дублировании записей с классами, количество которых меньше в сравнении с классом с наибольшим количеством записей.

Дисбаланс идет в сторону класса под номером два. Чтобы сбалансировать изначально было вычислены соотношения второго класса к остальным:

```

rat21 = len(df.loc[df['performance'] == 2]) //
len(df.loc[df['performance'] == 1])
rat23 = len(df.loc[df['performance'] == 2]) //
len(df.loc[df['performance'] == 3])

```

```

rat20      =      len(df.loc[df['performance']      ==      2])      //
len(df.loc[df['performance'] == 0])

```

Затем количество записей этих классов было увеличено в `rat` раз, где `rat` это значение соотношения класса два к классу, количество которого увеличивается:

```

df_1 = df.loc[df['performance'] == 1]
df_1 = df_1.loc[df_1.index.repeat(rat21)]

df_3 = df.loc[df['performance'] == 3]
df_3 = df_3.loc[df_3.index.repeat(rat23)]

df_0 = df.loc[df['performance'] == 0]
df_0 = df_0.loc[df_0.index.repeat(rat20)]

df_new = pd.concat([df.loc[df['performance'] == 2],
df_1]).sample(frac=1)
df_new = pd.concat([df_new, df_3]).sample(frac=1)
df_new = pd.concat([df_new, df_0]).sample(frac=1)

```

Наконец, убираю ненужные атрибуты и перемешиваю строки:

```

df = df.drop(columns=['G1', 'G2', 'G3'])
df = df.iloc[np.random.permutation(len(df))]

```

Данные готовы к обучению модели.

Далее необходимо спроектировать модель. Для создания модели нейронной сети я использовал библиотеку PyTorch.

Мной был создан класс `StudentPerformanceNN`. Конструктор данного класса принимает следующие параметры:

- `input_size` – количество нейронов входного слоя,
- `hidden_size` – количество нейронов скрытого слоя,
- `hidden_layers_count` – количество скрытых слоев,
- `output_size` – количество нейронов выходного слоя,
- `learning_rate` – скорость обучения,

– epochs– количество эпох обучения.

Внутри конструктора происходит инициализация полей класса, модели нейронной сети. Создается модель с помощью метода `_build_model`, определяется функция потерь `CrossEntropyLoss` и оптимизатор `Adam` с переданной скоростью обучения. Так же определяется набор метрик, которые будут сниматься в процессе обучения модели:

```
def __init__(self, input_size, output_size,
hidden_layers_count, hidden_size, learning_rate, epochs):
    self.input_size = input_size
    self.output_size = output_size
    self.hidden_layers_count = hidden_layers_count
    self.hidden_size = hidden_size
    self.learning_rate = learning_rate
    self.epochs = epochs

    self.model = self._build_model()
    self.loss_fn = nn.CrossEntropyLoss()
    self.optimizer =
torch.optim.Adam(self.model.parameters(),
lr=self.learning_rate)

    self.metrics = {'loss': [], 'accuracy': [], 'recall':
[], 'precision': [], 'f1': []}
```

Внутри метода `_build_model` изначально определяются слои нейронной сети. Первым добавляется линейный слой (`nn.Linear`) с размерами входного слоя и скрытого слоя, за которым следует слой активации `ReLU` (`nn.ReLU`). Затем в цикле добавляются скрытые слои. Количество скрытых слоев определяется параметром `hidden_layers_count`. Каждый скрытый слой состоит из линейного слоя `nn.Linear` с размерами скрытого слоя и слоя активации `ReLU`. В конце добавляется последний линейный слой с размерами скрытого слоя и выходного слоя, за которым следует последний слой активации `ReLU`. Наконец,

создается модель нейронной сети с помощью класса `nn.Sequential()`, в конструктор которого передается список слоев:

```
def _build_model(self):
    layers = [nn.Linear(self.input_size,
self.hidden_size), nn.ReLU()]

    for _ in range(self.hidden_layers_count - 1):
        layers.append(nn.Linear(self.hidden_size,
self.hidden_size))
        layers.append(nn.ReLU())

    layers.append(nn.Linear(self.hidden_size,
self.output_size))

    model = nn.Sequential(*layers)

    return model
```

Для тренировки модели был создан метод `train`, который принимает следующие параметры:

- `train_data`– объект класса `torch.utils.data.dataloader.DataLoader`, который содержит данные для обучения модели,
- `test_data`– объект того же класса, который содержит данные для тестирования модели,
- `is_log`– параметр типа `Boolean`, от значения которого зависит будут ли в процессе обучения в консоль выводятся промежуточные значения снимаемых метрик (`accrasy`, `recall`, `precision`, `f1`, `loss`).

Внутри метода модель переводится в режим тренировки, затем запускается цикл, который прекратится, когда будет достигнуто заданное количество эпох. Внутри данного цикла запускается еще один цикл, который перебирает входные параметры и истинные классы тренировочных данных. Внутри данного цикла с начала обнуляются градиенты в оптимизаторе. Далее

предсказываются классы. Затем вычисляется значение функции потерь и значения градиентов функции потерь. Далее делается шаг оптимизации (обновляются параметры модели на основе вычисленных градиентов). Перед завершением эпохи обучения вызывается метод `_save_metrics`, который вычисляет и сохраняет значения метрик на данной эпохе обучения. Затем, если включено логирование (параметр `is_log = true`), вызывается метод `_print_last_metrics`, который в консоль выводит значение метрик на данной эпохе обучения:

```
def train(self, train_data, test_data, is_log):
    self.model.train()
    for epoch in range(self.epochs):

        for inputs, labels in train_data:
            self.optimizer.zero_grad()
            outputs = self.model(inputs)
            loss = self.loss_fn(outputs, labels)
            loss.backward()
            self.optimizer.step()

        self._save_metrics(test_data)

        if epoch % 10 == 0 and is_log:
            self._print_last_metrics(epoch)
```

Упомянутый ранее метод `_save_metrics` в качестве параметра принимает данные для тестирования нейронной сети. Внутри метода вызывается метод `test_model`, который используется для тестирования модели и снятия метрик, значения которых он возвращает. Полученные значения метрик сохраняются в поле класса `metrics`, который имеет тип `Map<String, double>`:

```
def _save_metrics(self, test_data):
    epoch_accuracy,    epoch_recall,    epoch_precision,
    epoch_f1, epoch_loss = self.test_model(test_data)
```

```
self.metrics['accuracy'].append(epoch_accuracy)
self.metrics['recall'].append(epoch_recall)
self.metrics['precision'].append(epoch_precision)
self.metrics['f1'].append(epoch_f1)
self.metrics['loss'].append(epoch_loss)
```

Другой упомянутый ранее метод `_print_last_metrics` в качестве параметра принимает номер текущей эпохи обучения. Внутри метода из поля класса `metrics` берутся последние снятые метрики и выводятся в консоль:

```
def _print_last_metrics(self, epoch):
    last_loss = self.metrics['loss'][-1]
    last_accuracy = self.metrics['accuracy'][-1]
    last_recall = self.metrics['recall'][-1]
    last_precision = self.metrics['precision'][-1]
    last_f1 = self.metrics['f1'][-1]

    print(f'\nEpoch: {epoch}')
    print(f'Accuracy: {last_accuracy}')
    print(f'Recall: {last_recall}')
    print(f'Precision: {last_precision}')
    print(f'F-1: {last_f1}')
    print(f'Loss: {last_loss}')
```

Упомянутый ранее метод `test_model` в качестве параметра принимает данные для тестирования. Внутри метода модель переводится в режим тестирования. На основании входных параметров из данных для тестирования нейронная сеть предсказывает класс успеваемости студентов. Затем на основании предсказаний и истинных классов вычисляются следующие метрики:

- accuracy – соотношение правильных предсказаний к общему числу предсказаний,
- recall – доля объектов положительного класса, которую определила нейронная сеть из всех объектов положительного класса,

– precision– доля правильно предсказанных положительных объектов среди всех объектов, которая нейронная сеть определила, как положительный класс,

– f1– среднее гармоническое recall и precision,

– loss– значение, которое выдает функция потерь. Она отражает разницу между предсказанными значениями модели и истинными значениями целевой переменной.

Затем модель снова переводится в режим обучения. Метод возвращает значения вычисленных метрик:

```
def test_model(self, test_data):
    self.model.eval()
    for inputs, labels in test_data:
        with torch.no_grad():
            outputs = self.model(inputs)

            _, predicted = torch.max(outputs.data, 1)

            total = labels.size(0)
            correct = (predicted == labels).sum().item()
            y_pred = predicted.cpu().numpy()
            y_true = labels.numpy()

            accuracy = correct / total
            recall = recall_score(y_true, y_pred,
average='macro')
            precision = precision_score(y_true, y_pred,
average='macro', zero_division=0)
            f1 = f1_score(y_true, y_pred,
average='macro')

            loss = self.loss_fn(outputs, labels).item()

    self.model.train()
    return accuracy, recall, precision, f1, loss
```

Для сохранения модели был добавлен метод `save_model`, который вызывает метод `_get_model_name` для определения имени модели. Модель сохраняется в папку `models` разрешением «.pt»:

```
def save_model(self):
    model_name = self._get_model_name()
    torch.save(self.model.state_dict(),
os.path.join('models', model_name + '.pt'))
```

Упомянутый ранее метод `_get_model_name` формирует и возвращает имя модели из значения метрик accuracy, f1, из чисел равных количеству эпох, количеству нейронов скрытого слоя, скорости обучения, количеству нейронов входного слоя, количеству нейронов выходного слоя:

```
def _get_model_name(self):
    last_accuracy = self.metrics['accuracy'][-1]
    last_f1 = self.metrics['f1'][-1]
    int_accuracy = int(last_accuracy * 1000)
    int_f1 = int(last_f1 * 1000)
    return
f'model_a{int_accuracy}_f1{int_f1}_e{self.epochs}_hs{self.hi
dden_size}_hlc{self.hidden_layers_count}' \
    f'_lr{round(self.learning_rate,
4)}_is{self.input_size}_os{self.output_size}'
```

Для построения и сохранения графиков изменения метрик относительно эпох обучения был реализован метод `plot_metrics`. Данный метод вызывает внутри себя пять методов, каждый из которых отвечает за отрисовку и сохранение графика определенной метрики:

```
def plot_metrics(self):
    self._plot_loss()
    self._plot_accuracy()
    self._plot_recall()
    self._plot_precision()
    self._plot_f1()
```

Упомянуты ранее методы похожи между собой каждый использует библиотеку `matplotlib` для построения графика и затем вызывает метод `_save_plot` для его сохранения. Один из таких методов представлен ниже:

```
def _plot_f1(self):
    plt.figure(figsize=(10, 5))
    plt.plot(self.metrics['f1'])
    plt.xlabel('Epoch')
    plt.ylabel('F-1')
    self._save_plot(plt, 'f1.jpg')
```

Метод `_save_plot` в качестве параметров принимает:

- `plt`– сформированный с помощью библиотеки `matplotlib` график;
- `metric_graph_name`– название файла, в который будет сохранен график.

Внутри метода для определения названия папки, куда будут сохраняться графики вызывается метод `_get_model_name`. Затем график сохраняется в каталоге `metrics/<Название модели>`:

```
def _save_plot(self, plt, metric_graph_name):
    model_name = self._get_model_name()

    if not os.path.exists(os.path.join('metrics',
model_name)):
        os.mkdir(os.path.join('metrics', model_name))
        plt.savefig(os.path.join('metrics', model_name,
metric_graph_name))
        plt.close()
```

Далее был разработан алгоритм автоматического подбора гиперпараметров, обучения модели и сохранения наиболее точной.

Процесс подбора гиперпараметров, обучения модели и сохранения наиболее точной был построен следующим образом:

- из датасета загружаются и обрабатываются данные (процесс обработки описан ранее). Далее они случайным образом делятся на

тренировочные и тестовые в соотношении 80 к 20. Для этого вызывается метод `get_data` класса `DataPrepare`;

– запускается бесконечный цикл `while`. В каждой итерации цикла:

- 1) случайным образом определяется количество нейронов скрытого слоя (от 10 до 50);
- 2) Далее случайным образом определяется количество скрытых слоев (от 10 до 70);
- 3) После случайным образом определяется скорость обучения (от 0.001 до 0.002);
- 4) Затем случайным образом определяется количество эпох обучения (от 5 до 90);
- 5) После создается экземпляр класса `StudentPerformanceNN` с заданными параметрами;
- 6) Далее модель обучается на обучающем наборе данных;
- 7) После обучения вычисляется `accuracy` и `f1` модели на тестовом наборе данных и, если значения этих метрик лучше, чем предыдущие наилучшие их значения, обученная модель сохраняется с соответствующими параметрами.

На каждой итерации также выводится текущую значения `accuracy` и `f1` и их наилучшие показатели за время выполнения цикла:

```
best_accuracy = -1
best_f1 = -1

print('\n\nPrepare data...')
train_data, test_data =
DataPrepare().get_data('../Portuguese.csv')

while True:
    hidden_size = rand.randint(10, 50)
    hidden_layers_count = rand.randint(10, 70)
    learning_rate = rand.uniform(0.001, 0.0017)
```

```

neural_network      =      StudentPerformanceNN2(input_size,
output_size, hidden_layers_count, hidden_size, learning_rate,
                                                    epochs)

print('\nStart train...')
print(
    f'epoch=      {epochs},      hidden_size=      {hidden_size},
hidden_layers_count={hidden_layers_count},      lr=
{learning_rate}')
neural_network.train(train_data, test_data, is_log=False)

print('\nTest model...')
accuracy,      recall,      precision,      f1,      loss      =
neural_network.test_model(test_data)

print(f'Accuracy on test set: {100 * accuracy:.2f}%')
print(f"F1-score on test set: {100 * f1:.2f}")
print(f'Best accuracy: {100 * best_accuracy:.2f}%')
print(f"Best f1-score: {100 * best_f1:.2f}")

if accuracy >= best_accuracy and f1 > best_f1:
    print('\nSave model...')
    best_accuracy = accuracy
    best_f1 = f1
    neural_network.save_model()
    neural_network.plot_metrics()

```

Наглядно процесс подбора гиперпараметров и обучения модели представлен в виде блок-схемы на рисунке 9.

2.2 Разработка серверной части

После проектирования и обучения нейронной сети необходимо разработать микросервис, в который она будет интегрирована и с которым будет взаимодействовать клиент.

2.2.1 Постановка задачи

Как было сказано ранее API сервера будет состоять из двух методов:

- 1) формирование и отправка списка учебных классов,
- 2) формирование и отправка списка учеников с предсказанной успеваемостью.

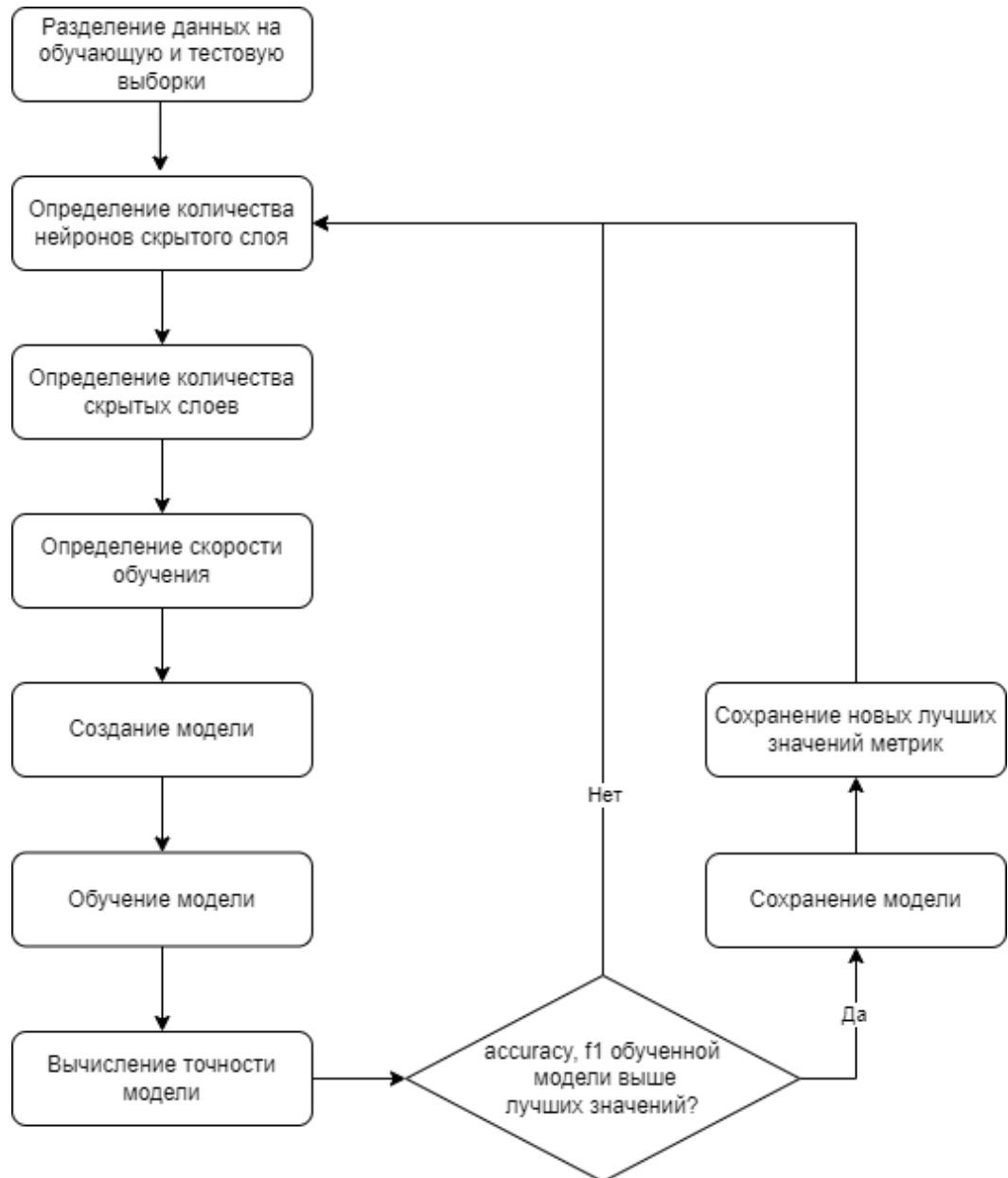


Рисунок 9 – Блок- схема процесс подбора гиперпараметров и обучения модели

Для реализации методов нужно продумать две точки доступа и структуры данных для обмена информацией между клиентом и сервером.

Планируется, что стартовой будет страница приложения со списком учебных классов. Поэтому для запроса учебных классов клиенту необходимо будет обратиться с GET запросом к базовой точке доступа сервера.

Когда сервер примет запрос он должен будет обратиться к базе данных, достать оттуда учебные классы и передать их клиенту. Для этого клиенту должно быть отправлено JSON [5] сообщение, представленное ниже.

```
[
  {
    "id": Integer,
    "number": String,
    "character ": String
  }
]
```

Здесь:

- id– уникальный идентификатор учебного класса,
- number– номер класса,
- character– буква класса.

Из предоставленного списка пользователь может выбрать конкретный класс, чтобы посмотреть список учеников и их возможную успеваемость. Для запроса учеников клиенту необходимо будет обратиться с GET запросом к точке доступа <URL сервера>/students. В качестве query параметра необходимо передать уникальный идентификатор класса.

Когда сервер примет запрос, он должен будет обратиться к базе данных, собрать всю информацию об ученика класса и передать ее в нейронную сеть для получения предсказания. Затем основную информацию об учениках с предсказанием их успеваемости нужно отправить клиенту. Для этого клиенту должно быть отправлено JSON сообщение, представленное ниже:

```
[
  {
```

```

        "class_group_id": Integer,
        "first_name": String,
        "last_name": String,
        "performance": Integer,
        "color": String,
        "true_performance": Integer
    },
]

```

Здесь:

- class_group_id уникальный идентификатор учебного класса,
- first_name – имя ученика,
- last_name – фамилия ученика,
- performance– номер, который соответствует типу успеваемости,
- цвет, который соответствует типу успеваемости,
- true_performance– номер, который соответствует истинному типу успеваемости (для проверки предсказания нейронной сети).

Параметр color будет формироваться следующим образом:

- hex код красного цвета, если успеваемость низкая,
- hex код оранжевого цвета, если успеваемость удовлетворительная,
- hex код зеленого цвета, если успеваемость выше удовлетворительной.

Для хранения данных, необходимо спроектировать базу данных, которая должна состоять из следующих таблиц:

- class group, которая должна содержать информацию об учебных классах,
- student data, которая должна содержать информацию, необходимую для предсказания успеваемости ученика,
- student, которая должна собирать в себе всю информацию об учениках.

ER- диаграмма данной базы данных представлена на рисунке 10.

2.2.2 Выбор инструментов разработки

Так как нейронная сеть была написана на языке Python с использованием библиотеки PyTorch, то для простой интеграции обученной модели нейронной сети в микросервис лучше всего будет использовать тот же язык для его разработки.

Далее необходимо выбрать фреймворк для разработки микросервиса. Выбор пал на следующие фреймворки:

- django,
- cherryPy,
- pyramid.

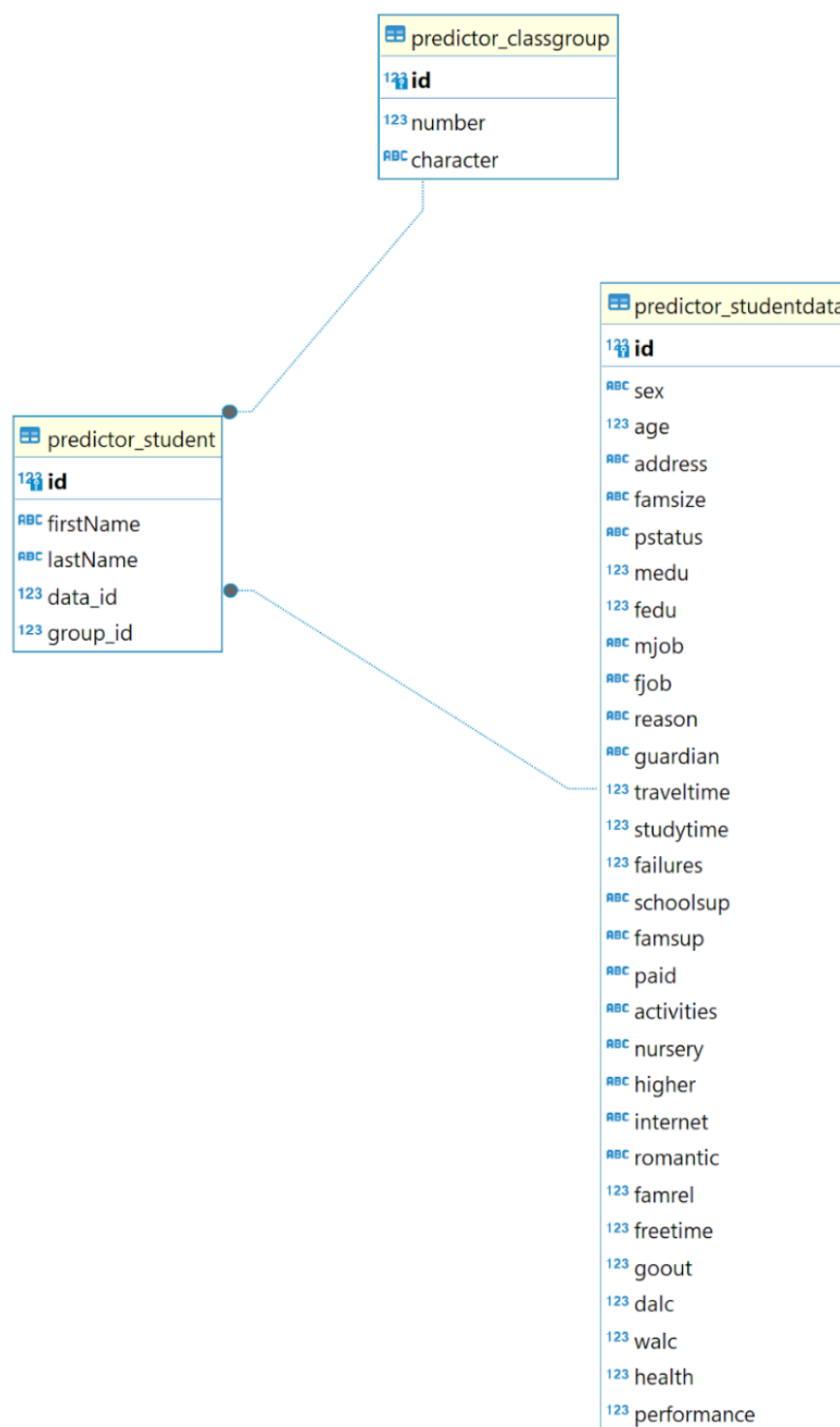


Рисунок 10 – ER- диаграмма базы данных

Django— один из самых популярных фреймворков. Он позволяет создавать веб- приложения любой сложности.

К преимуществам данного фреймворка можно отнести:

– большое сообщество разработчиков: популярность данного фреймворка и его открытый исходный код способствует появлению большого

количества библиотек, которые облегчают взаимодействие с фреймворком и расширяют его функционал,

- наличие собственного ORM,
- система кэширования и интернализации,
- система авторизации и аутентификации,
- шаблонизатор.

К недостаткам можно отнести ограниченную гибкость разработки. Фреймворк имеет жесткую структуру. Это может затруднить реализацию некоторого специфического или нестандартного функционала.

CherryPy – один из старейших фреймворков. Он зарекомендовал себя, как стабильный и производительный продукт. CherryPy максимально соответствует принципу «pythonic style», что позволяет создавать веб-приложения, как обычную объектно-ориентированную программу на Python.

К преимуществам данного фреймворка можно отнести:

- возможность запускать различные HTTP-серверы (например, на нескольких портах) одновременно,
- мощная система конфигурирования,
- встроенные инструменты работы с сессиями, кэшированием, аутентификацией, статическим контентом,
- встроенная поддержка профилирования и тестирования.

К недостаткам можно отнести:

- низкая популярность: фреймворк менее известен, чем Django, что может затруднить поиск ресурсов и готовых решений,
- отсутствие встроенного ORM.

Pyramid – молодой фреймворк, который набирает популярность. Он не ограничивает разработчиков жесткой структурой и в тоже время может предоставить все необходимые инструменты для создания проекта любой сложности.

К преимуществам данного фреймворка можно отнести:

- шаблонизатор,
- кэширование HTTP,
- встроенная поддержка сессий,
- управление транзакциями, в том числе при работе сразу с несколькими базами данных,
- встроенные инструменты для работы с аутентификацией.

К недостаткам данного фреймворка можно отнести отсутствие Object-Relational Mapping.

В рамках разработки серверной части будет использоваться фреймворк Django. Подробная документация, большое сообщество разработчиков, встроенная ORM являются весомыми аргументами в его пользу. Недостаток в виде жесткой структуры не позволяет допустить множество случайных ошибок. Этот факт оборачивает недостаток в достоинство данного фреймворка.

Далее необходимо выбрать базу данных.

Мой выбор остановился на реляционной базе данных SQLite. Этому способствовали следующие причины:

- django по умолчанию использует SQLite,
- компактный размер,
- высокая производительность,
- не требует установки сервера базы данных.

Затем необходимо выбрать интегрированную среду разработки. Основными критериями при выборе среды разработки были:

- возможность поддержки языка Python,
- интеграция с Git.

Этим условиям соответствуют следующие три среды разработки:

- 1) IntelliJ IDEA,
- 2) PyCharm,
- 3) Visual Studio Code.

IntelliJ IDEA популярная среда разработки, созданная компанией JetBrains, которая поддерживает множество языков программирования. Основным направлением данной среды является программирование на языке Java.

Преимущества:

- высокая интеллектуальность среды (дописывание кода, предложения о рефакторинге, навигация по коду),
- поддержка систем контроля версий,
- встроенный отладчик программы.

Недостатки:

- продукт предоставляется по платной подписке,
- высокая требовательность к использованию ресурсов оперативной памяти компьютера.

Среда разработки PyCharm была создана компанией Jet Brains специально для разработки на языке Python.

Преимущества:

- наличие бесплатной версии с богатым функционалом,
- авто завершение и инспекции кода,
- высокая интеллектуальность среды (дописывание кода, предложения о рефакторинге, навигация по коду),
- поддержка систем контроля версий,
- встроенный отладчик программы.

Недостатком является высокая требовательность к использованию ресурсов оперативной памяти компьютера.

Среда разработки Visual Studio Code была создана компанией Microsoft на основе ранее упомянутой IntelliJ IDEA для разработки разного рода приложений (мобильных приложений для устройств, управляемых операционной системой Android, веб приложений).

Преимущества:

- бесплатный продукт,
- встроенный эмулятор мобильных устройств,
- высокая интеллектуальность среды (дописывание кода, предложения о рефакторинге, навигация по коду),
- поддержка систем контроля версий,
- встроенный отладчик программы.

К недостатку можно отнести низкую производительность в сравнении с конкурентами.

На основании приведенных выше данных было принято решение в качестве среды разработки использовать PyCharm.

2.2.3 Реализация серверной части

Изначально я спроектировал базу данных для хранения информации об обучающихся. Для работы с базой данных Django предоставляет механизм миграций. Он используется для автоматического создания, применения и отката изменений в базе данных.

Для создания таблиц в базе данных были созданы классы-модели, которые наследуются от класса `django.db.models.Model`. Данный базовый класс предоставляет множество методов для работы с базой данных. Каждое поле подобного класса должно инициализироваться объектом специального класса, который необходим для определения типа данных поля в сгенерированной таблице базы данных. Я использовал следующие классы:

- `models.IntegerField`– создаст в базе данных целочисленное поле, значение которого может располагаться в промежутке от минуса 2147483648 до 2147483647,
- `models.CharField`– создаст в базе данных поле, которое может хранить строку,

– `models.ForeignKey`– создаст в базе данных поле, которое хранит внешний ключ (ссылку на запись в другой таблице).

Таким образом были написаны следующие модели:

`ClassGroup`– данная модель олицетворяет учебный класс. Содержит в себе поля `number` (номер класса), `character` (буквенное обозначение класса):

```
class ClassGroup(models.Model):  
    number = models.IntegerField()  
    character = models.CharField(max_length=1)
```

`StudentData`– данная модель содержит в себе данные об ученике, которые необходимы для предсказания его успеваемости:

```
class StudentData(models.Model):  
    sex = models.CharField(max_length=1)  
    age = models.IntegerField()  
    address = models.CharField(max_length=1)  
    famsize = models.CharField(max_length=3)  
    pstatus = models.CharField(max_length=1)  
    medu = models.IntegerField()  
    fedu = models.IntegerField()  
    mjob = models.CharField(max_length=128)  
    fjob = models.CharField(max_length=128)  
    reason = models.CharField(max_length=128)  
    guardian = models.CharField(max_length=128)  
    traveltime = models.IntegerField()  
    studytime = models.IntegerField()  
    failures = models.IntegerField()  
    schoolsup = models.CharField(max_length=128)  
    famsup = models.CharField(max_length=128)  
    paid = models.CharField(max_length=128)  
    activities = models.CharField(max_length=128)  
    nursery = models.CharField(max_length=128)  
    higher = models.CharField(max_length=128)  
    internet = models.CharField(max_length=128)  
    romantic = models.CharField(max_length=128)
```

```

famrel = models.IntegerField()
freetime = models.IntegerField()
goout = models.IntegerField()
dalc = models.IntegerField()
walc = models.IntegerField()
health = models.IntegerField()
performance = models.IntegerField()

```

Student— данная модель олицетворяет ученика. Содержит в себе поля `firstName` (Имя), `lastName` (Фамилия), `group` (id класса), `data` (id данных для предсказания успеваемости):

```

class Student(models.Model):
    firstName = models.CharField(max_length=128)
    lastName = models.CharField(max_length=128)
    group = models.ForeignKey(to=ClassGroup,
on_delete=models.PROTECT)
    data = models.ForeignKey(to=StudentData,
on_delete=models.PROTECT)

```

Далее мной были применены команды `manage.py makemigrations` и `manage.py migrate` для создания и применения миграций соответственно. Эти действия сгенерировали файл базы данных SQLite со всеми описанными таблицами.

Так же была написана еще одна модель `StudentPerformancePredicton`, которая содержит поля `class_group_id`, `first_name`, `last_name`, `performance`. Данная модель будет сформирована после предсказания успеваемости студента, сериализована в JSON и отправлена клиенту.

Механизм сериализации объектов реализован с помощью 2 классов:

- 1) `ModelSerializer`,
- 2) `Serializer`.

`ModelSerializer` предоставляет пакет `django rest framework`. Он используется для сериализации моделей, которые наследуются от класса `django.db.models.Model`. Для трансформации объекта в JSON необходимо

написать специальный класс- сериализатор, который наследуется от класса ModelSerializer. Данный класс должен содержать в себе подкласс Meta, в полях которого указываются:

- класс, объект которого будет трансформирован в JSON,
- какие поля класса необходимо сериализовать.

Таким образом был написан класс- сериализатор ClassGroupSerializer, который позволяет трансформировать в объект JSON ранее описанную модель ClassGroup:

```
class ClassGroupSerializer(serializers.ModelSerializer):  
    class Meta:  
        model = ClassGroup  
        fields = '__all__'
```

Serializer – это класс, который используется для сериализации объектов. Класс-сериализатор должен наследоваться от serializers.Serializer и содержать в себе поля, которые будут трансформированы в JSON.

Таким образом был написан класс StudentPerformancePredictionSerializer, который позволяет трансформировать в JSON ранее описанную модель StudentPerformancePredicton:

```
class  
StudentPerformancePredictionSerializer(serializers.Serializer):  
    class_group_id = serializers.IntegerField()  
    first_name = serializers.CharField()  
    last_name = serializers.CharField()  
    performance = serializers.IntegerField()
```

Далее я приступил к разработке API сервера. Оно было реализовано с помощью класса APIView, который предоставляет пакет django rest framework.

Для реализации запроса, который возвращает список всех учебных классов был реализован класс GroupList, который наследуется от упомянутого ранее APIView. Внутри данного класса написан метод get, который берет из

базы данных все записи таблицы ClassGroup, трансформирует их в JSON и отправляет клиенту:

```
class GroupList(APIView):
    @staticmethod
    def get(request):
        groups = ClassGroup.objects.all()
        serializer = ClassGroupSerializer(groups, many=True)
        return Response(serializer.data)
```

Для реализации запроса, который возвращает список студентов конкретного класса, с предсказанием их успеваемости был реализован класс StudentWithPrediction, который наследуется от упомянутого ранее APIView. Внутри данного класса написан метод get, который в качестве параметра принимает объект request. Данный объект содержит в себе данные, которые пришли вместе с запросом от клиента. Из данного объекта берется идентификатор учебного класса. Затем по данному идентификатору из базы данных выбираются студенты, которые обучаются в данном классе. С помощью обученной ранее нейронной сети для каждого студента осуществляется предсказание его успеваемости, затем формируется объект StudentPerformancePredicton, который добавляется в список. Сформированный список трансформируется в JSON и отправляется клиенту:

```
class StudentWithPrediction(APIView):
    @staticmethod
    def get(request):
        group_id = request.query_params['group_id']
        group_students = Student.objects.filter(group=group_id)
        predictor = StudentPerformancePredictor()

        student_performance_predictions = []
        for student in group_students:
            student_data = StudentData.objects.filter(id=student.data.id)
```

```

        prediction = predictor.predict(student_data)
        student_performance_predict =
StudentPerformancePrediction(group_id, student.firstName,
student.lastName,

prediction)

student_performance_predictions.append(student_performance_p
redict)

        serializer =
StudentPerformancePredictionSerializer(student_performance_p
redictions, many=True)

        return Response(serializer.data)

```

Для привязки методов API к определенным URL в Django используется список `urlpatterns`, который содержит в себе объекты `path`. Данный объект предоставляется фреймворком:

```

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', GroupList.as_view()),
    path('student/', StudentWithPrediction.as_view()),
    path('init/', InitDB.as_view())
]

```

Далее я приступил к разработке класса для работы с нейронной сетью. Был реализован класс `StudentPerformancePredictor`. В конструктор данного класса передаются следующие параметры:

- `in_size`– количество нейронов входного слоя,
- `hidden_size`– количество нейронов скрытого слоя,
- `hidden_layer_count`– количество скрытых слоев,
- `out_size`– количество нейронов выходного слоя.

Внутри конструктора вызывается метод `_create_model`, который возвращает модель нейронной сети:

```

        def __init__(self, in_size=30, hidden_size=39,
hidden_layers_count=11, output_size=4):
            self.model = self.__create_model(in_size,
hidden_size, hidden_layers_count, output_size)

```

Упомянутый ранее метод `_create_model` в качестве параметров принимает все параметры, которые были переданы в конструктор класса и на их основании создает модель нейронной сети и загружает ее веса из файла с разрешением `.pt`, который был создан при обучении нейронной сети:

```

    @staticmethod
        def __create_model(input_size, hidden_size,
hidden_layers_count, output_size):
            layers = [nn.Linear(input_size, hidden_size),
nn.ReLU()]

            for _ in range(hidden_layers_count - 1):
                layers.append(nn.Linear(hidden_size,
hidden_size))
                layers.append(nn.ReLU())

            layers.append(nn.Linear(hidden_size, output_size))

            model = nn.Sequential(*layers)

            model.load_state_dict(torch.load("predictor/nn_models/model_
a841_f1838_e100_hs39_hlc11_lr0.0012_is30_os4.pt"))
            model.eval()

            return model

```

Так же внутри данного класса реализован метод `predict`, который в качестве параметра принимает объект `StudentData`. Внутри данного метода вызывается метод `__get_inputs` для преобразования входных данных в вид,

который необходим для работы с нейронной сетью и отправляет в модель, которая дает предсказание:

```
def predict(self, student_data):
    df_tensor = self.__get_inputs(student_data)

    with torch.no_grad():
        outputs = self.model(df_tensor)

    _, predicted = torch.max(outputs.data, 1)

    return predicted
```

Упомянутый ранее метод `__get_inputs` в качестве параметра принимает объект `StudentData`, формирует из него тензор и возвращает:

```
@staticmethod
def __get_inputs(student_data):
    data = list(student_data.values())
    df = pd.DataFrame(data)
    df = df.drop(columns=['performance', 'id'])
    df_np = df.to_numpy()

    df_tensor =
    torch.from_numpy(df_np.astype(np.float32))
    return df_tensor
```

3 Описание результатов

Было произведено множество итераций подбора гиперпараметров модели нейронной сети. Первым удачным результатом стала модель, которая выдает предсказания с точностью в шестьдесят девять процентов. Модель состояла из:

- одного входного слоя (количество нейронов равно количеству признаков, а именно тридцати),
- десяти скрытых слоев (каждый слой состоит из четырнадцати нейронов),
- одиннадцати слоев активации, которые используют функцию ReLU,
- одного выходного слоя, который состоит из четырех нейронов (по одному на предсказываемый класс).

Скорость обучения была равна 0.0013.

Графики изменения показателя точности относительно эпохи обучения можно увидеть на рисунке 11.

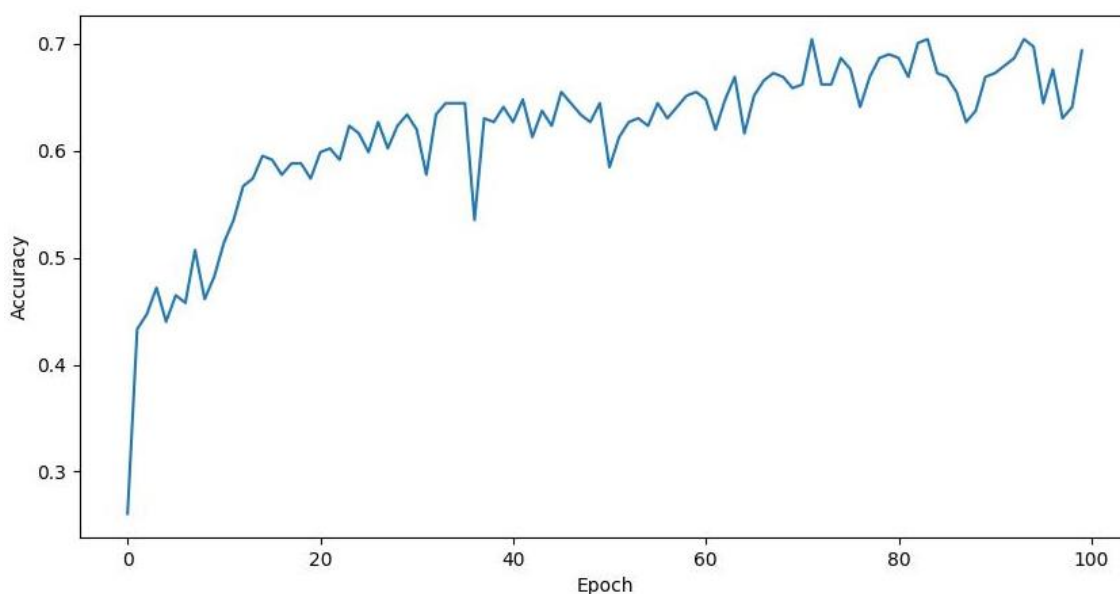


Рисунок 11 – Графики изменения показателя точности относительно эпохи обучения

Можно заметить, что кривая графика идет по восходящей, что свидетельствует об обучении модели нейронной сети.

Так же процесс обучения можно проследить на графике изменения значения функции потерь относительно эпохи обучения. Такой график можно увидеть на рисунке 12.

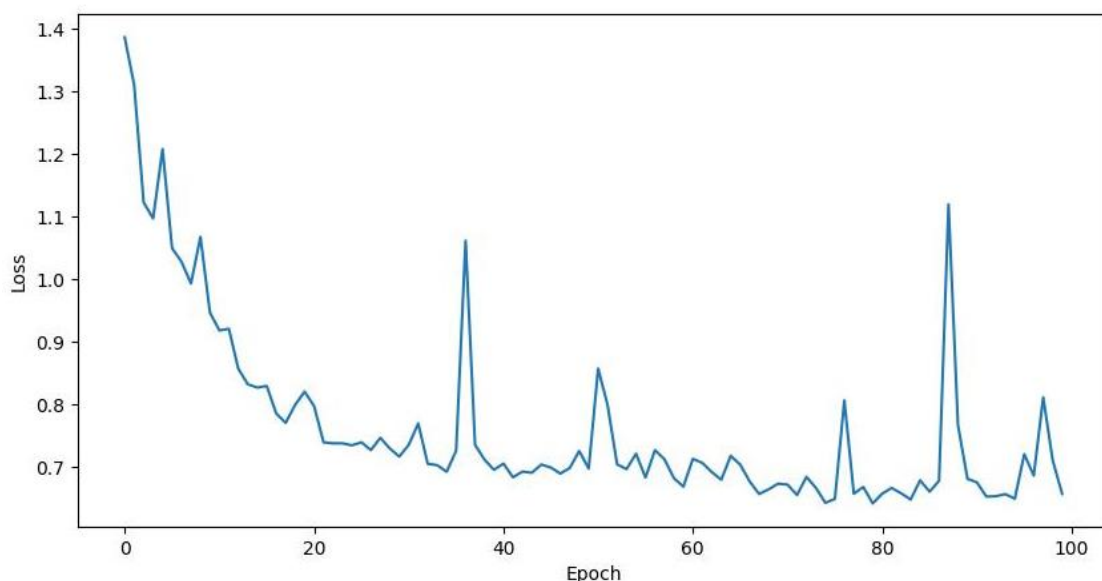


Рисунок 12 – Графики изменения значения функции потерь относительно эпохи обучения

Можно заметить, что кривая графика идет по нисходящей, что так же свидетельствует об обучении модели нейронной сети.

Высокая точность в одиночку не свидетельствует о высоком качестве модели, поэтому были сняты еще метрики.

Графики изменения показателя F1- метрики относительно эпохи обучения можно увидеть на рисунке 13.

Можно заметить, что кривая графика идет по восходящей, что свидетельствует о увеличении качества модели нейронной сети.

Так же для «полноты картины» была снята метрика precision, график изменения которой относительно эпохи обучения можно увидеть на рисунке 14. Можно заметить, что кривая графика идет по восходящей, что свидетельствует о увеличении качества модели нейронной сети.

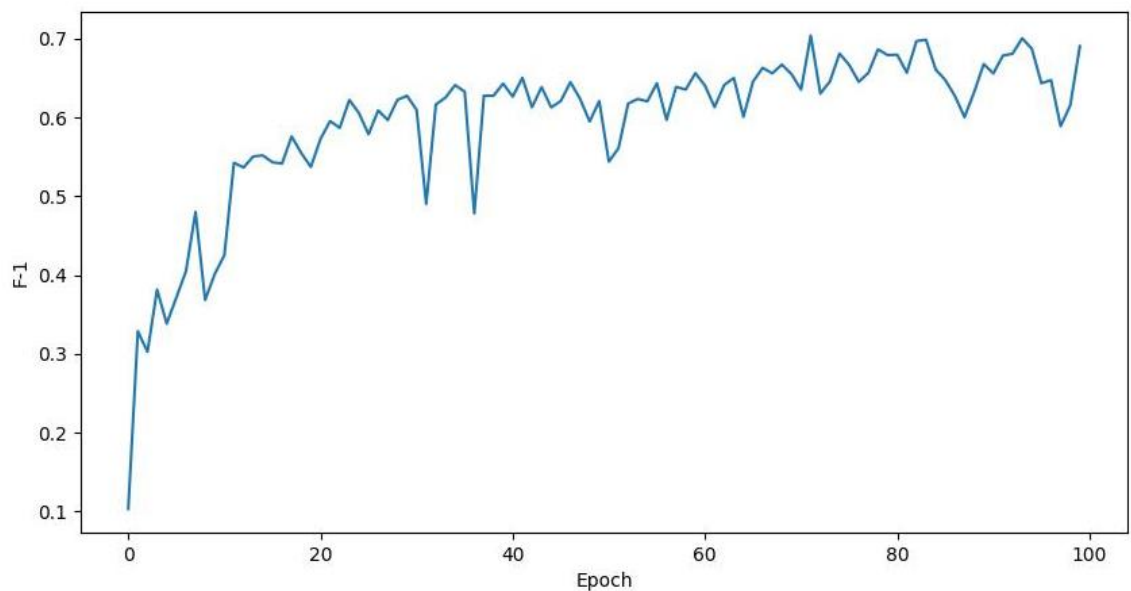


Рисунок 13 – Графики изменения показателя F1- метрики относительно эпохи обучения

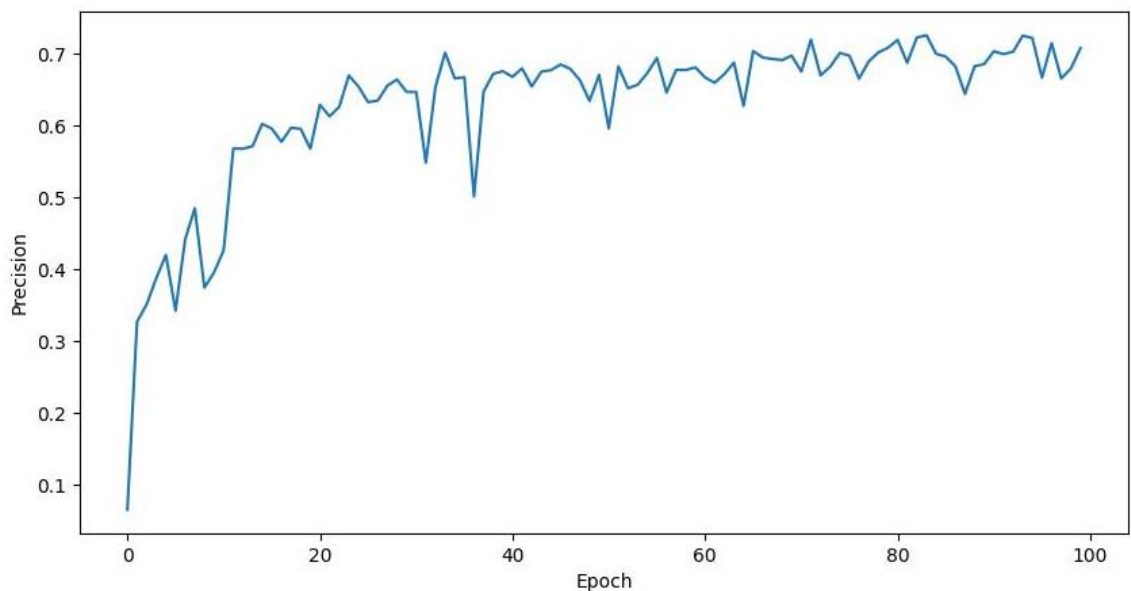


Рисунок 14 – Графики изменения показателя precision- метрики относительно эпохи обучения

Еще была снята метрика recall, график изменения которой относительно эпохи обучения можно увидеть на рисунке 15. Можно заметить, что кривая графика идет по восходящей, что свидетельствует о увеличении качества модели нейронной сети.

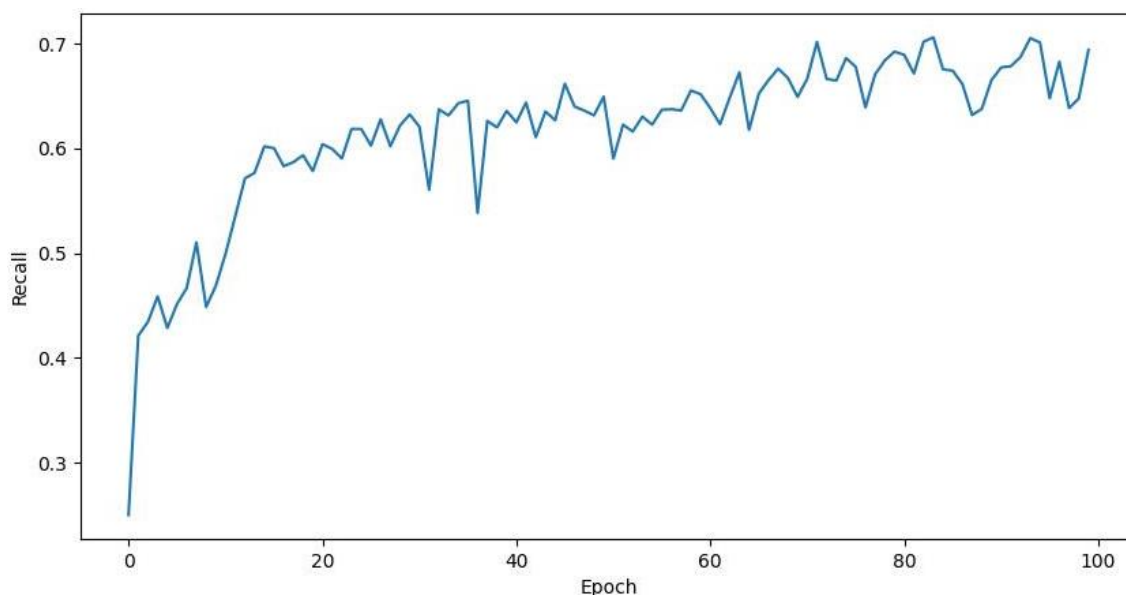


Рисунок 15 – Графики изменения показателя recall- метрики относительно эпохи обучения

Следующим удачным результатом стала модель, которая выдает предсказания с точностью в семьдесят пять процентов. Модель состояла из:

- одного входного слоя (количество нейронов равно количеству признаков, а именно тридцати),
- двадцати одного скрытого слоя (каждый слой состоит из дтридцати нейронов),
- двадцати двух слоев активации, которые используют функцию ReLU,
- одного выходного слоя, который состоит из четырех нейронов (по одному на предсказываемый класс).

Скорость обучения была равна 0.001.

Графики изменения показателя точности относительно эпохи обучения можно увидеть на рисунке 16. Можно заметить, что кривая графика идет по восходящей, что свидетельствует об обучении модели нейронной сети.

График изменения значения функции потерь относительно эпохи обучения можно увидеть на рисунке 17. Можно заметить, что кривая графика идет по нисходящей, что так же свидетельствует об обучении модели нейронной сети.

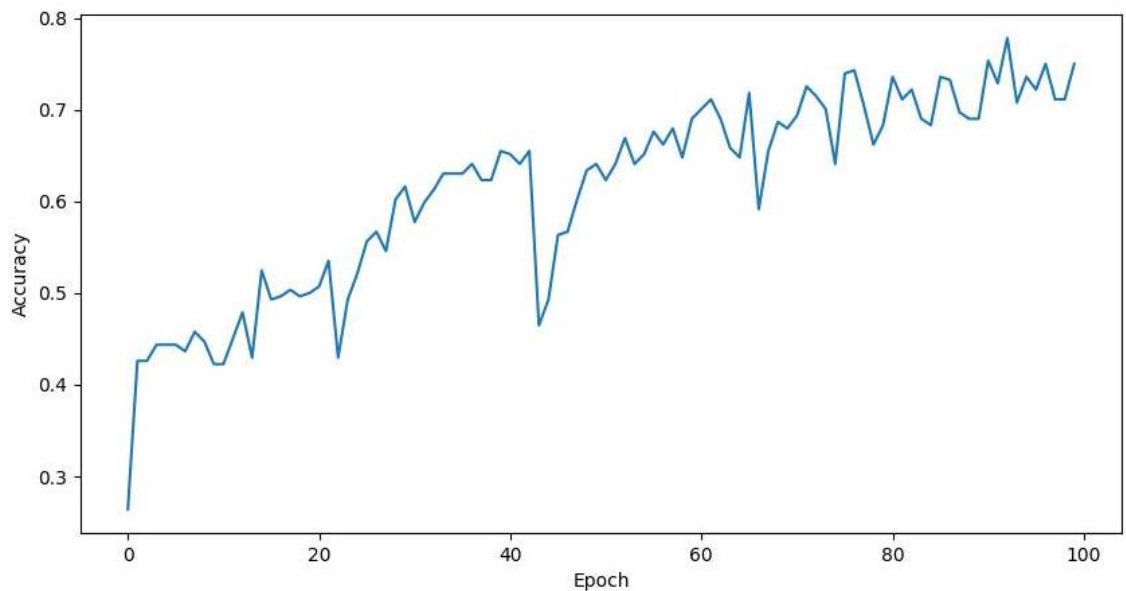


Рисунок 16 – Графики изменения показателя точности относительно эпохи обучения

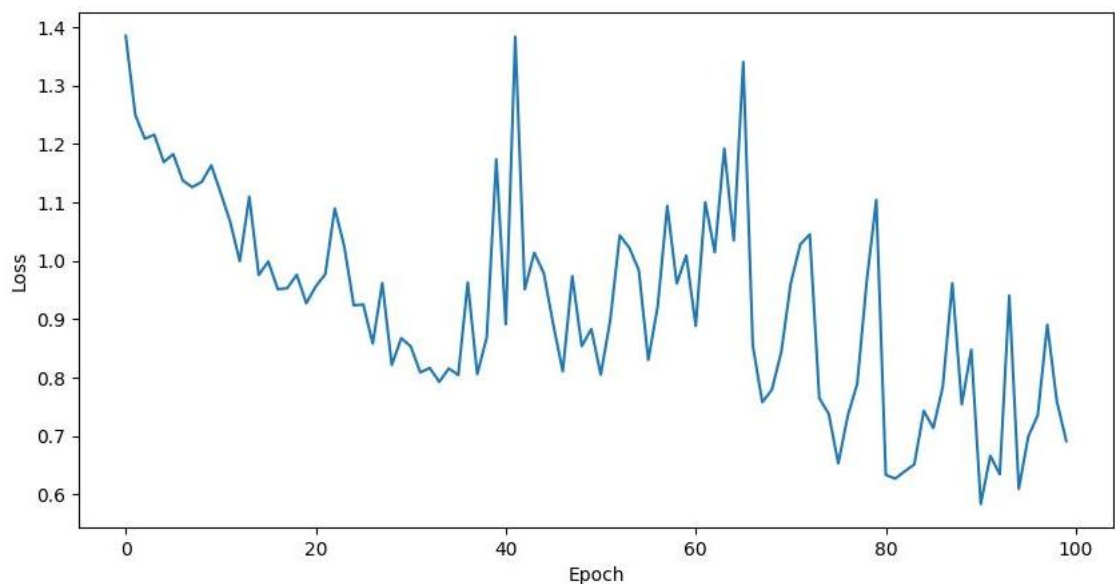


Рисунок 17 – Графики изменения значения функции потерь относительно эпохи обучения

График изменения показателя f1- метрики относительно эпохи обучения представлен на рисунке 18. Можно заметить, что кривая графика идет по восходящей, что свидетельствует об увеличении качества модели нейронной сети.

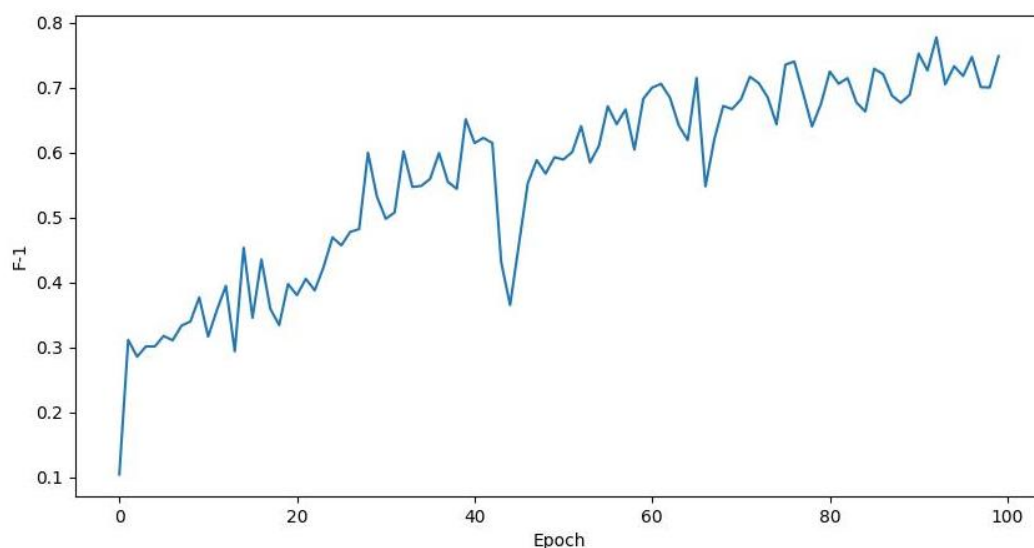


Рисунок 18 – Графики изменения показателя F1- метрики относительно эпохи обучения

График изменения показателя precision- метрики относительно эпохи обучения представлен на рисунке 19.

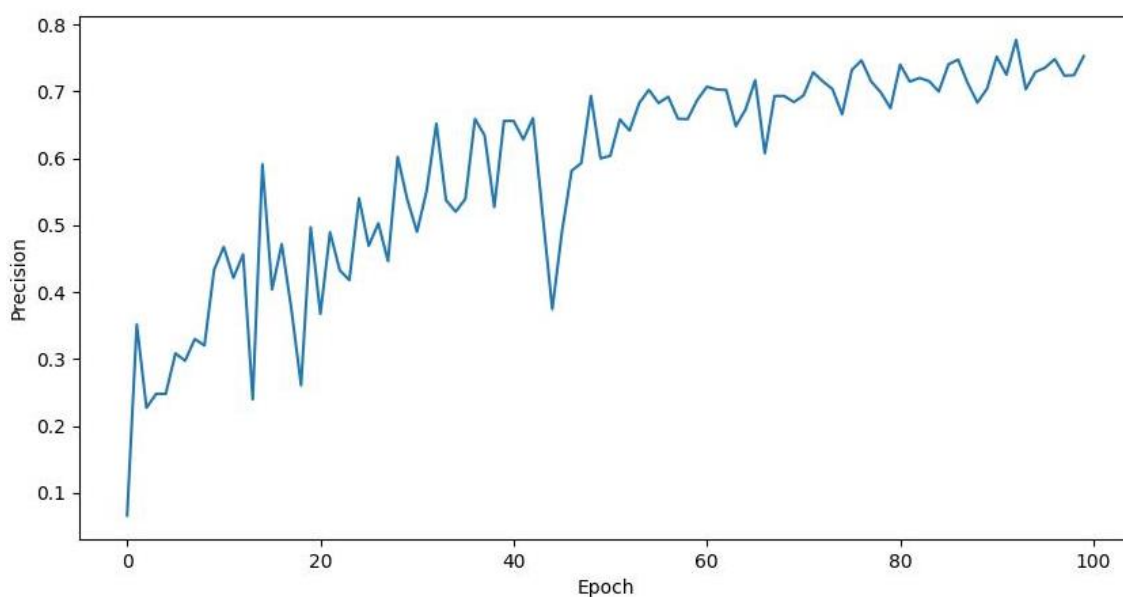


Рисунок 19 – Графики изменения показателя precision- метрики относительно эпохи обучения

Можно заметить, что кривая графика идет по восходящей, что свидетельствует об увеличении качества модели нейронной сети.

График изменения показателя recall- метрики относительно эпохи обучения представлен на рисунке 20.

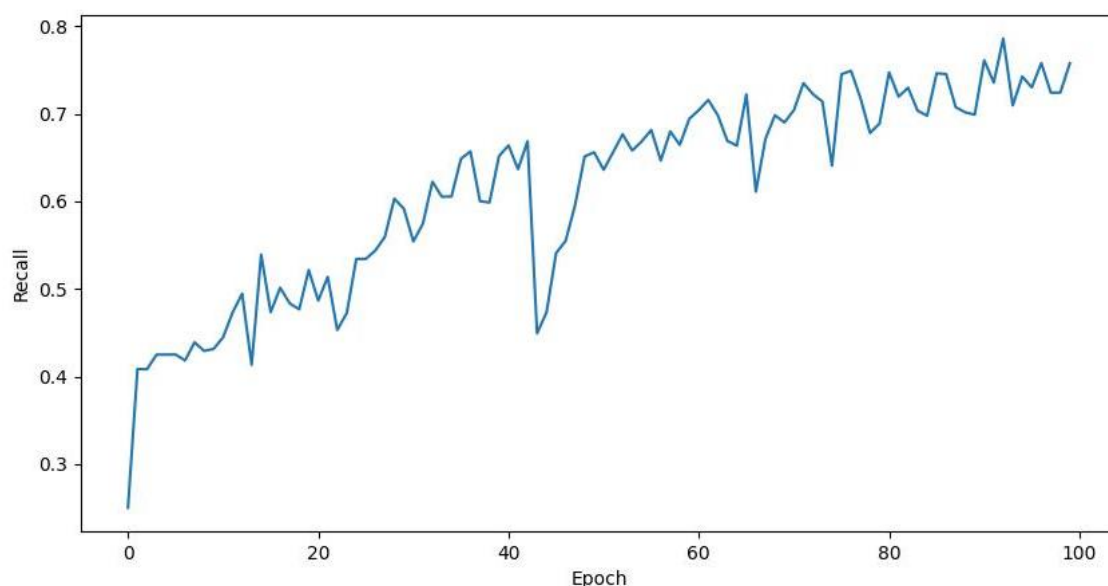


Рисунок 20 – Графики изменения показателя recall- метрики относительно эпохи обучения

Можно заметить, что кривая графика идет по восходящей, что свидетельствует об увеличении качества модели нейронной сети.

В результате работы алгоритма подбора гиперпараметров самым удачным результатом стала модель, которая состоит из:

- одного входного слоя (количество нейронов равно количеству признаков, а именно тридцати),
- одиннадцати скрытых слоев (каждый слой состоит тридцати девяти нейронов),
- двенадцати слоев активации, которые используют функцию ReLU,
- одного выходного слоя, который состоит из четырех нейронов (по одному на предсказываемый класс).

Скорость обучения равна 0.0012.

В результате обучения модель стала способна предсказывать успеваемость студентов с точностью в восемьдесят четыре процента.

Графики изменения показателя точности относительно эпохи обучения можно увидеть на рисунке 21.

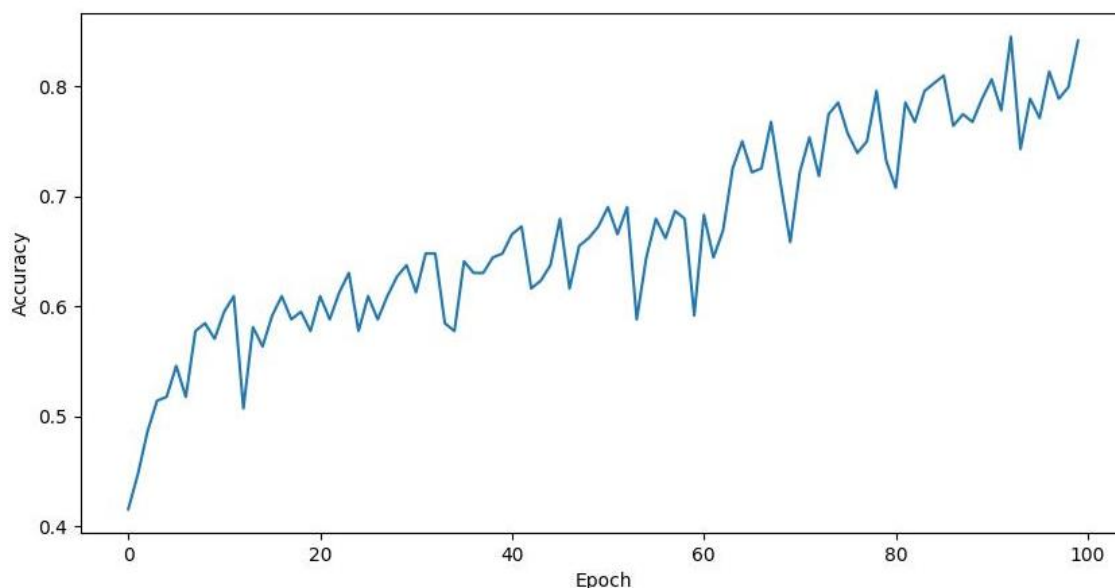


Рисунок 21 – Графики изменения показателя точности относительно эпохи обучения

Можно заметить, что кривая графика идет по восходящей, что свидетельствует об обучении модели нейронной сети.

График изменения значения функции потерь относительно эпохи обучения можно увидеть на рисунке 22.

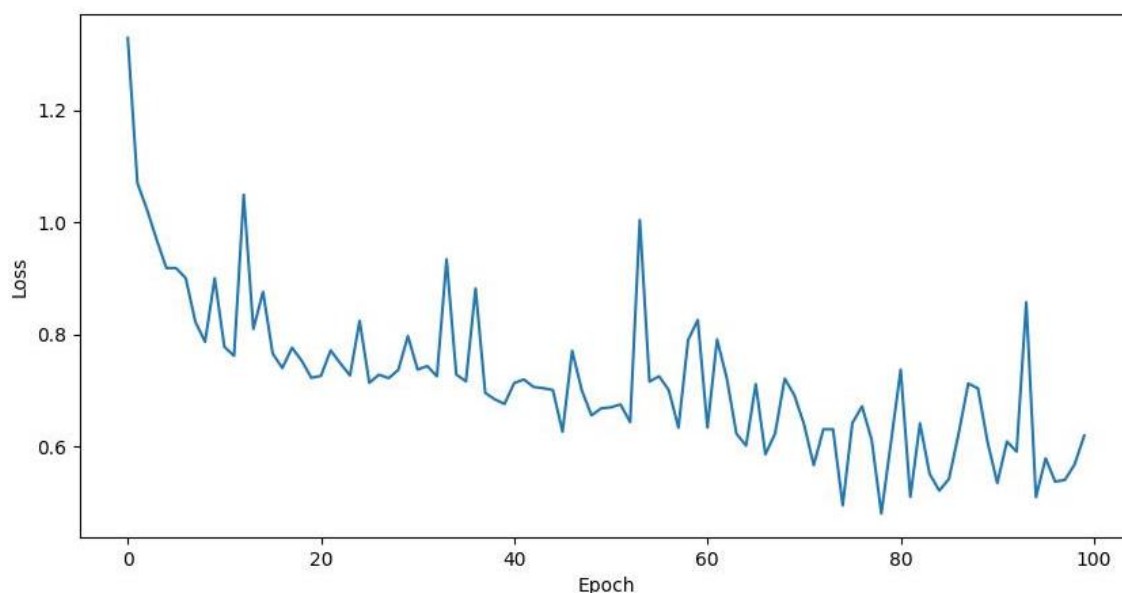


Рисунок 22 – Графики изменения значения функции потерь относительно эпохи обучения

Можно заметить, что кривая графика идет по нисходящей, что так же свидетельствует об обучении модели нейронной сети.

График изменения показателя f1- метрики относительно эпохи обучения представлен на рисунке 23.

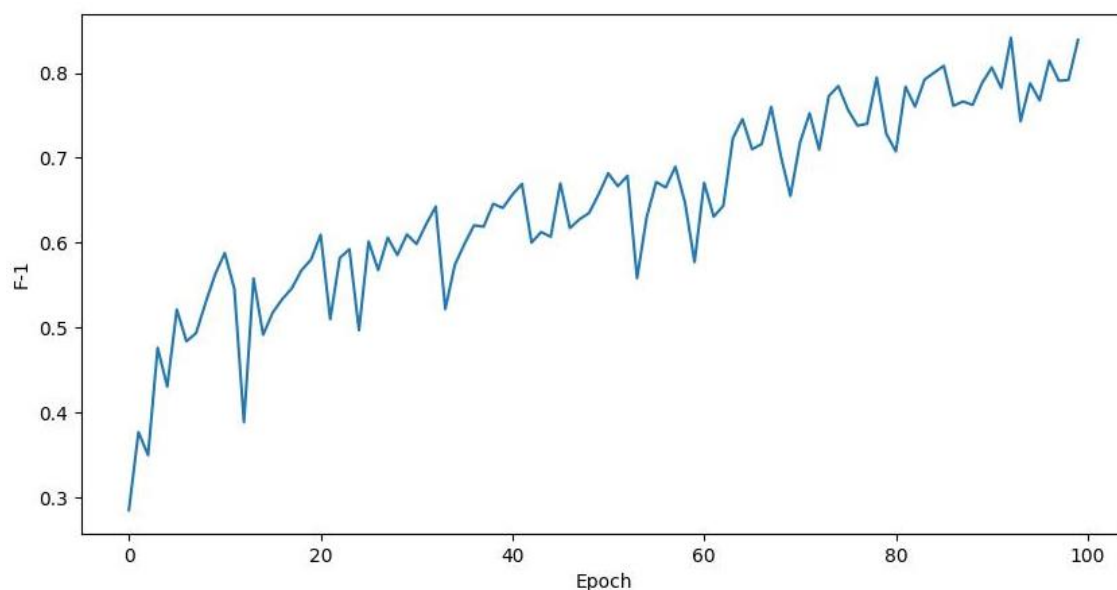


Рисунок 23 – Графики изменения показателя F1- метрики относительно эпохи обучения

Можно заметить, что кривая графика идет по восходящей, что свидетельствует об увеличении качества модели нейронной сети.

График изменения показателя precision- метрики относительно эпохи обучения представлен на рисунке 24.

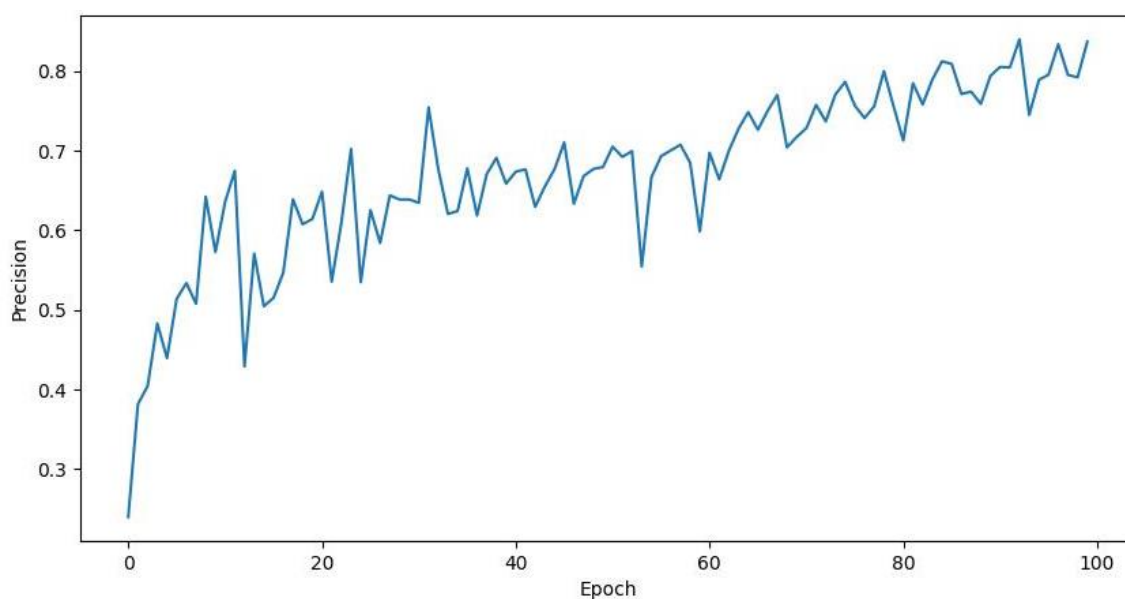


Рисунок 24 – Графики изменения показателя precision- метрики относительно эпохи обучения

Можно заметить, что кривая графика идет по восходящей, что свидетельствует об увеличении качества модели нейронной сети.

График изменения показателя recall- метрики относительно эпохи обучения представлен на рисунке 25.

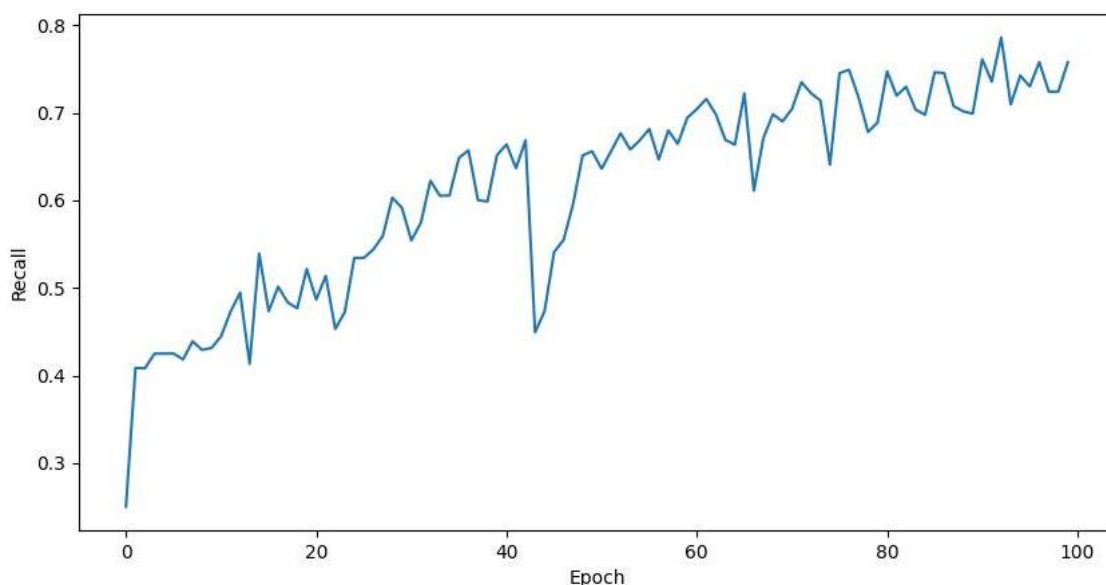


Рисунок 25 – Графики изменения показателя recall- метрики относительно эпохи обучения

Можно заметить, что кривая графика идет по восходящей, что свидетельствует об увеличении качества модели нейронной сети.

Работу разработанного микросервиса можно проверить с помощью Insomnia.

Insomnia— это высокотехнологичное и комплексное программное обеспечение для тестирования REST API.

Микросервис будет запускаться локально на порту 8000, потому базовым адресом, по которому будет обращаться клиент является «http://127.0.0.1:8000/».

Endpoint, по которому клиент может обратиться для получения списка учебных классов— это базовый адрес микросервиса. При отправке get запроса,

если не возникала ошибка клиент получит ответ с кодом 200 и список JSON, с информацией об учебных классах, а именно:

- идентификатор класса,
- номер класса,
- буква класса.

Скриншот с ответом сервера представлен на рисунке 26.

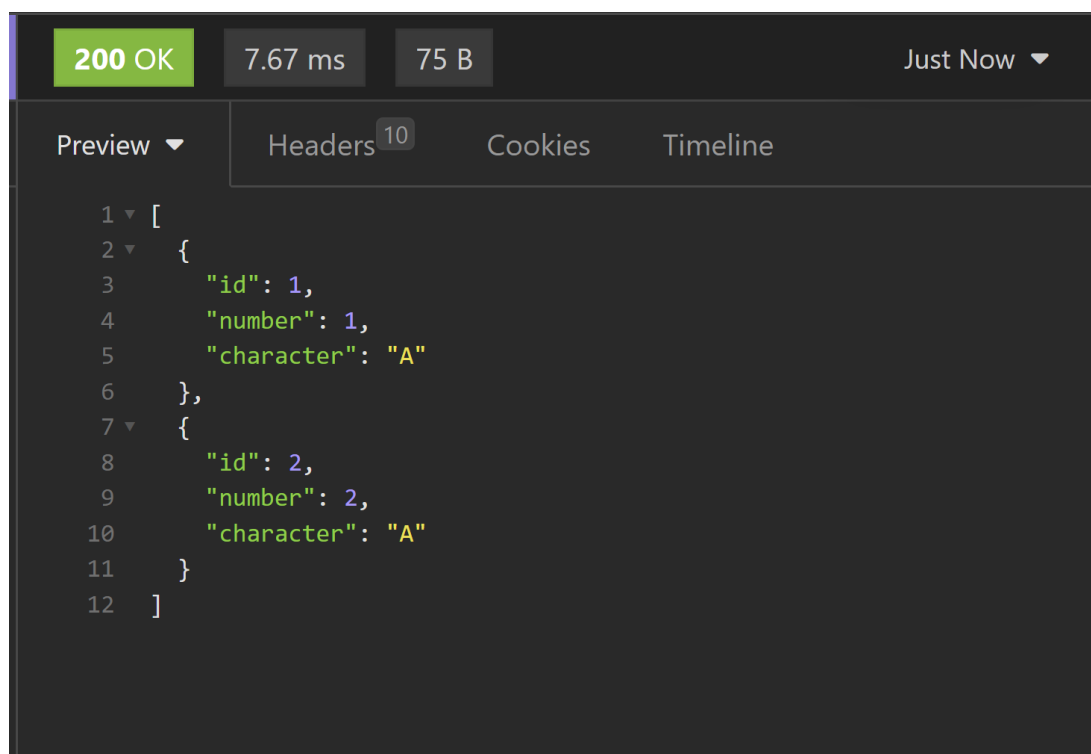


Рисунок 26 – Ответ на запрос учебных классов

Endpoint, по которому клиент может обратиться для получения списка учеников с предсказанием их успеваемости- это «<http://127.0.0.1:8000/students>». Клиенту необходимо отправить get запрос и в качестве query параметра group_id указать идентификатор учебного класса. Если не возникала ошибка клиент получит ответ с кодом 200 и список JSON, с информацией об учениках а именно:

- идентификатор ученика,
- идентификатор класса,
- имя ученика,
- фамилия ученика,

- предсказанная успеваемость,
- цвет, который соответствует предсказанной успеваемости,
- правильная итоговая успеваемость ученика.

Скриншот с ответом сервера представлен на рисунке 27.

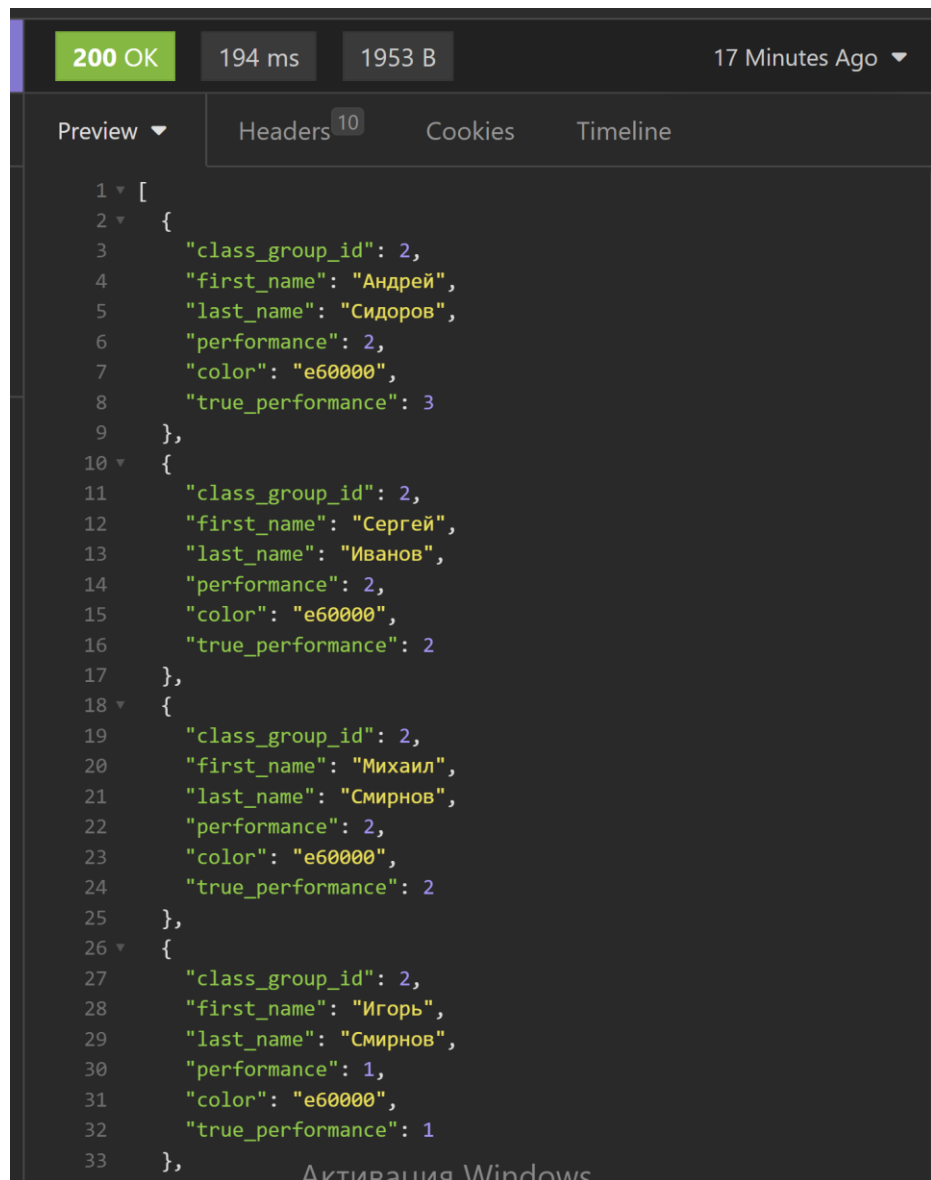


Рисунок 27- Ответ на запрос успеваемости учеников

ЗАКЛЮЧЕНИЕ

В процессе исследования была подтверждена актуальность диссертации. Также был проведен анализ целевой аудитории и ее потребностей. На основании потребностей был определен основной функционал проекта, который реализуется в рамках диссертации.

Были рассмотрены аналоги работы, выявлены их преимущества и недостатки. Было выявлено, что ни один из них не обладает функционалом, который предоставляет данный проект.

Был проведен анализ популярных типов нейронных сетей, на основании которого было решено разрабатывать сеть прямого распределения в рамках данной диссертации.

После, в рамках выбора инструментов разработки был выбран язык программирования Python. Так же были рассмотрены четыре популярных библиотеки для проектирования и обучения нейронных сетей: TensorFlow, Keras, Caffe и PyTorch. На основании их сравнения последняя была выбрана для разработки модели нейронной сети. Еще были рассмотрены три фреймворка для создания веб-приложений: Django, Pyramid, CherryPy. На основании их анализа первый был выбран для разработки микросервиса. Для реализации проекта была выбрана база данных SQLite. Этому послужили простота интеграции с ORM Django и легковесность данной БД. Также были проанализированы три среды разработки: IntelliJ IDEA, PyCharm, Visual Studio Code. После рассмотрения их преимуществ и недостатков вторая была выбрана для разработки.

Согласно поставленным задачам, была спроектирована и обучена нейронная сеть для предсказания успеваемости обучающихся, которая дает предсказание с точностью в восемьдесят четыре процента. Так же был разработан микросервис, в который была интегрирована обученная модель нейронной сети. В процессе разработки были укреплены навыки работы с языком Python. Так же были укреплены знания и навыки проектирования и

обучения нейронных сетей. Были приобретены новые знания и навыки разработки микросервисов с использованием фреймворка Django.

В процессе реализации данного проекта были достигнуты все поставленные цели. Разработанные нейронная сеть и микросервис работают исправно и выполняет все предназначенные им задачи. В перспективе проект может улучшаться. Планируется повысить точность модели, добавить функционал авторизации пользователей.-

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Школы и колледжи с 1 января должны использовать в образовательном процессе государственные информационные системы [Электронный ресурс]. – URL: <https://edu.gov.ru/press/6408/shkoly-i-kolledzhi-s-1-yanvary-a-dolzheny-ispolzovat-v-obrazovatelnom-processe-gosudarstvennye-informacionnye-sistemy> (дата обращения 01.06.2024).

2. Федеральный закон "О внесении изменений в Федеральный закон "Об образовании в Российской Федерации" от 30.12.2021 N 472-ФЗ (последняя редакция) [Электронный ресурс]. – URL: https://www.consultant.ru/document/cons_doc_LAW_405488 (дата обращения 01.06.2024).

3. Введение в архитектуры нейронных сетей [Электронный ресурс]. – URL: <https://habr.com/ru/companies/oleg-bunin/articles/340184> (дата обращения 02.06.2024).

4. Как победить несбалансированность датасета: метод upsampling data [Электронный ресурс]. – URL: <https://habr.com/ru/articles/568266> (дата обращения 03.06.2024).

5. Что такое JSON [Электронный ресурс]. – URL: <https://habr.com/ru/articles/554274> (дата обращения 03.06.2024).

ПРИЛОЖЕНИЕ А

(обязательное)

Цифровой помощник педагога

Листинг программы

ПРИЛОЖЕНИЕ А.1

Файл student_performance_nn.py

```
import os

import torch
import torch.nn as nn
import matplotlib.pyplot as plt

from sklearn.metrics import precision_score, recall_score,
f1_score


class StudentPerformanceNN2:
    def __init__(self, input_size, output_size,
hidden_layers_count, hidden_size, learning_rate, epochs):
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_layers_count = hidden_layers_count
        self.hidden_size = hidden_size
        self.learning_rate = learning_rate
        self.epochs = epochs

        self.model = self._build_model()
        self.loss_fn = nn.CrossEntropyLoss()
        self.optimizer =
torch.optim.Adam(self.model.parameters(), lr=self.learning_rate)

        self.metrics = {'loss': [], 'accuracy': [], 'recall':
[], 'precision': [], 'f1': []}

    def _build_model(self):
```

```

        layers = [nn.Linear(self.input_size,
self.hidden_size), nn.ReLU()]

        for _ in range(self.hidden_layers_count - 1):
            layers.append(nn.Linear(self.hidden_size,
self.hidden_size))
            layers.append(nn.ReLU())

        layers.append(nn.Linear(self.hidden_size,
self.output_size))

        model = nn.Sequential(*layers)

        return model

def train(self, train_data, test_data, is_log):
    self.model.train()
    for epoch in range(self.epochs):

        for inputs, labels in train_data:
            self.optimizer.zero_grad()
            outputs = self.model(inputs)
            loss = self.loss_fn(outputs, labels)
            loss.backward()
            self.optimizer.step()

        self._save_metrics(test_data)

    if epoch % 10 == 0 and is_log:
        self._print_last_metrics(epoch)

```

```

def _save_metrics(self, test_data):
    epoch_accuracy,    epoch_recall,    epoch_precision,
epoch_f1, epoch_loss = self.test_model(test_data)

    self.metrics['accuracy'].append(epoch_accuracy)
    self.metrics['recall'].append(epoch_recall)
    self.metrics['precision'].append(epoch_precision)
    self.metrics['f1'].append(epoch_f1)
    self.metrics['loss'].append(epoch_loss)

def _print_last_metrics(self, epoch):
    last_loss = self.metrics['loss'][-1]
    last_accuracy = self.metrics['accuracy'][-1]
    last_recall = self.metrics['recall'][-1]
    last_precision = self.metrics['precision'][-1]
    last_f1 = self.metrics['f1'][-1]

    print(f'\nEpoch: {epoch}')
    print(f'Accuracy: {last_accuracy}')
    print(f'Recall: {last_recall}')
    print(f'Precision: {last_precision}')
    print(f'F-1: {last_f1}')
    print(f'Loss: {last_loss}')

def plot_metrics(self):
    self._plot_loss()
    self._plot_accuracy()
    self._plot_recall()
    self._plot_precision()
    self._plot_f1()

```

```
def _plot_loss(self):
    plt.figure(figsize=(10, 5))
    plt.plot(self.metrics['loss'])
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    self._save_plot(plt, 'loss.jpg')

def _plot_accuracy(self):
    plt.figure(figsize=(10, 5))
    plt.plot(self.metrics['accuracy'])
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    self._save_plot(plt, 'accuracy.jpg')

def _plot_recall(self):
    plt.figure(figsize=(10, 5))
    plt.plot(self.metrics['recall'])
    plt.xlabel('Epoch')
    plt.ylabel('Recall')
    self._save_plot(plt, 'recall.jpg')

def _plot_precision(self):
    plt.figure(figsize=(10, 5))
    plt.plot(self.metrics['precision'])
    plt.xlabel('Epoch')
    plt.ylabel('Precision')
    self._save_plot(plt, 'precision.jpg')

def _plot_f1(self):
    plt.figure(figsize=(10, 5))
```

```

plt.plot(self.metrics['f1'])
plt.xlabel('Epoch')
plt.ylabel('F-1')
self._save_plot(plt, 'f1.jpg')

def _save_plot(self, plt, metric_graph_name):
    model_name = self._get_model_name()

    if not os.path.exists(os.path.join('metrics',
model_name)):

        os.mkdir(os.path.join('metrics', model_name))

        plt.savefig(os.path.join('metrics', model_name,
metric_graph_name))

        plt.close()

def _get_model_name(self):
    last_accuracy = self.metrics['accuracy'][-1]
    last_f1 = self.metrics['f1'][-1]
    int_accuracy = int(last_accuracy * 1000)
    int_f1 = int(last_f1 * 1000)
    return
f'model_a{int_accuracy}_f1{int_f1}_e{self.epochs}_hs{self.hidden_
size}_hlc{self.hidden_layers_count}' \
        f'_lr{round(self.learning_rate,
4)}_is{self.input_size}_os{self.output_size}'

def test_model(self, test_data):
    self.model.eval()

    for inputs, labels in test_data:
        with torch.no_grad():
            outputs = self.model(inputs)

```

```

_, predicted = torch.max(outputs.data, 1)

total = labels.size(0)
correct = (predicted == labels).sum().item()
y_pred = predicted.cpu().numpy()
y_true = labels.numpy()

accuracy = correct / total
recall = recall_score(y_true, y_pred,
average='macro')

precision = precision_score(y_true, y_pred,
average='macro', zero_division=0)
f1 = f1_score(y_true, y_pred, average='macro')
loss = self.loss_fn(outputs, labels).item()

self.model.train()

return accuracy, recall, precision, f1, loss

def save_model(self):
    model_name = self._get_model_name()
    torch.save(self.model.state_dict(),
os.path.join('models', model_name + '.pt'))

```

ПРИЛОЖЕНИЕ А.2

Файл data_prepare.py

```

import numpy as np

import pandas as pd

import torch

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

```

```

from torch.utils.data import DataLoader, TensorDataset

class DataPrepare:

    def get_data(self, file_path):

        data = self._get_prepared_DF(file_path)

        X = data.drop('performance', axis=1).values
        y = data['performance'].values

        label_encoder = LabelEncoder()
        y = label_encoder.fit_transform(y)

        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

        # Преобразование данных в тензоры
        X_train = torch.Tensor(X_train)
        X_test = torch.Tensor(X_test)
        y_train = torch.LongTensor(y_train)
        y_test = torch.LongTensor(y_test)

        batch_size = 32

        train_dataset = TensorDataset(X_train, y_train)

        train_loader = DataLoader(dataset=train_dataset,
batch_size=batch_size, shuffle=True)

        test_dataset = TensorDataset(X_test, y_test)

```

```

        test_loader = DataLoader(dataset=test_dataset,
batch_size=y_test.shape[0])

    return train_loader, test_loader

def _get_prepared_DF(self, file_path):
    df = pd.read_excel(file_path)

    # Создаю столбец цуспеваемости и делю столбец G3 на
категории успеваемости

    df['performance'] = pd.cut(df['G3'], bins=[-1, 5, 10, 15,
20], labels=False)

    num_cols = df.select_dtypes(include='int64').columns

    num_cols = num_cols.drop(labels=['Medu', 'Fedu',
'travelttime', 'studytime', 'failures', 'famrel',

'travelttime', 'studytime', 'failures', 'famrel',
'freetime', 'goout',
'Dalc', 'Walc', 'health'])

    object_cols = df.select_dtypes(include='object').columns

    df[object_cols] = df[object_cols].apply(lambda x:
pd.Categorical(x).codes)

    not_binary_object_cols = [col for col in object_cols if
len(df[col].unique()) > 2]

    # Добавляю числовые категориальные переменные

    not_binary_object_cols.extend(['Medu', 'Fedu',
'travelttime', 'studytime', 'failures', 'famrel',

'travelttime', 'studytime', 'failures', 'famrel',
'freetime', 'goout',
'Dalc', 'Walc', 'health'])

    df.loc[:, not_binary_object_cols] =
self.__frequency_encoding(df.loc[:, not_binary_object_cols])

```

```

# Нормализуем числовые столбцы, кроме класса успеваемости
for c in num_cols:
    if c != 'performance':
        df[c] = (df[c] - df[c].min()) / (df[c].max() -
df[c].min())

# Убираем ненужные колонки
df = df.drop(columns=['G1', 'G2', 'G3'])

# Убираю несбалансированность датасета

# Вычисляю во сколько раз класса 2 больше, чем остальных
rat21 = len(df.loc[df['performance'] == 2]) //
len(df.loc[df['performance'] == 1])

rat23 = len(df.loc[df['performance'] == 2]) //
len(df.loc[df['performance'] == 3])

rat20 = len(df.loc[df['performance'] == 2]) //
len(df.loc[df['performance'] == 0])

# Дублирую rat раз классы, которых не хватает
df_1 = df.loc[df['performance'] == 1]
df_1 = df_1.loc[df_1.index.repeat(rat21)]

df_3 = df.loc[df['performance'] == 3]
df_3 = df_3.loc[df_3.index.repeat(rat23)]

df_0 = df.loc[df['performance'] == 0]
df_0 = df_0.loc[df_0.index.repeat(rat20)]

```

```

        df_new = pd.concat([df.loc[df['performance'] == 2],
df_1]).sample(frac=1)

        df_new = pd.concat([df_new, df_3]).sample(frac=1)

        df_new = pd.concat([df_new, df_0]).sample(frac=1)


        # Перемешиваем строки

        df = df_new.iloc[np.random.permutation(len(df_new))]

        return df


@staticmethod
def __frequency_encoding(df):
    new_df = df.copy()

    for c in df.columns:

        mapa = df[c].value_counts().to_dict()

        new_df[c] = df[c].map(mapa)

    return new_df

```

ПРИЛОЖЕНИЕ А.3

Файл main.py

```

from data_prepare import DataPrepare
from student_performance_NN_2 import StudentPerformanceNN2
import random as rand


input_size = 30
output_size = 4
hidden_layers_count = 21
hidden_size = 30
learning_rate = 0.001
epochs = 100


best_accuracy = -1

```

```

best_f1 = -1

print('\n\nPrepare data...')
train_data, test_data =
DataPrepare().get_data('../Portuguese.csv')

while True:
    hidden_size = rand.randint(10, 50)
    hidden_layers_count = rand.randint(10, 70)
    learning_rate = rand.uniform(0.001, 0.0017)

    neural_network = StudentPerformanceNN2(input_size,
output_size, hidden_layers_count, hidden_size, learning_rate,
epochs)

    print('\nStart train...')
    print(
        f'epoch= {epochs}, hidden_size= {hidden_size},
hidden_layers_count={hidden_layers_count}, lr=
{learning_rate}')
    neural_network.train(train_data, test_data, is_log=False)

    print('\nTest model...')
    accuracy, recall, precision, f1, loss =
neural_network.test_model(test_data)

    print(f'Accuracy on test set: {100 * accuracy:.2f}%')
    print(f"F1-score on test set: {100 * f1:.2f}")
    print(f'Best accuracy: {100 * best_accuracy:.2f}%')
    print(f"Best f1-score: {100 * best_f1:.2f}")

    if accuracy >= best_accuracy and f1 > best_f1:
        print('\nSave model...')
        best_accuracy = accuracy
        best_f1 = f1
        neural_network.save_model()

```

```
neural_network.plot_metrics()
```

ПРИЛОЖЕНИЕ А.4

Файл views.py

```
from rest_framework.response import Response

from rest_framework.views import APIView

import random as rand

import pandas as pd

import numpy as np


from predictor.nn import StudentPerformancePredictor

from predictor.serializers import *

from predictor.models import *


class GroupList(APIView):

    @staticmethod

    def get(request):

        groups = ClassGroup.objects.all()

        serializer = ClassGroupSerializer(groups, many=True)

        return Response(serializer.data)


class StudentWithPrediction(APIView):
```

```

def get(self, request):

    group_id = request.query_params['group_id']

    group_students = Student.objects.filter(group=group_id)

    predictor = StudentPerformancePredictor()

    student_performance_predictions = []

    for student in group_students:

        student_data =
StudentData.objects.filter(id=student.data.id)

        prediction = predictor.predict(student_data)

        color = self.__getStudentPredictionColor(prediction)

        true_performance = student.data.performance

        student_performance_predict =
StudentPerformancePrediction(group_id, student.firstName,
student.lastName,

prediction, color, true_performance)

    student_performance_predictions.append(student_performance_predic
t)

    serializer =
StudentPerformancePredictionSerializer(student_performance_predic
tions, many=True)

    return Response(serializer.data)

```

```

    @staticmethod

    def __getStudentPredictionColor(prediction):

        if prediction > 2:

            return '07c120'

        return 'e60000'


class InitDB(APIView):

    first_names = ["Андрей", "Сергей", "Михаил", "Игорь",
                  "Алена", "Виктория", "Мария", "Александр", "Ирина", "Дарья",
                  "Екатерина", "София", "Юлия", "Мария",
                  "Артем"]

    last_names = ["Иванов", "Смирнов", "Сидоров", "Семенов",
                  "Жуков"]

    def get(self, request):

        df = self.__get_prepared_DF()

        for i in range(5):

            group = ClassGroup(number=i + 1, character='A')

            group.save()

            for j in range(15):

```

```
row = j + i * 15

student_data = StudentData(

    school=df.loc[row, 'school'],

    sex=df.loc[row, 'sex'],

    age=df.loc[row, 'age'],

    address=df.loc[row, 'address'],

    famsize=df.loc[row, 'famsize'],

    pstatus=df.loc[row, 'Pstatus'],

    medu=df.loc[row, 'Medu'],

    fedu=df.loc[row, 'Fedu'],

    mjob=df.loc[row, 'Mjob'],

    fjob=df.loc[row, 'Fjob'],

    reason=df.loc[row, 'reason'],

    guardian=df.loc[row, 'guardian'],

    traveltime=df.loc[row, 'traveltime'],

    studytime=df.loc[row, 'studytime'],

    failures=df.loc[row, 'failures'],

    schoolsup=df.loc[row, 'schoolsup'],

    famsup=df.loc[row, 'famsup'],

    paid=df.loc[row, 'paid'],

    activities=df.loc[row, 'activities'],

    nursery=df.loc[row, 'nursery'],
```

```

        higher=df.loc[row, 'higher'],

        internet=df.loc[row, 'internet'],

        romantic=df.loc[row, 'romantic'],

        famrel=df.loc[row, 'famrel'],

        freetime=df.loc[row, 'freetime'],

        goout=df.loc[row, 'goout'],

        dalc=df.loc[row, 'Dalc'],

        walc=df.loc[row, 'Walc'],

        health=df.loc[row, 'health'],

        absences=df.loc[row, 'absences'],

        performance=df.loc[row, 'performance'])

    student_data.save()

    student = Student(firstName=self.first_names[j],
lastName=self.last_names[rand.randint(0, 4)],

                        group=group,

                        data=student_data)

    student.save()

students = Student.objects.all()

serializer = StudentSerializer(students, many=True)

return Response(serializer.data)

def __get_prepared_DF(self):

```

```

df = pd.read_excel('predictor/Portuguese.csv')

# Создаю столбец успеваемости и делю столбец G3 на
категории успеваемости

df['performance'] = pd.cut(df['G3'], bins=[-1, 5, 10, 15,
20], labels=False)

num_cols = df.select_dtypes(include='int64').columns

num_cols = num_cols.drop(labels=['Medu', 'Fedu',
'travelttime', 'studytime', 'failures', 'famrel',

                                'freetime', 'goout',
'Dalc', 'Walc', 'health'])

object_cols = df.select_dtypes(include='object').columns

df[object_cols] = df[object_cols].apply(lambda x:
pd.Categorical(x).codes)

not_binary_object_cols = [col for col in object_cols if
len(df[col].unique()) > 2]

# Добавляю числовые категориальные переменные

not_binary_object_cols.extend(['Medu', 'Fedu',
'travelttime', 'studytime', 'failures', 'famrel',

                                'freetime', 'goout',
'Dalc', 'Walc', 'health'])

```

```

        df.loc[:, not_binary_object_cols] =
self.__frequency_encoding(df.loc[:, not_binary_object_cols])

# Нормализуем числовые столбцы, кроме класса успеваемости

for c in num_cols:

    if c != 'performance':

        df[c] = (df[c] - df[c].min()) / (df[c].max() -
df[c].min())

# Убираем ненужные колонки

df = df.drop(columns=['G1', 'G2', 'G3'])

# Перемешиваем строки

df = df.iloc[np.random.permutation(len(df))]

df = df.reset_index(drop=True)

return df

@staticmethod

def __frequency_encoding(df):

    new_df = df.copy()

    for c in df.columns:

        mapa = df[c].value_counts().to_dict()

```

```
new_df[c] = df[c].map(mapa)

return new_df
```

ПРИЛОЖЕНИЕ A.5

Файл nn.py

```
import torch

import torch.nn as nn

import pandas as pd

import numpy as np


class StudentPerformancePredictor:

    def __init__(self, in_size=30, hidden_size=39,
hidden_layers_count=11, output_size=4):

        self.model = self.__create_model(in_size, hidden_size,
hidden_layers_count, output_size)

    @staticmethod

    def __create_model(input_size, hidden_size,
hidden_layers_count, output_size):

        layers = [nn.Linear(input_size, hidden_size), nn.ReLU()]

        for _ in range(hidden_layers_count - 1):

            layers.append(nn.Linear(hidden_size, hidden_size))
```

```
layers.append(nn.ReLU())
```

```
layers.append(nn.Linear(hidden_size, output_size))
```

```
model = nn.Sequential(*layers)
```

```
model.load_state_dict(torch.load("predictor/nn_models/model_a841_
f1838_e100_hs39_hlc11_lr0.0012_is30_os4.pt"))
```

```
model.eval()
```

```
return model
```

```
def predict(self, student_data):
```

```
    df_tensor = self.__get_inputs(student_data)
```

```
    with torch.no_grad():
```

```
        outputs = self.model(df_tensor)
```

```
        _, predicted = torch.max(outputs.data, 1)
```

```
    return predicted
```

```
@staticmethod

def __get_inputs(student_data):

    data = list(student_data.values())

    df = pd.DataFrame(data)

    df = df.drop(columns=['performance', 'id'])

    df_np = df.to_numpy()

    df_tensor = torch.from_numpy(df_np.astype(np.float32))

    return df_tensor
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Цифровой помощник педагога

Презентация