Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Разработка архитектуры глубокого обучения способной обрабатывать табличные данные для решения задач мета-обучения

Обучающийся / Student Гарипов Роман Исмагилович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group M34381
Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика
Образовательная программа / Educational program Информатика и программирование 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Руководитель ВКР/ Thesis supervisor Забашта Алексей Сергеевич, кандидат технических наук, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "ординарный доцент")

Обучающийся/Student

Документ подписан
Гарипов Роман Исмагилович
15.05.2023

(эл. подпись/ signature)

Гарипов Роман Исмагилович
(Фамилия И.О./ name and surname)

Руководитель ВКР/ Thesis supervisor

Документ подписан
Забашта Алексей Сергеевич
15.05.2023

(эл. подпись/ signature)

Забашта Алексей Сергеевич
(Фамилия И.О./ name and surname)

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ / OBJECTIVES FOR A GRADUATION THESIS

Обучающийся / Student Гарипов Роман Исмагилович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group M34381
Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика
Образовательная программа / Educational program Информатика и программирование 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка архитектуры глубокого обучения способной обрабатывать табличные данные для решения задач мета-обучения
Руководитель ВКР/ Thesis supervisor Забашта Алексей Сергеевич, кандидат технических наук, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "ординарный доцент")

Основные вопросы, подлежащие разработке / Key issues to be analyzed
Требуется разработать архитектуру глубокого обучения способную обрабатывать табличные данные для решения задач мета-обучения. В рамках этого необходимо:

- Разработать способ обработки табличного набора данных нейронной сетью, предположительно основываясь на методе Deep Sets

- Разработать архитектуру глубокого обучения, способную обрабатывать табличные наборы данных с применением разработанного способа обработки табличного набора данных

- Реализовать разработанную архитектуру глубокого обучения на языке Python с использованием библиотеки глубокого обучения PyTorch, провести эксперименты

- Сравнить разработанную архитектуру с современными решениями задачи мета-обучения

Форма представления материалов ВКР / Format(s) of thesis materials:
программный код, презентация, пояснительная записка

Дата выдачи задания / Assignment issued on: 01.03.2023

Срок представления готовой ВКР / Deadline for final edition of the thesis 15.05.2023

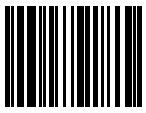
Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: да / yes
Тема в области прикладных исследований / Subject of applied research: да / yes

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/ Thesis supervisor

Документ подписан
Забашта Алексей Сергеевич
09.05.2023

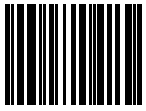


(эл. подпись)

Забашта Алексей Сергеевич

Задание принял к исполнению/ Objectives assumed BY

Документ подписан
Гарипов Роман Исмагилович
09.05.2023

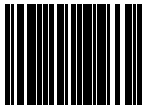


(эл. подпись)

Гарипов Роман Исмагилович

Руководитель ОП/ Head of educational program

Документ подписан
Станкевич Андрей Сергеевич
22.05.2023



(эл. подпись)

Станкевич Андрей Сергеевич

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

АННОТАЦИЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ SUMMARY OF A GRADUATION THESIS

Обучающийся / Student Гарипов Роман Исмагилович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group M34381
Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика
Образовательная программа / Educational program Информатика и программирование 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка архитектуры глубокого обучения способной обрабатывать табличные данные для решения задач мета-обучения
Руководитель ВКР/ Thesis supervisor Забашта Алексей Сергеевич, кандидат технических наук, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "ординарный доцент")

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ DESCRIPTION OF THE GRADUATION THESIS

Цель исследования / Research goal
Цель работы - исследование и разработка архитектуры глубокого обучения, способной обрабатывать табличные данные способом, инвариантным к перестановке строк и столбцов для решения задач мета-обучения
Задачи, решаемые в ВКР / Research tasks
Задачи работы: - Разработка способа обработки табличного набора данных нейронной сетью, предположительно основываясь на методе Deep Sets - Разработка архитектуры глубокого обучения, способной обрабатывать табличные наборы данных с применением разработанного способа обработки табличного набора данных - Реализация разработанной архитектуры глубокого обучения на языке Python с использованием библиотеки глубокого обучения PyTorch, проведение экспериментов - Сравнение разработанной архитектуры с современными решениями задачи мета-обучения
Краткая характеристика полученных результатов / Short summary of results/findings
Разработан способ обработки табличного набора данных нейронной сетью, обрабатывающий таблицы как множества множеств. Разработана архитектуры глубокого обучения, способная обрабатывать табличные наборы данных. Разработанная архитектура реализована на языке Python с использованием библиотеки глубокого обучения PyTorch. Разработанная архитектура демонстрирует прирост качества относительно современных

аналогов в задаче мета-обучения.

Наличие выступлений на конференциях по теме выпускной работы / Conference reports on the topic of the thesis

1. XII Конгресс молодых ученых ИТМО, 03.04.2023 - 06.04.2023 (Конференция, статус - всероссийский)

Обучающийся/Student

Документ подписан
Гарипов Роман Исмагилович
15.05.2023

(эл. подпись/ signature)

Гарипов Роман Исмагилович
(Фамилия И.О./ name and surname)

Руководитель ВКР/ Thesis supervisor

Документ подписан
Забашта Алексей Сергеевич
15.05.2023

(эл. подпись/ signature)

Забашта Алексей Сергеевич
(Фамилия И.О./ name and surname)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработка систем машинного обучения – это сложный процесс, который состоит из нескольких этапов. В общем случае, разработка систем машинного обучения включает в себя следующие этапы:

а) Сбор данных и разметка данных: этот этап включает в себя сбор и разметку данных, которые будут использоваться для обучения. Это могут быть как структурированные данные, так и неструктурированные данные, такие как тексты, изображения и видео.

б) Предварительный анализ и обработка данных: анализ данных заключается в изучении набора данных, первичном осмотре данных, обнаружении закономерностей, которые могут послужить основанием для выбора конкретной модели. На этом этапе данные проходят через предварительную обработку, которая может включать в себя очистку данных, заполнение пропусков, фильтрацию шумов, нормализацию и т.д.. Этот этап важен для того, чтобы данные были готовы для использования в обучении.

в) Выбор модели: на этом этапе выбирается модель машинного обучения, которая будет использоваться для обучения и дальнейшего использования в разрабатываемой системе. Это может быть любая модель машинного обучения, включая нейронные сети, деревья решений, метод опорных векторов и другие.

г) Обучение модели: этот этап включает в себя обучение выбранной модели на собранных данных. Обучение может проходить с помощью методов обучения с учителем, без учителя или с подкреплением, в зависимости от характера данных и задачи, которую нужно решить.

д) Оценка модели: после того, как модель обучена на данных, она оценивается с помощью метрик качества на отложенной выборке набора данных. Оценка модели позволяет определить, насколько хорошо модель справляется с поставленной задачей.

е) Интеграция модели: после того, как модель была протестирована и оценена, она интегрируется в систему, которая будет использовать ее для решения задачи, для которой она была разработана.

Каждый из этих этапов это отдельная и сложная задача. Выбор модели это отдельная специфичная область. Алгоритмы машинного обучения отличаются

друг от друга, некоторым алгоритмам важно чтобы признаки объектов были независимы, некоторые алгоритмы сильно зависят от распределения признаков. Поэтому важно правильно подбирать под каждую отдельную задачу алгоритм машинного обучения. Цель мета-обучения – обучить такую модель, которая по набору данных сможет определить алгоритм, который сможет добиться максимальной точности на наборе данных не запуская его. Алгоритмы мета-обучения позволяют ускорить процесс разработки систем машинного обучения и искусственного интеллекта, так как появляется возможность автоматически по набору данных предсказывать, какой алгоритм лучше с точки зрения метрики качества подойдет для конкретного набора данных не запуская сам алгоритм на наборе данных. В мета-обучении используются как и простые методы, основанные на обучении моделей, способных по статистическим и структурным признакам набора данных определять наиболее точный алгоритм, так и более сложные методы, например глубокие нейронные сети, в том числе генеративно-состязательные.

Табличные наборы данных обладают свойством инвариантности относительно перестановки строк и столбцов. Если изменить порядок строк и столбцов в наборе данных, то скрытая зависимость в этих данных не изменится, классические алгоритмы машинного обучения по-прежнему будут получать такие же результаты как и без перестановки. Это свойство важно учитывать при разработке алгоритмов мета-обучения. Классический подход мета-обучения [16] извлекает признаки, называемые мета-признаками, из набора данных не обращая внимания на порядок строк и столбцов. Однако, этот подход извлекает фиксированные статистические и структурные признаки.

В настоящее время, на практике глубокие нейронные сети демонстрируют способность извлекать сложные скрытые зависимости в данных и формировать признаки, иногда тяжело интерпретируемые для человека. Предложенный в работе алгоритм мета-обучения Deep Table применяет глубокую нейронную сеть, использующую инвариантность табличных наборов данных к перестановке строк и столбцов для извлечения зависимостей и формирования признаков табличных наборов данных. Существует подход использующий генеративно-состязательные сети [6] для задачи мета-обучения [5]. В генеративно-состязательном подходе одновременно обучаются две модели: генеративная модель G которая фиксирует распределение данных, и дискрими-

национная модель D, которая оценивает вероятность того, что выборка получена из обучающих данных, а не из G. Дискриминатор в архитектуре LM-GAN является свёрточной нейронной сетью. К табличному набору применяется алгоритм стабилизации, приводящий табличные наборы данных равные с точностью до перестановки строки и столбцов к одному и тому же виду. Далее стабилизированный набор данных подается на вход дискриминатору. Свёрточная сеть, используемая в дискриминаторе, задает ограничения на размер табличных данных, которые могут обрабатываться этой сетью. Помимо этого, этот подход требует выбора и применения алгоритма стабилизации, от которого зависит качество итоговой модели. Предложенный же подход Deep Table фокусируется на обработке табличных наборов как множеств, состоящих из множеств, поэтому не фиксирует размер табличного набора данных. В Deep Table применяются обучаемые функции, извлекающих мета-признаки так же, как это делается в классическом подходе, но не фиксируются сами функции, они подбираются в процессе обучения нейронной сети.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ МЕТА-ОБУЧЕНИЯ И ОБЗОР РЕШЕНИЙ

1.1. Постановка задачи

В машинном обучении существует задача выбора алгоритма [14]. Формулируется она следующим образом: задано множество алгоритмов машинного обучения $\mathbb{A}$, набор данных D, который состоит из признаков X объектов и значений целевых переменных объектов y, и функция ошибки $\mathcal{L}$. Необходимо выбрать такой алгоритм

$$a_{best} = \underset{a \in \mathbb{A}}{\operatorname{argmin}} \mathcal{L}(a(X), y)$$

Задача мета-обучения заключается в выборе такого алгоритма без запуска самих алгоритмов на наборе данных.

1.2. Обзор существующих решений

Все рассмотренные решения используют заранее отобранные наборы данных, на которых модели обучаются выбирать оптимальный для набора данных алгоритм машинного обучения.

1.2.1. Классический мета-признаковый подход

Идея этого метода заключается в формировании векторного представления табличного набора данных в виде так называемых мета-признаков. Как уже было отмечено выше, табличный набор данных обладает свойством инвариантности относительно перестановки строк и столбцов. Это означает следующее: если изменить порядок строк и/или столбцов, то информация содержащаяся в табличном наборе данных, зависимости и закономерности в нем не изменятся. Большинство алгоритмов машинного обучения в том или ином виде поддерживают этот инвариант. Например, если обучить линейную регрессию на каком-то табличном наборе данных, переставить в нем строки и столбцы, и обучить регрессию на полученном наборе данных, то линейная регрессия выучит точно такую же функций зависимости признаков от целевой переменной с точностью до перестановки признаков/коэффициентов в линейной регрессии. Имеется в виду обучение линейной регрессии аналитическим способом.

1.2.1.1. Базовые мета-признаки

Базовые мета-признаки табличных наборов данных это число объектов, чисто признаков, доля категориальных признаков и т.д.

1.2.1.2. Статистические мета-признаки

Статистические мета-признаки, это статистики извлеченные из столбцов и их агрегаций функцией, не зависящей от порядка аргументов. Например, выборочное среднее, медиана, мода, выборочная дисперсия и т.д. Первичная аггрегирующая функция применяется ко всем столбцам табличного набора данных, это первичная агрегация. К результатам первичной агрегации применяется вторичная агрегирующая функция, точно так же независящая от порядка аргументов, ее результатом является одно число, являющееся мета-признаков. Таким образом комбинируя различные первичные и вторичные агрегирующие функции, получается мета-признаковое описание табличного набора данных. Важно, чтобы агрегирующие функции были не зависели от порядка аргументов, так как необходимо табличные наборы данных равные с точностью до перестановки строк и/или столбцов, представлять одинаковыми мета-признаками.

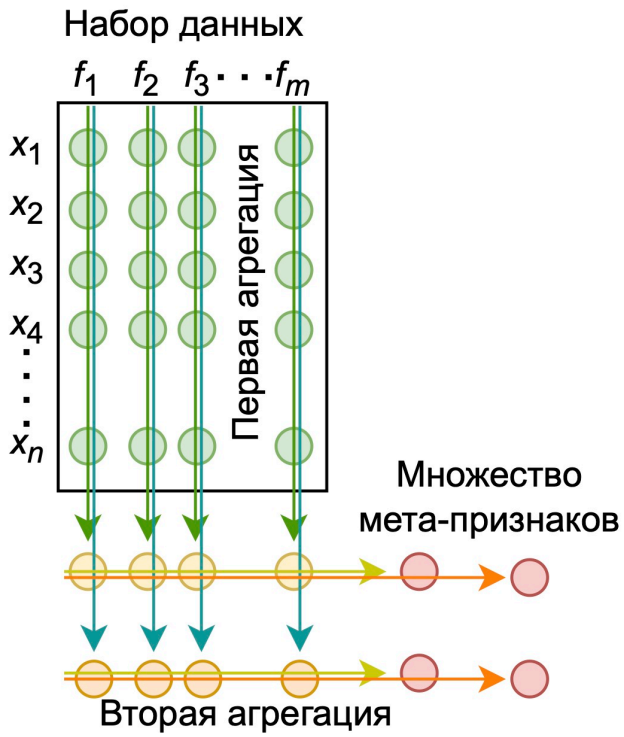


Рисунок 1 – Схема построения мета-признаков

1.2.1.3. Структурные мета-признаки

Структурные мета-признаки это параметры модели, обученной на табличном наборе данных. Например, параметры дерева решений обученного на табличном наборе данных. Структурными признаками в таком случае будут параметры дерева: глубина каждого листа, количество вершин на определенной высоте дерева, число объектов в каждом листе и любые другие характеристики построенного дерева. Так это это могут быть, например, параметры обученной линейной/логистической регрессии, параметры нейронной сети обученной на табличном наборе данных и т.д..

По построенным мета-признакам предполагается обучить модель, предсказывающую самый точный алгоритм из множества A или точность каждого алгоритма. Вычислим мета-признаки каждого табличного набора данных из обучающей выборки. Для того, чтобы получить целевую переменную, необходимо запустить на каждом табличном наборе данных все алгоритмы из A. Таким образом общая схема такая: составляется мета-признаковое описание и вычисляется целевая переменная для каждого табличного набора данных и обучается модель, которая будет решать задачу мета-обучения.

Недостаток этого подхода заключается в его простоте и в том, что для построения мета-признаков применяются фиксированные функции агрегации. Далее будет экспериментально показано, что если использовать обучаемые функции, для извлечения признаков, точность предсказания самого точного алгоритма возрастет.

1.2.2. Нейронная сеть LM-GAN

Это обусловленная генеративно-состязательная архитектура. В ней есть генератор и дискриминатор. Генератор принимает так называемое условие, которому должны соответствовать сгенерированные данные, и шум, из которого генерируются данные. Генератор генерирует табличные наборы данных с заданными условием, условие передается в генератор в виде мета-признаков. Таким образом, генератор в этой архитектуре генерирует табличный набор данных с заданными мета-признаками. Задача дискриминатора заключается в распознавании сгенерированных данных, то есть он должен отличать реальные от сгененирированных. Дискриминатор получает на вход набор данных, возвращает по нему уверенность в том, что эти данные реальные, а так же

лямбда-вектор. Это вектор, в котором каждый элемент равняется уверенности модели в том, что соответствующий алгоритм является оптимальным на наборе данных.

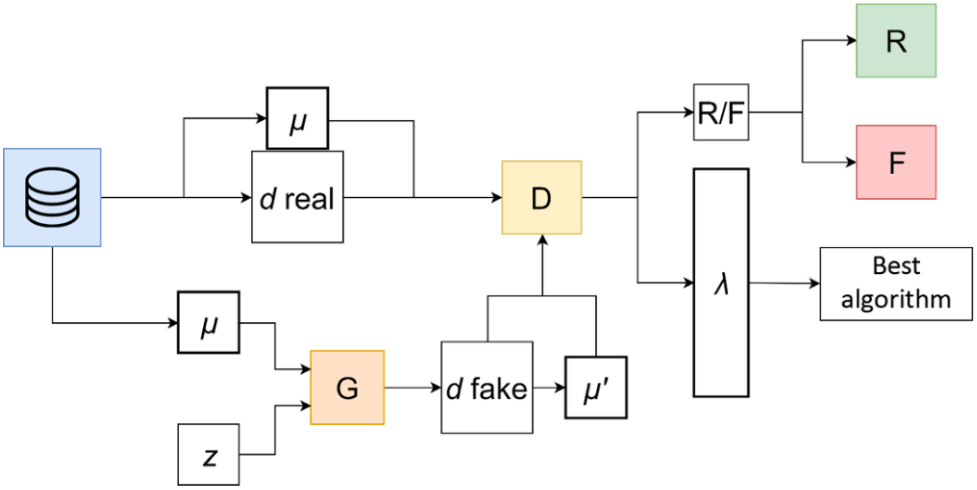


Рисунок 2 – Архитектура LM-GAN [5]

Эта архитектура обучается в генеративно-состязательной постановке [6]. И сочета ет в себе идеи CGAN [9], поскольку является так называемой обусловленной моделью и DCGAN [13], поскольку применяются свёрточные слои.

Так же, дискриминатор в LM-GAN архитектуре обладает достаточно большим количеством параметров и может обрабатывать только табличные наборы данные фиксированного размера.

Однако, перед обработкой табличного набора данных в дискриминаторе, к нему применяется так называемый алгоритм стабилизации. Его смысл заключается в том, чтобы привести табличные наборы данных равные с точности до перестановки строк и/или столбцов к одному и тому же табличному набору данных. После стабализации, табличный набор данных подается на вход дискриминатору, который применяет операцию свертки к табличному набору данных.

Табличный набор данных в каком-то смысле обрабатывается как картинка, представляющая класс эквивалентных с точностью до перестановки табличных наборов данных, то есть эта особенность табличных наборов данных используется искусственным образом.

1.3. Модель Deep Sets

Для обработки нейронными сетями множеств существует модель предложенная в статье Deep Sets [2]. Слои, предложенные в этой работе будут использоваться в разработанной архитектуре, поэтому необходимо их описать.

В этой работе предложено 2 слоя нейронной сети.

$$\text{Invariant Model:} \quad f(X) = p(\sum_{x \in X} \phi(x))$$

$$\text{Equivariant Model:} \quad f(x) = \sigma(\phi(x) + \gamma \cdot maxpool(X))$$

Invariant model дает нам способ применить агрегацию множества в число, просуммировав значения преобразований элементов множества с последующим применением функции p, которая, например, может производить нормировку итогового значения. N множеств размера M, то есть матрица размера (N,M), после применения функции Invariant model будет иметь вид вектора столбца размера N. Если же каждый элемент множества представлен в виде вектора размера H_0, то есть имеем дело с тензором размера (N,M,H_0), в результате применения функции получим матрицу размера (N,H_0).

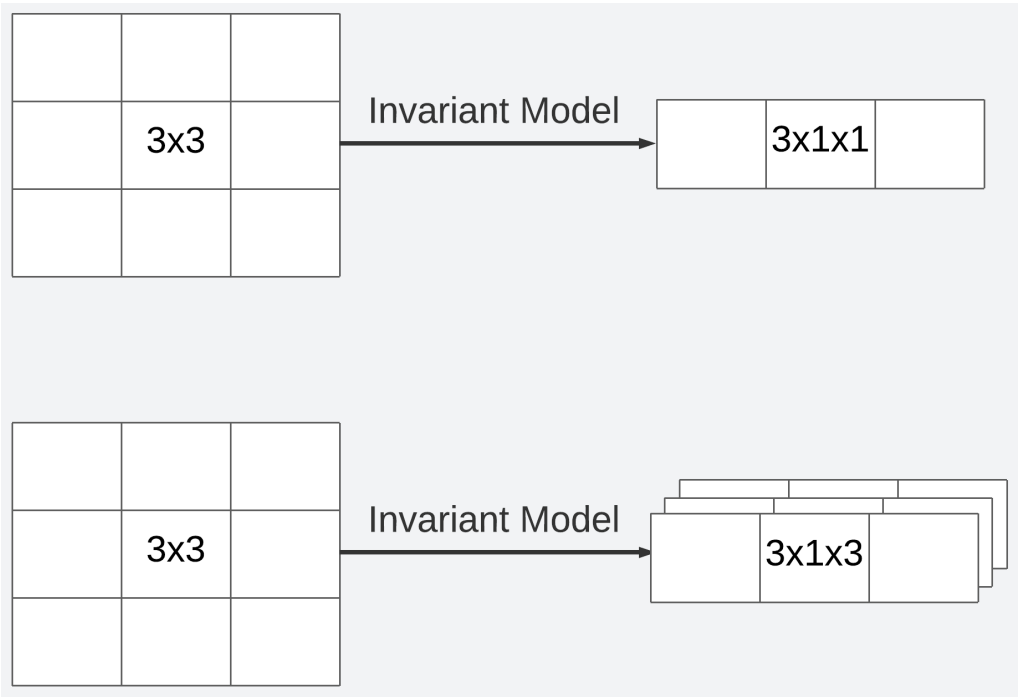


Рисунок 3 – Пример применения Invariant Model

Equivariant model это способ преобразования элементов множества. К элементу множества применяется преобразование ϕ, складывается с макси-

мальным элементом множества, умноженным на параметр γ, к сумме применяется функция активации. Таким образом, применив функцию Equivariant model к N множествам размера M, то есть к матрице размера (N,M), получится матрица такого же размера. Применим эту функцию к N множествам размера M, каждый элемент множества представлен в виде вектора размера H_0, то есть тензор размера (N,M,H_0), получится тензор размера (N,M,H_1).

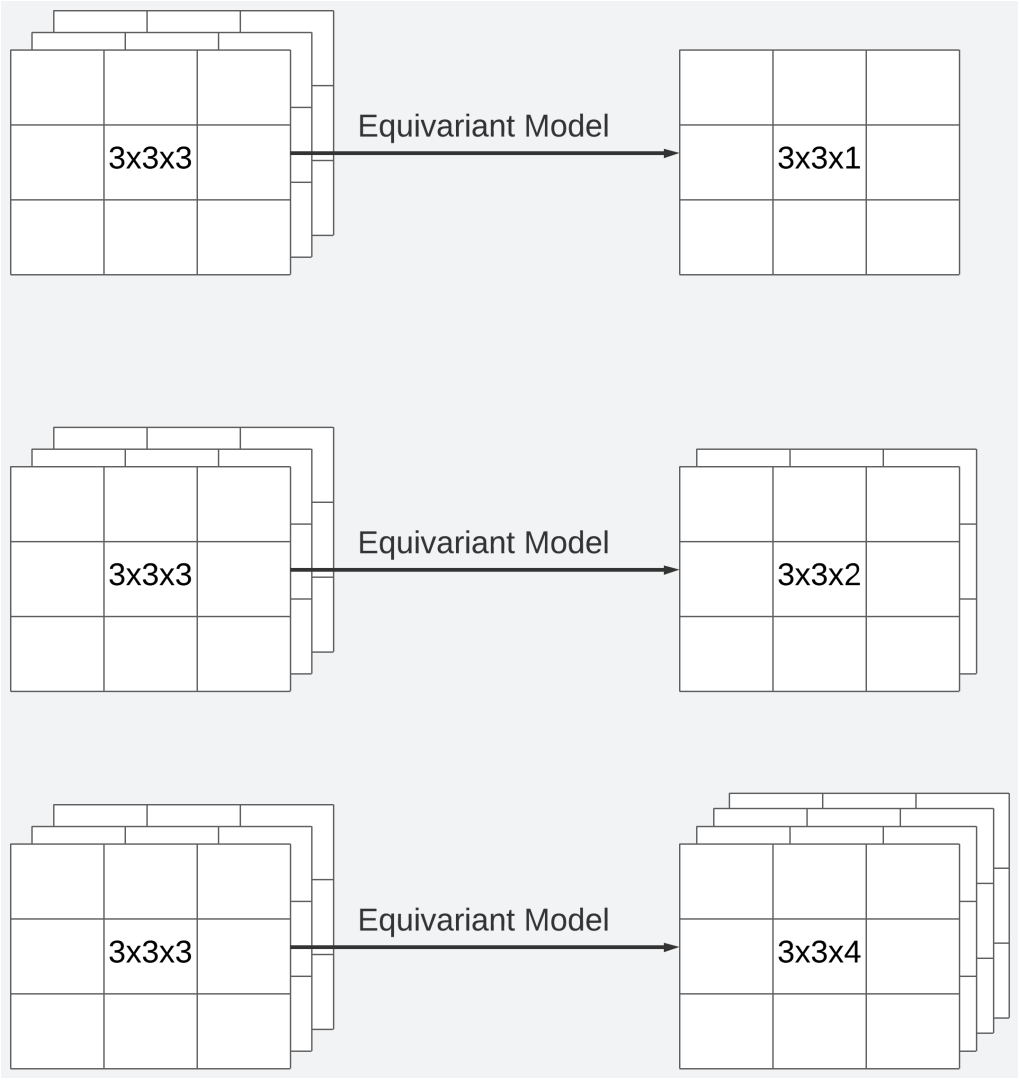


Рисунок 4 – Пример применения Equivariant Model

Стоит отметить, что преобразование ϕ может представляет собой линейную операцию, то есть умножение на константу в случае, когда элементы множества представлены в виде чисел, и умножение на матрицу в случае, когда элементы множества представлены в виде векторов. Обучаемые в этих модулях будут именно множители.

Подводя итоги, размерности тензоров после обработки этими слоями можно задать следущей схемой:
Invariant model:

— $(N, M) \to (N)$

— $(N, M, H_0) \to (N, H_1)$

Equivariant model:

— $(N, M) \to (N, M)$

— $(N, M, H_0) \to (N, M, H_1)$

1.4. Выводы по обзору

Классический подход обрабатывает табличные наборы данных независимо от порядка строк и/или столбцов в них, но при этом он достаточно прост, вычисляются конкретные базовые, статистические и структурные признаки.

LM-GAN решает не только задачу мета-обучения, но еще и задачу генерации наборов данных с заданными свойствами. Для этого применяются глубокие нейронные сети, которые позволяют достигать точности гораздо выше чем у классического подхода. Однако минусами является то, что есть необходимость применять алгоритм стабилизации, от выбора которого зависит точность модели и то, что табличный набор данных обрабатывается свёрточной нейронной сетью, принято считать, что свертки предназначены для картинок.

Таким образом, становятся понятны требования, которые можно предъявить к разрабатываемому решению задачи мета-обучения:

а) Разработанная архитектура должна применять глубокое обучение и нейронные сети к задаче мета-обучения.

б) Разработанная архитектура должна обрабатывать табличные наборы данных независимо от порядка строк и столбцов, например обрабатывать в качестве множеств, содержащих множества, как это делают агрегирующие функции в классическом подходе.

в) Разработанная архитектура должна уметь работать с табличными наборами данных различных размеров.

г) Разработанная архитектура не должна уступать в точности уже существующим решениям, в частности LM-GAN и классическому подходу.

1.5. Задача мультиклассификации

В машинном обучении классификация с несколькими метками или классификация с несколькими выходами — это вариант задачи классификации, ко-

гда каждому экземпляру может быть назначено несколько неисключительных меток. Классификация с несколькими метками является обобщением классификации с несколькими классами, которая представляет собой проблему с одной меткой отнесения экземпляров точно к одному из нескольких (более двух) классов. В задаче с несколькими метками метки не являются исключительными, и нет ограничений на количество классов, которым может быть присвоен экземпляр.

Формально классификация с несколькими метками — это проблема поиска модели, которая отображает входные данные x в двоичные векторы y, то есть каждому элементу (метке) в y присваивается значение 0 или 1.

В нашем случае будет рассмотрена именно такая задача классификации в рамках задачи мета-обучения, так как допустим случай, что на табличном наборе данных два алгоритма машинного обучения достигают одинаковых результатов.

Выводы по главе 1

В данной главе проведен обзор современных методов, используемых для решения задачи мета-обучения. Описаны слои нейронных сетей предложенных в статье Deep Sets. Описана постановка задачи мета-обучения, некоторые базовые понятие мета-обучения, такие как мета-признаки. Описаны недостатки современных методов мета-обучения. Сформулированы обоснования необходимости разработки новых решений и обозначены требования, предъявляемые к разрабатываемому решению задачи мета-обучения.

ГЛАВА 2. ПРЕДЛОЖЕННОЕ РЕШЕНИЕ DEEP TABLE

В этой главе будет подробно описана разработанная архитектура. Так же будет приведена общая схема архитектуры предложенного решения для общего случай табличных наборов данных.

Разработанная архитектура называется Deep Table. Как можно понять из описания слоев Invariant Model и Equivariant model, комбинируя их можно сконструировать глубокую нейронную сеть. Invariant Model позволяет агрегировать множество в число, а Equivariant model позволяет применять преобразование ко всем элементам множества. Таким образом, в ходе изучения статьи в которой предложены эти слои, появилась идеи применить эти слои для агрегации табличного набора данных так же, как это делается в классическом подходе мета-обучения. Сначала преобразуем каждый столбец как множество, применим к нему операцию Equivariant Model, увеличив размер вектора, который представляет каждый элемент столбца. Далее применим Invariant Model, чтобы сагрегировать каждое множество в вектор. Получим таким образом, что каждый столбец представлен несколькими агрегатами. Далее преобразуем эти векторы-агрегаты операцией Equivariant Model. А далее опять применим Invariant Model, чтобы сагрегировать множество векторов в один финальный вектор, который и будет является вектором глубоких мета-признаков. После каждого преобразования Equivariant Model будем применять функцию активацию ReLU [4] в лучших традициях глубокого обучения. В современном глубоком обучении принято между линейными слоями добавлять функцию активацию ReLU. А наши Invariant Model и Equivariant Model как раз представляют из себя линейные полносвязные слои, там производится умножение на обучаемую матрицу и суммирование в вектором сдвига. Таким образом мы получим мета-признаки которые далее в тексте будут называться глубокими. Получив вектор признаков, представляющий табличный набор данных, подадим его на вход классификатору, который решает задачу мультиклассификации и на выходе имеет вектор, где каждый элемент равен уверенности модели в том, что соотвествующий алгоритм достигает наилучшей точности на заданном наборе данных.

Общая схема предложенной архитектуры:

а) По набору данных вычисляются мета-признаки

б) Набор данных подается в блок нейронный сети, который агрегирует табличный набор данных в вектор глубоких мета-признаков
в) Вектор глубоких мета-признаков сконкатенированный с вектором мета-признаков подается на вход классификатору

Модель обучается в постановке задачи классификации.

Поскольку, для обучения необходимо в несколько эпох проходится по всем табличным наборам данных из обучающей выборки, причем по каждому набору данных столько раз, сколько эпох отводится на обучение, логично вычислить один раз заранее мета-признаки для каждого набора данных и сохранить их в файл, чтобы не вычислять каждый раз заново. Точно так же можно поступить с лямбда-вектором каждого табличного набора данных, то есть с целевой переменной каждого объекта нашей обучающей выборки. Предпосчитаем заранее для каждого табличного набора данных из обучающей выборки, какой алгоритм является на нем оптимальным с точки зрения точности. Поставим 1 в компоненту вектора, соответствующую самому точному алгоритму, в остальные 0. Если таких алгоритмов несколько, то есть у нескольких алгоритмов равны точности, то поставим 1 в компоненты всем этим алгоритмам.

В ходе работы было разработано несколько похожих архитектур, рассмотрим детально каждую из них, а так же схему архитектуры для общего случая.

2.1. Нейронная сеть Deep Table

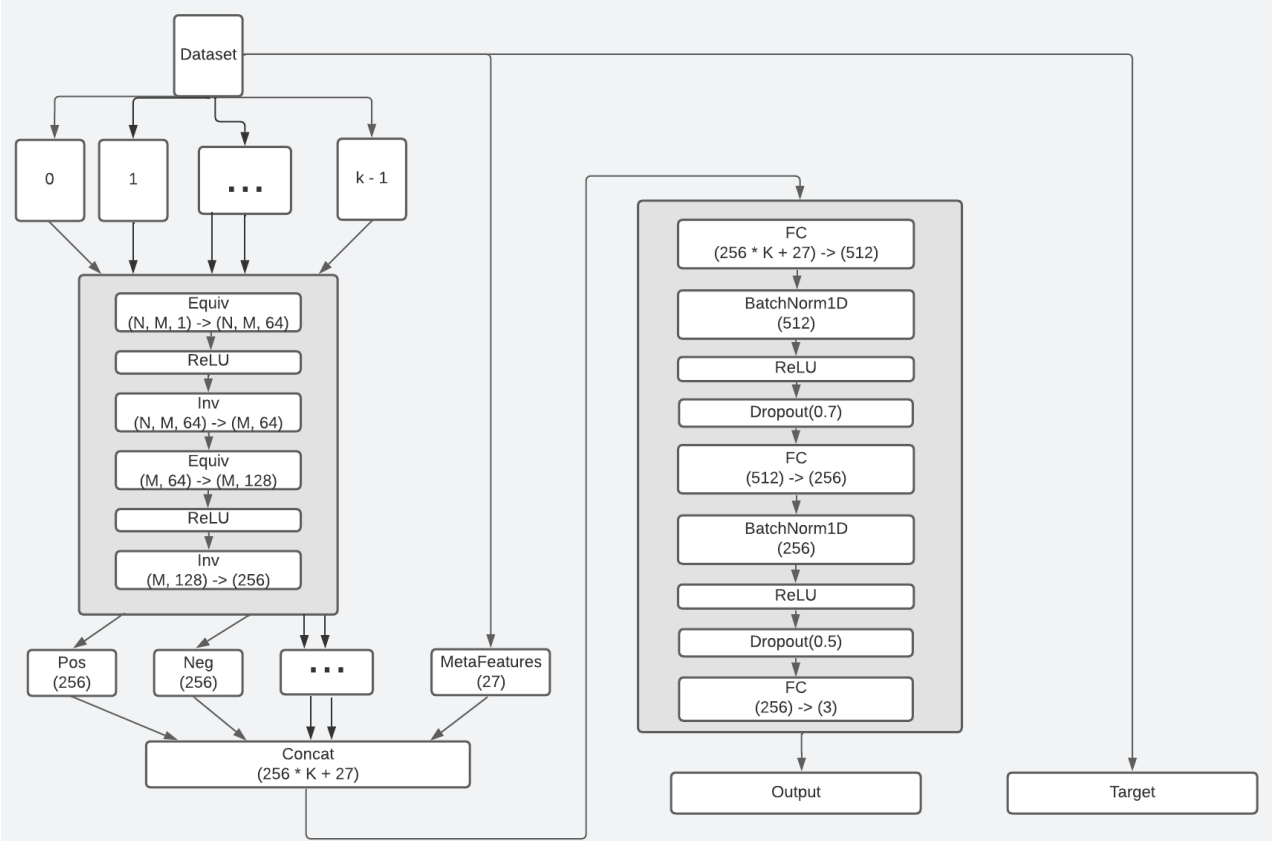


Рисунок 5 – Общая схема архитектуры Deep Table

В общем случае, табличный набор данных подается по отдельности в часть нейронной сети, отвечающую за агрегацию табличного набора данных в вектор глубоких мета-признаков. Таким образом, формируется вектор глубоких мета-признаков для каждого класса, они все вместе конкатенируются вместе с вектором мета-признаков извлеченным классическим способом. Далее подаются на вход классификатору, который выглядит как классический нейросетевой классификатор, состоящий из чередующихся линейных слоев, батчевой нормализации [7], функции активации ReLU [4] и слоя Dropout [3]. При этом, размеры скрытых состояний уменьшаются по мере прохода по слоям классификатора.

Эта архитектура удовлетворяет требованиям, определенным в прошлой главе, так как применяет нейронную сеть и глубокое обучение для решения задачи мета-обучения, агрегирует табличный набор данных способом, инвариантным к перестановке строк и/или столбцов табличного набора данных, может обрабатывать наборы данных произвольного размера, так как все операции с табличным набором данных производятся как будто бы это множество.

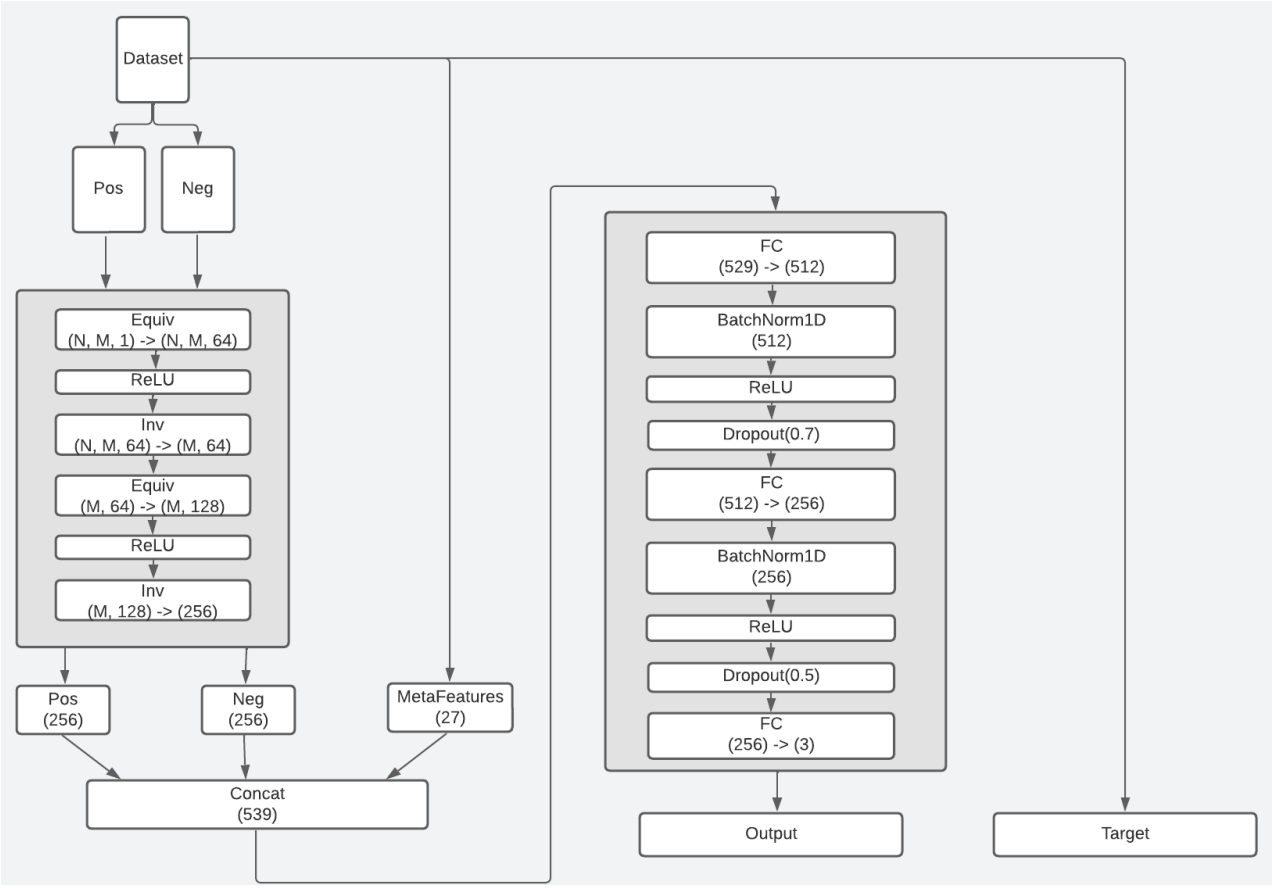


Рисунок 6 – Архитектура Deep Table для описанной задачи мета-обучения

На изображении представлена архитектура Deep Table для частного случая, когда табличные наборы данных представлены двумя классами. Эта архитектура удовлетворяет требованиям, определенным в прошлой главе, так как применяет нейронную сеть и глубокое обучение для решения задачи мета-обучения, агрегирует табличный набор данных способом, инвариантным к перестановке строк и/или столбцов табличного набора данных, может обрабатывать наборы данных произвольного размера, так как все операции с табличным набором данных производятся как будто бы это множество.

2.2. Нейронная сеть Deep Table Multiple Extractors

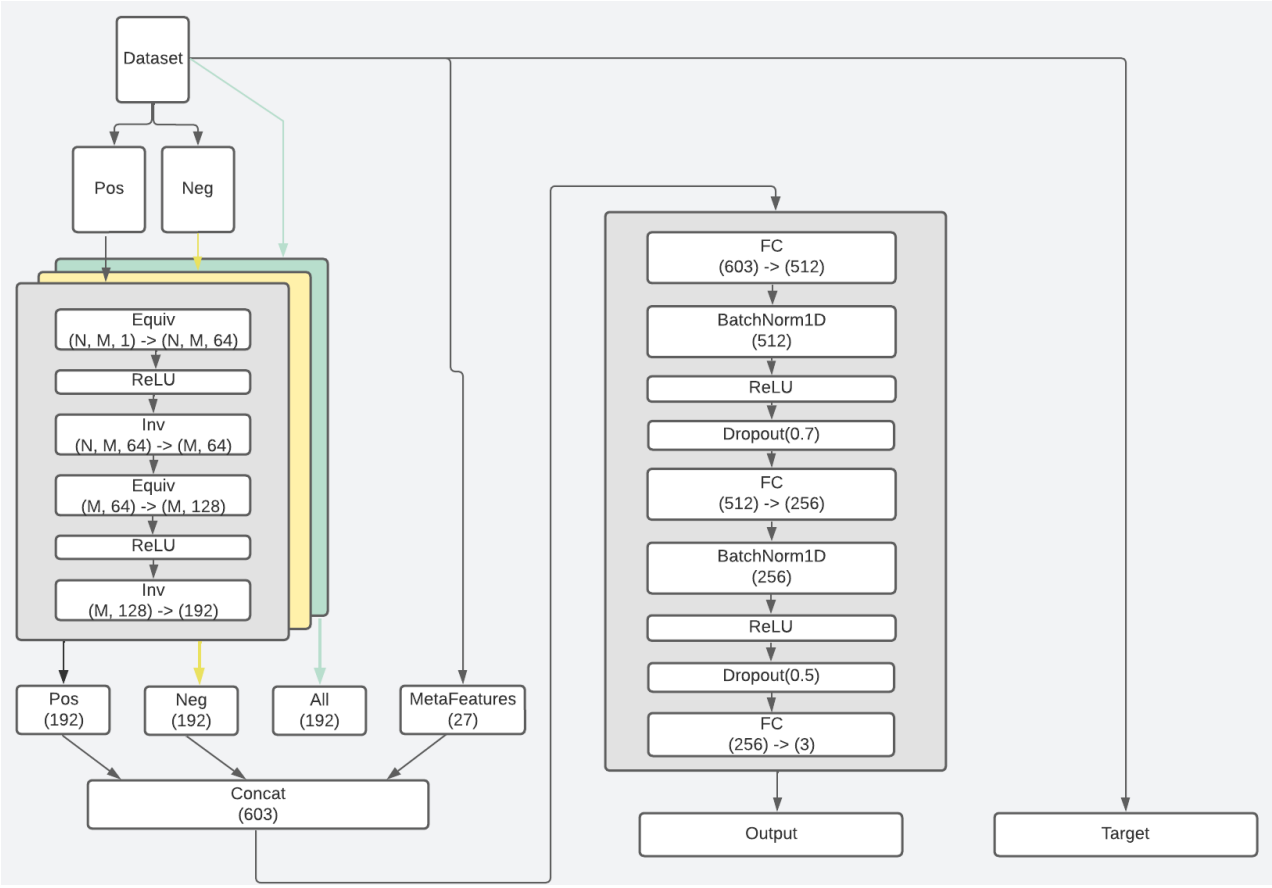


Рисунок 7 – Архитектура Deep Table

Эта архитектура отличается от предыдущей тем, что для каждого класса, а так же для всего набора данных целиком, используются отдельные блоки нейронной сети, агрегирующие табличный набор данных в вектор глубоких мета-признаков. Причем эти блоки абсолютно идентичные, однако инициализация параметров происходит случайная, поэтому все они выучат различные зависимости в процессе обучений нейронной сети. Точно так же вектора глубоких мета-признаков конкатенируются все вместе, Эта архитектура удовлетворяет требованиям, определенным в прошлой главе, так как применяет нейронную сеть и глубокое обучение для решения задачи мета-обучения, агрегирует табличный набор данных способом, инвариантным к перестановке строк и/или столбцов табличного набора данных, может обрабатывать наборы данных произвольного размера, так как все операции с табличным набором данных производятся как будто бы это множество.

2.3. Нейронная сеть Deep Table Flattened

В этом подходе, табличный набор данных представляется в виде одного вектора, то есть все его строчки записываются подряд в вектор. И подаются на вход последовательности Eqiuvariant Model, Invariant Model. И из них получается вектор глубоких мета-признаков, вычисленный из одного множества, где каждый элемент представлен векторов, размер которого равен размеру скрытого состояния Equivariant Model. Эта архитектура удовлетворяет требованиям, определенным в прошлой главе, так как применяет нейронную сеть и глубокое обучение для решения задачи мета-обучения, агрегирует табличный набор данных способом, инвариантным к перестановке строк и/или столбцов табличного набора данных, может обрабатывать наборы данных произвольного размера, так как все операции с табличным набором данных производятся как будто бы это множество.

2.4. Нейронная сеть Deep Table Label Channel

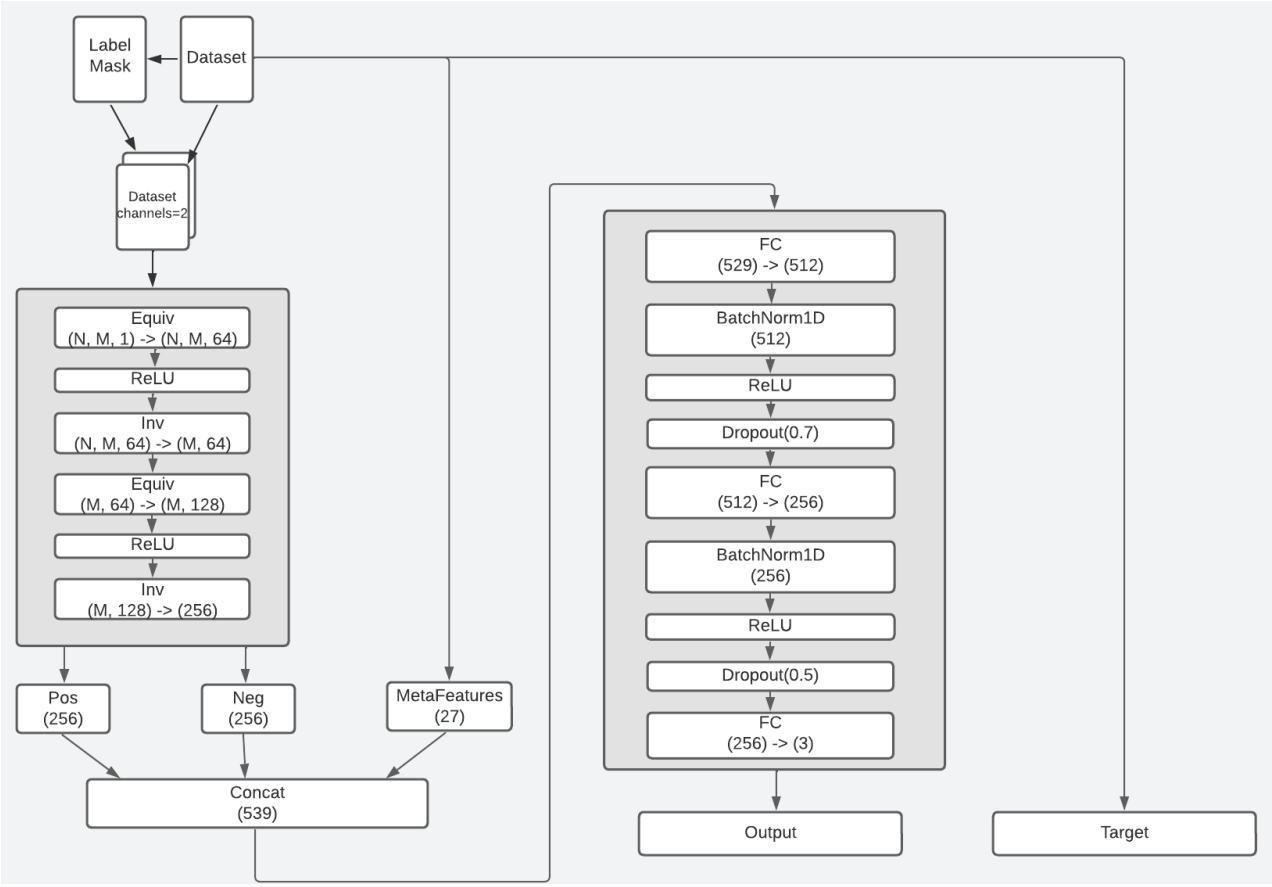


Рисунок 8 – Архитектура Deep Table

В данной архитектуре, информация о классе представляется в виде дополнительного входного канала табличного набора данных, подающегося на

вход нейронной сети. Эта архитектура удовлетворяет требованиям, определенным в прошлой главе, так как применяет нейронную сеть и глубокое обучение для решения задачи мета-обучения, агрегирует табличный набор данных способом, инвариантным к перестановке строк и/или столбцов табличного набора данных, может обрабатывать наборы данных произвольного размера, так как все операции с табличным набором данных производятся как будто бы это множество.

Выводы по главе 2

В этой главе были разобраны предложенные архитектуры для решения задач мета-обучения, описан способ обработки табличного набора данных слоями нейронных сетей, предложенных в статье Deep Sets.

ГЛАВА 3. РЕАЛИЗАЦИЯ АРХИТЕКТУР, РЕЗУЛЬТАТЫ И ДЕТАЛИ ЭКСПЕРИМЕНТОВ, СРАВНЕНИЕ С ДРУГИМИ РЕШЕНИЯМИ

3.1. Конфигурация и детали экспериментов

Сначала будут описаны общие детали всех экспериментов, далее будет описан итеративный процесс экспериментов, некоторые из которых показали положительные результат, некоторые отрицательный, в рамках улучшения метрики на отложенной выборке. Так же будут затронуты детали реализации разработанной архитектуры с использованием библиотеки PyTorch [12].

3.1.1. Общие детали всех экспериментов

Эксперименты проводились с архитектурой, которая решает такую задачу мета-обучения и количество выходных значений сети подстроены под нее: выбираем из следующих алгоритмов

а) KNN($n\_neighbors = 3$)

б) SVM($kernel = "rbf", gamma = 2, C = 1$)

в) Naive Bayes($var\_smoothing = 10^{-9}$)

Рассматриваются табличные наборы данных для задачи бинарной классификации. Табличные наборы данных такие же, как в статье по LM-GAN. Там было отобрано несколько наборов данных для многоклассовой классификации из открытого источника OpenML [10]. В каждом таком наборе данных перебирались пары классов, чтобы сформировать табличный набор данных для бинарной классификации. Они сформированы так, что каждый набор данных состоит из 64 элементов положительного и 64 элементов отрицательного класса. Обучающая выборка состоит из 8000 табличных наборов данных, тестовая выборка состоит из 1911 табличных наборов данных. Это табличные наборы данных такие же, как в статье по LM-GAN. Там было отобрано несколько наборов данных для многоклассовой классификации из открытого источника OpenML [10]. На каждом из табличных наборов запускались 3 указанных алгоритма. Таким образом, целевым значением для каждого набора данных является вектор размера 3, в котором значения элементов это либо ноль либо один, единица соответствует алгоритму, которые является наиболее точным на заданном наборе данных.

Некоторая часть конфигурации экспериментов описывается в программном коде в так называемом классе *Config*, пример в Листинг 5.

3.1.2. Детали итеративного процесса экспериментов

Если какой-либо эксперимент демонстрировал прирост качества на отложенном тестовом наборе данных, то изменение положенное в основу этого эксперимента применялось и для последующих экспериментов.

а) Изначальная конфигурация выглядела следующим образом: выход модели это вектор размера 3, где каждое значения находится в диапазоне от нуля до единицы. Рассматривается простая модель, которая применяет к табличному набору данных только Invariant Model слои, они последовательно применяются для свертки строк, а потом уже для свертки свернутых строк в вектор. Свернув табличный набор данных в вектор, модель применяет двухслойный классификатор к полученному вектору, чтобы получить уверенности модели в каждом алгоритме зафиксированном для нашей задачи мета-обучения. То есть используется только табличный набор данных, без мета-признаков. Обучается при помощи *MSELoss* функцией потерь и оптимизатором Adam [8].

б) Далее по аналогии с LM-GAN в качестве признаков, на основе которых решается задача мульти-классификации, применялись так же и мета-признаки табличного набора данных. Это позволило увеличить точность модели на отложенной выборке, что так же продемонстрировано в таблице в разделе 3.4. .

в) Модель была усложнена, применяются не только Invariant Model слои, но еще и Equivariant Model слои, как это изображено на картинках приведенных в главе 2. Так же, было увеличено количество слоев в самом классификаторе, теперь там 3 линейных слоя с функцией активации ReLU между ними.

г) На этом этапе выход модели был изменён. Теперь модель возвращет не число от нуля до единицы, а так называемый логарифм отношения шансов. Произведен переход в непосредственно функции потерь задачи мультиклассификации. Так как выходы теперь представляют логирифмы отношения шансов, так называемые логиты, необходимо использовать функцию потерь которая умеет их обрабатывать. В PyTorch это *BCEWithLogitsLoss*. Это позволило повысить точность модели, подбирать корректную функция потерь очень важно в задачах глубокого обучения. Так же, BCEWithLogitsLoss обладает большей вычислительной

устойчивостью в виду применения так называемого LogSumExpTrick [1, страница 178].

д) Далее, была применена техника zero-centering. Это способ преобразовать данные так, чтобы их выборочное среднее являлось нулем. В библиотеке scikit-learn [15] это StandardScaler. Такое преобразование применяется как и к каждому признаку табличных наборов, так и к мета-признакам. Параметры преобразования выучиываются по тренировочной выборке, такие же параметры используются и для тестовой выборки. Эта техника позволила улучшить сходимость модели. Количество шагов оптимизации необходимых, для того чтобы достичь фиксированную конкретную точность сократилось.

е) Далее было применено транспонирование табличного набора данных, для того, чтобы соответствовать классическому подходу мета-обучения, в котором сначала агрегируются именно столбцы, а потом уже агрегаты столбцов. Это тоже позволило повысить точность.

ж) Были добавлены слои BatchNorm1d и DropOut, была подобрана архитектура, достигающая наивысшей точности на отложенной тестовой выборке.

и) Далее были произведены эксперименты, направленные на изменения представления информации о классе для каждого объекта табличного набора данных. Ранее на вход модели просто подавался столбец класса объектов. Была опробована модель, которая помимо этого столбца, применяет агрегацию табличного набора данных в вектор отдельно для всех классов. Эта модель далее будет именоваться Deep Table.

к) Эксперимент по агрегации векторов извлеченных из частей табличного набора данных соответствующих разным классам как множества, то есть с применением слоев Deep Sets. Этот эксперимент продемонстрировал ухудшение точности на отложенном тестовом наборе данных.

л) Так же, был опробован подход, в котором для каждого класса используется отдельный блок сжимающий табличный набор данных в вектор признаков. Далее этот подход будет называться Deep Table Multiple Extractors. Результаты этого эксперимента можно увидеть в таблице в разделе 3.4.

м) Последним был экспримент, в котором информация о классе подается как второй канал входного тензора. То есть, берется исходный табличный набор данных, в нему добавляется матрица такой же размерности, где все элементы равны метке класса, которому принадлежит соответствующий объект. Далее этот подход будет называться Deep Table Label Channel.

3.2. Реализация

Для реализации нейронных сетей Deep Table использовалась open-source реализация [11] слоев нейронных сетей описанных в статье Deep Sets.

3.2.1. Нейронная сеть Deep Table

Листинг 1 содержит реализацию нейронной сети Deep Table.
В классе модели Deep Table выделен метод forward_, который применяет прямой проход табличного набора данных по нейронной сети. В нем к входному табличному набору данных применяется операция unsqueeze для того, чтобы преобразовать входной табличный набор данных к тензору размерности $(batch\_size, N, M, 1)$. Далее, применяется операция flatten(0, 1) для того, чтобы преобразовать тензор к размерности $(batch\_size \cdot N, M, 1)$. В таком виде модули Deep Sets корректно обработают табличный набор данных как множество столбцов. Далее в forward при помощи вызовов _forward табличный набор данных агрегируется в вектор признаков, отдельно часть набора данных с положительной меткой класса, отдельно часть набора данных с отрицательной меткой класса. Эти выходы конкатенируется с мета-признаками вычисленными классическими способами. И подаются на вход уже непосредственно классифицирующей части нейронной сети.

3.2.2. Нейронная сеть Deep Table Flattened

Листинг 2 содержит реализацию нейронной сети Deep Table Flattened.
В классе модели Deep Table Flattened аналогично классу модели Deep Table производится преобразование тензора, для получения его необходимой размерности. Однако, в случае обработки табличного набора данных как одного вектора, вторичная агрерация не нужна, поэтому слоев отвечающих за агрегацию тут всего два, это equiv_0 и inv_0.

3.2.3. Нейронная сеть Deep Table Multiple Extractors

Листинг 3 содержит реализацию нейронной сети Deep Table Multiple Extractors.

В методе forward класса модели Deep Table Multiple Extractors применяются аналогичные классу Deep Table преобразования тензора для получаения его необходимой размерности. Отличие от Deep Table заключется в том, что для каждого класса есть своя группа слоев отвечающая за агрегацию табличного набора данных в вектор признаков, так же есть отдельная группа слоев отвечающая за агрегацию в вектор признаков непосредственно всего табличного набора данных целиком. Слоев в этой модели больше, но размерности скрытых состояний уменьшены, чтобы не увеличивать количество параметров модели значительно по сравнению с моделью Deep Table. Для удобства реализации метода forward, метод _forward принимает на вход строку идентифицирующую группу слоев, применемую к табличному набору данных.

3.2.4. Нейронная сеть Deep Table Label Channel

Листинг 4 содержит реализацию нейронной сети Deep Table Label Channel.

Особенность реализации этой нейронной сети заключается в том, что количество входных каналов стало равным двум в слое equiv_0. Табличный набор данных в этом случае подается на вход целиком, без разделения объектов по классам. Поэтому достаточно просто вызвать функцию _forward для того чтобы сагрегировать табличный набор данных в вектор признаков.

3.3. Результаты экспериментов

Среди основных результатов экспериментов, проделанных итеративно, можно выделить три группы: положительно повлиявшие на качество модели на отложенной тестовой выборке, отрицательно повлиявшие и незначительно повлиявшие среди всех предложенных моделей. К первой группе относятся следующие:

а) Мета-признаки конкатенируемые к вектору признаков табличного набора данных, построенных блоком агрегирующим табличный набор данных в вектор.
б) Нормализация входных данных нейронной сети при помощи StandardScaler.

Ко второй группе относится:

а) Агрегация векторов извлеченных из частей табличного набора данных соответствующих разным классам как множества, с применением сло-

ев Deep Sets и с помощью предложенных в статье Deep Sets вариантов агрегаций

К третьей группе относится:

а) Представление информации о классе в виде дополнительного канала, в виде отдельной обработки частей табличного набора данных соответствующих разным классам и в виде применения различных блоков нейронной сети, сжимающих табличный набор данных в вектор признаков к разным частям табличного набора данных соответствующим разным классам

3.4. Сравнение с другими методами мета-обучения

Таблица 1 – Сравнение точности методов мета-обучения на отложенной выборке

Подход	Данные	Модель	Точность
Классический	MF	Decision Tree	0.186 ± 0.000
Классический	MF	KNN	0.286 ± 0.000
Классический	MF	SVC	0.228 ± 0.000
Классический	MF	Logistic Regression	0.173 ± 0.000
Классический	MF	DNN	0.300 ± 0.012
LM-GAN	SD	LM-GAN CNN Discriminator	0.463 ± 0.006
LM-GAN	MF + SD	LM-GAN CNN Discriminator	0.527 ± 0.005
Deep Table	**RD**	**Deep Table**	**0.438 ± 0.007**
Deep Table	**RD**	**Deep Table Flattened**	**0.418 ± 0.010**
Deep Table	**RD**	**Deep Table Multiple Extractors**	**0.449 ± 0.010**
Deep Table	**RD**	**Deep Table Label Channel**	**0.441 ± 0.002**
Deep Table	**MF + RD**	**Deep Table**	**0.578 ± 0.006**
Deep Table	**MF + RD**	**Deep Table Flattened**	**0.576 ± 0.009**
Deep Table	**MF + RD**	**Deep Table Multiple Extractors**	**0.577 ± 0.002**
Deep Table	**MF + RD**	**Deep Table Label Channel**	**0.571 ± 0.008**

Под точностью в этой таблице понимается доля табличных наборов данных из тестовой выборки, на которых верно идентифицирован оптимальный алгоритм. В колонке *Данные* указаны данные, подающиеся на вход соответствующей модели. *MF* обозначает мета-признаки, *SD* обозначает стабилизированный набор данных, а *RD* обозначает набор данных. В таблице приведено число параметров для моделей, являющихся многослойными нейронными сетями, для остальных моделей количество параметров не приведено, потому что оно достаточно мало. Как видно из таблицы, решения, основан-

ные на использовании только мета-признаков в качестве входов модели, не позволяют достичь такой точности, как другие решения. LM-GAN без мета-признаков (6 строчка) справляется лучше, чем Deep Table подходы. Среди Deep Table подходов без мета-признаков лидером по точности является Deep Table Multiple Extractors. Точность у модели Deep Table Multiple Exctractors без мета-признаков больше, чем у Deep Table модели Deep Table Flattened. Модель Deep Table Flattened по сути является просто применением Deep Sets к вектору, который является табличным набором данных представленным в виде одного вектора, это вектор из сконкатенированных столбцов входного набора данных. Это означает, что разработанный подход в случае применения его без мета-признаков позволяет достичь большей точности, чем Deep Sets, хоть и разрыв по точности не такой и большой. Среди моделей, которые на вход применяют не только табличный набор данных, но еще и мета-признаки, самой точной является модель Deep Table. Также стоит отметить, что Deep Table подход с мета-признаками обходит по точности LM-GAN подход в нашей задаче. Но не стоит его недооценивать, так как эта архитектура решает так же и задачу генерации табличных наборов данных с заданными мета-признаками.

Итого, по этой таблице можно сделать вывод, что методы основанные только на мета-признаках не так точны и могут быть улучшены использованием самих табличных наборов данных в качестве входа модели. Разработанные архитектуры Deep Table способны достигать точности обходящей современные методы решения задачи мета-обучения при меньшем числе параметров по сравнению с LM-GAN Discriminator. Однако, точность по сравнению с обычным Deep Sets отличается в положительную сторону не столь сильно.

ЗАКЛЮЧЕНИЕ

Разработан способ обработки табличного набора данных нейронной сетью, обрабатывающий таблицы как множества множеств. Разработана архитектуры глубокого обучения, способная обрабатывать табличные наборы данных. Разработанная архитектура реализована на языке Python с использованием библиотеки глубокого обучения PyTorch. Разработанная архитектура демонстрирует прирост качества относительно современных аналогов в задаче мета-обучения.

ГЛАВА 4. ПРИЛОЖЕНИЕ

Листинг 1 – Нейронная сеть Deep Table на PyTorch

```
import torch
from deepsets import deepsetlayers
import torch.nn.functional as F

class DeepSetModelV5(torch.nn.Module):
    def __init__(self, hidden_size_0=1, hidden_size_1=1,
       predlast_hidden_size=1, meta_size = 27, out_classes=1):
        super().__init__()
        self.hidden_size_0 = hidden_size_0
        self.hidden_size_1 = hidden_size_1
        self.predlast_hidden_size = predlast_hidden_size
        self.meta_size = meta_size
        self.out_classes = out_classes
        self.inv_0 = deepsetlayers.InvLinear(hidden_size_0,
           hidden_size_0)
        self.inv_1 = deepsetlayers.InvLinear(2 * hidden_size_1, 4
           * hidden_size_1)
        self.equiv_0 = deepsetlayers.EquivLinear(1, hidden_size_0)
        self.equiv_1 = deepsetlayers.EquivLinear(hidden_size_0, 2
           * hidden_size_1)
        self.relu = torch.nn.ReLU()
        self.regressor = torch.nn.Sequential(
            torch.nn.Linear(8 * hidden_size_1 + meta_size, 2 *
               predlast_hidden_size),
            torch.nn.BatchNorm1d(2 * predlast_hidden_size),
            torch.nn.ReLU(),
            torch.nn.Dropout(CONFIG.FIRST_DROPOUT_RATE),
            torch.nn.Linear(2 * predlast_hidden_size,
               predlast_hidden_size),
            torch.nn.BatchNorm1d(predlast_hidden_size),
            torch.nn.ReLU(),
            torch.nn.Dropout(CONFIG.SECOND_DROPOUT_RATE),
            torch.nn.Linear(predlast_hidden_size, out_classes),
        )

    def _forward(self, x):
        x = x.unsqueeze(-1)
        # (batch_size, N, M, 1)
        N = x.shape[1]
```

```
        x = x.flatten(0, 1)
        # (batch_size * N, M, 1)
        x = self.equiv_0(x)
        # (batch_size * N, M, hidden_size_0)
        x = self.relu(x)
        x = self.inv_0(x)
        # (batch_size * N, hidden_size_0)
        x = x.reshape(-1, N, self.hidden_size_0)
        # (batch_size, N, hidden_size_0)

        x = self.equiv_1(x)
        # (batch_size, N, hidden_size_1)
        x = self.relu(x)
        x = self.inv_1(x)
        # (batch_size, hidden_size_1)
        return x


    def forward(self, pos, neg, y):

        pos_vec = self._forward(pos)
        neg_vec = self._forward(neg)

        y = y.view(-1, self.meta_size)
        # (batch_size, meta_size)

        x = torch.hstack([pos_vec, neg_vec, y])

        # (batch_size, hidden_size_1 + meta_size)
        x = self.regressor(x)
        # (batch_size, out_classes)
        return x
```

Листинг 2 – Нейронная сеть Deep Table Flattened на PyTorch

```
from deepsets import deepsetlayers
import torch.nn.functional as F

class DeepSetFlattenedModel(torch.nn.Module):
    def __init__(self, hidden_size_0=1, hidden_size_1=1,
       predlast_hidden_size=1, meta_size = 27, out_classes=1):
        super().__init__()
```

```
        self.hidden_size_0 = hidden_size_0
        self.hidden_size_1 = hidden_size_1
        self.predlast_hidden_size = predlast_hidden_size
        self.meta_size = meta_size
        self.out_classes = out_classes
        self.inv_0 = deepsetlayers.InvLinear(hidden_size_0,
           hidden_size_1)
        self.equiv_0 = deepsetlayers.EquivLinear(1, hidden_size_0)
        self.relu = torch.nn.ReLU()
        self.classifier = torch.nn.Sequential(
            torch.nn.Linear(hidden_size_1 + meta_size, 2 *
               predlast_hidden_size),
            torch.nn.BatchNorm1d(2 * predlast_hidden_size),
            torch.nn.ReLU(),
            torch.nn.Dropout(CONFIG.FIRST_DROPOUT_RATE),
            torch.nn.Linear(2 * predlast_hidden_size,
               predlast_hidden_size),
            torch.nn.BatchNorm1d(predlast_hidden_size),
            torch.nn.ReLU(),
            torch.nn.Dropout(CONFIG.SECOND_DROPOUT_RATE),
            torch.nn.Linear(predlast_hidden_size, out_classes),
        )

    def forward(self, X, y):
        # (batch_size, N, M)
        X = X.unsqueeze(-1)
        X = X.flatten(1, 2)
        # (batch_size, N * M, 1)
        X = self.equiv_0(X)
        # (batch_size, N * M, hidden_size_0)
        X = self.inv_0(X)
        # (batch_size, hidden_size_1)

        y = y.view(-1, self.meta_size)
        # (batch_size, meta_size)

        x = torch.hstack([X, y])

        # (batch_size, hidden_size_1 + meta_size)
        x = self.classifier(x)
        # (batch_size, out_classes)
```

```
        return x
```

Листинг 3 – Нейронная сеть Deep Table Multiple Extractors на PyTorch

```
from deepsets import deepsetlayers
import torch.nn.functional as F

class DeepSetModelV6(torch.nn.Module):
    def __init__(self, hidden_size_0=1, hidden_size_1=1,
       predlast_hidden_size=1, meta_size = 27, out_classes=1):
        super().__init__()
        self.hidden_size_0 = hidden_size_0
        self.hidden_size_1 = hidden_size_1
        self.predlast_hidden_size = predlast_hidden_size
        self.meta_size = meta_size
        self.out_classes = out_classes

        # pos
        self.inv_0_pos = deepsetlayers.InvLinear(hidden_size_0,
           hidden_size_0)
        self.inv_1_pos = deepsetlayers.InvLinear(hidden_size_1, 3
           * hidden_size_1 // 2)
        self.equiv_0_pos = deepsetlayers.EquivLinear(1,
           hidden_size_0)
        self.equiv_1_pos = deepsetlayers.EquivLinear(hidden_size_0
           , hidden_size_1)

        # neg
        self.inv_0_neg = deepsetlayers.InvLinear(hidden_size_0,
           hidden_size_0)
        self.inv_1_neg = deepsetlayers.InvLinear(hidden_size_1, 3
           * hidden_size_1 // 2)
        self.equiv_0_neg = deepsetlayers.EquivLinear(1,
           hidden_size_0)
        self.equiv_1_neg = deepsetlayers.EquivLinear(hidden_size_0
           , hidden_size_1)

        # all
        self.inv_0_all = deepsetlayers.InvLinear(hidden_size_0,
           hidden_size_0)
        self.inv_1_all = deepsetlayers.InvLinear(hidden_size_1, 3
           * hidden_size_1 // 2)
```

```
        self.equiv_0_all = deepsetlayers.EquivLinear(1,
           hidden_size_0)
        self.equiv_1_all = deepsetlayers.EquivLinear(hidden_size_0
           , hidden_size_1)

        self.relu = torch.nn.ReLU()
        self.classifier = torch.nn.Sequential(
            torch.nn.Linear(3 * (3 * hidden_size_1 // 2) +
               meta_size, 2 * predlast_hidden_size),
            torch.nn.BatchNorm1d(2 * predlast_hidden_size),
            torch.nn.ReLU(),
            torch.nn.Dropout(CONFIG.FIRST_DROPOUT_RATE),
            torch.nn.Linear(2 * predlast_hidden_size,
               predlast_hidden_size),
            torch.nn.BatchNorm1d(predlast_hidden_size),
            torch.nn.ReLU(),
            torch.nn.Dropout(CONFIG.SECOND_DROPOUT_RATE),
            torch.nn.Linear(predlast_hidden_size, out_classes),
        )

    def _forward(self, x, mode='all'):
        if mode == 'pos':
            equiv_0 = self.equiv_0_pos
            equiv_1 = self.equiv_1_pos
            inv_0 = self.inv_0_pos
            inv_1 = self.inv_1_pos
        elif mode == 'neg':
            equiv_0 = self.equiv_0_neg
            equiv_1 = self.equiv_1_neg
            inv_0 = self.inv_0_neg
            inv_1 = self.inv_1_neg
        elif mode == 'all':
            equiv_0 = self.equiv_0_all
            equiv_1 = self.equiv_1_all
            inv_0 = self.inv_0_all
            inv_1 = self.inv_1_all

        x = x.unsqueeze(-1)
        # (batch_size, N, M, 1)
        N = x.shape[1]
        x = x.flatten(0, 1)
```

```
        # (batch_size * N, M, 1)
        x = equiv_0(x)
        # (batch_size * N, M, hidden_size_0)
        x = self.relu(x)
        x = inv_0(x)
        # (batch_size * N, hidden_size_0)
        x = x.reshape(-1, N, self.hidden_size_0)
        # (batch_size, N, hidden_size_0)

        x = equiv_1(x)
        # (batch_size, N, hidden_size_1)
        x = self.relu(x)
        x = inv_1(x)
        # (batch_size, hidden_size_1)
        return x


    def forward(self, pos, neg, y):

        pos_vec = self._forward(pos, 'pos')
        neg_vec = self._forward(neg, 'neg')

        all = torch.cat([pos, neg], dim=2)
        all_vec = self._forward(all, 'all')

        y = y.view(-1, self.meta_size)

        # (batch_size, meta_size)

        x = torch.hstack([pos_vec, neg_vec, all_vec, y])

        # (batch_size, hidden_size_1 + meta_size)
        x = self.classifier(x)
        # (batch_size, out_classes)
        return x
```

Листинг 4 – Нейронная сеть Deep Table Label Channel на PyTorch

```
    import torch
from deepsets import deepsetlayers
import torch.nn.functional as F
```

```
class DeepSetModelV7(torch.nn.Module):
    def __init__(self, hidden_size_0=1, hidden_size_1=1,
       predlast_hidden_size=1, meta_size = 27, out_classes=1,
       in_channels=2):
        super().__init__()
        self.hidden_size_0 = hidden_size_0
        self.hidden_size_1 = hidden_size_1
        self.predlast_hidden_size = predlast_hidden_size
        self.meta_size = meta_size
        self.out_classes = out_classes
        self.in_channels = in_channels
        self.inv_0 = deepsetlayers.InvLinear(hidden_size_0,
           hidden_size_0)
        self.inv_1 = deepsetlayers.InvLinear(2 * hidden_size_1, 4
           * hidden_size_1)
        self.equiv_0 = deepsetlayers.EquivLinear(self.in_channels,
            hidden_size_0)
        self.equiv_1 = deepsetlayers.EquivLinear(hidden_size_0, 2
           * hidden_size_1)
        self.relu = torch.nn.ReLU()
        self.classifier = torch.nn.Sequential(
            torch.nn.Linear(4 * hidden_size_1 + meta_size, 2 *
               predlast_hidden_size),
            torch.nn.BatchNorm1d(2 * predlast_hidden_size),
            torch.nn.ReLU(),
            torch.nn.Dropout(CONFIG.FIRST_DROPOUT_RATE),
            torch.nn.Linear(2 * predlast_hidden_size,
               predlast_hidden_size),
            torch.nn.BatchNorm1d(predlast_hidden_size),
            torch.nn.ReLU(),
            torch.nn.Dropout(CONFIG.SECOND_DROPOUT_RATE),
            torch.nn.Linear(predlast_hidden_size, out_classes),
        )

    def _forward(self, x):
        # (batch_size, N, M, 1)
        N = x.shape[1]
        x = x.flatten(0, 1)
        # (batch_size * N, M, 1)
        x = self.equiv_0(x)
        # (batch_size * N, M, hidden_size_0)
```

```
        x = self.relu(x)
        x = self.inv_0(x)
        # (batch_size * N, hidden_size_0)
        x = x.reshape(-1, N, self.hidden_size_0)
        # (batch_size, N, hidden_size_0)

        x = self.equiv_1(x)
        # (batch_size, N, hidden_size_1)
        x = self.relu(x)
        x = self.inv_1(x)
        # (batch_size, hidden_size_1)
        return x


    def forward(self, X, y):
        X = self._forward(X)

        y = y.view(-1, self.meta_size)
        # (batch_size, meta_size)

        x = torch.hstack([X, y])
        # (batch_size, hidden_size_1 + meta_size)

        x = self.classifier(x)
        # (batch_size, out_classes)
        return x
```

Листинг 5 – Конфигурационный класс

```
class CONFIG:
    # DATA
    features = 16
    instances = 64
    classes = 2

    # TRAIN
    num_epochs = 50
    train_batch_size = 1024
    learning_rate = 0.002
    criterion = 'torch.nn.BCEWithLogitsLoss'
    FIRST_DROPOUT_RATE = 0.7
    SECOND_DROPOUT_RATE = 0.5
```

```
test_batch_size = 1024

# MODEL
in_channels=2
hidden_size_0=64
hidden_size_1=128
predlast_hidden_size=256
meta_size=27
out_classes=3
standardscaler=True
meta_standardscaler=True
transpose=True

device_mps_or_cpu = ("mps" if torch.backends.mps.is_available
    () else "cpu")
device = torch.device("cuda:0" if torch.cuda.is_available()
    else device_mps_or_cpu)

# OTHER
seed = 42
num_workers = 8
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Ive J.* Natural Language Processing: A Machine Learning Perspective by Yue Zhang and Zhiyang Teng // Computational Linguistics. — 2022. — Апр. — Т. 48, № 1. — С. 233–235. — ISSN 0891-2017. — DOI: 10.1162/coli_r_00423. — eprint: https://direct.mit.edu/coli/article-pdf/48/1/233/2006603/coli_r_00423.pdf. — URL: https://doi.org/10.1162/coli%5C_r%5C_00423.

2 Deep Sets / M. Zaheer [et al.] // Advances in Neural Information Processing Systems. Vol. 30 / ed. by I. Guyon [et al.]. — Curran Associates, Inc., 2017. — URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf.

3 Dropout: A Simple Way to Prevent Neural Networks from Overfitting / N. Srivastava [et al.] // J. Mach. Learn. Res. — 2014. — Jan. — Vol. 15, no. 1. — P. 1929–1958. — ISSN 1532-4435.

4 *Fukushima K.* Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements // IEEE Transactions on Systems Science and Cybernetics. — 1969. — Vol. 5, no. 4. — P. 322–333. — DOI: 10.1109/TSSC.1969.300225.

5 Generating Datasets for Classification Task and Predicting Best Classifiers with Conditional Generative Adversarial Networks / I. Kachalsky [et al.] //. — Istanbul, Turkey : Association for Computing Machinery, 2020. — P. 97–101. — (ICAAI '19). — ISBN 9781450372534. — DOI: 10.1145/3369114.3369153. — URL: https://doi.org/10.1145/3369114.3369153.

6 Generative Adversarial Nets / I. Goodfellow [et al.] // Advances in Neural Information Processing Systems. Vol. 27 / ed. by Z. Ghahramani [et al.]. — Curran Associates, Inc., 2014. — URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

7 *Ioffe S.*, *Szegedy C.* Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37. — Lille, France : JMLR.org, 2015. — P. 448–456. — (ICML'15).

8 *Kingma D. P.*, *Ba J.* Adam: A Method for Stochastic Optimization. — 2017. — arXiv: 1412.6980 [cs.LG].

9 *Mirza M.*, *Osindero S.* Conditional Generative Adversarial Nets. — 2014. — arXiv: 1411.1784 [cs.LG].

10 OpenML: networked science in machine learning / J. Vanschoren [et al.] // SIGKDD Explorations. — 2013. — Vol. 15, no. 2. — P. 49–60. — DOI: 10.1145/2641190.2641198. — URL: http://doi.acm.org/10.1145/2641190.264119.

11 *Pernes D.* deepsets-digitsum. — 2019. — https://github.com/dpernes/deepsets-digitsum.

12 PyTorch: An Imperative Style, High-Performance Deep Learning Library / A. Paszke [et al.] // Advances in Neural Information Processing Systems 32 / ed. by H. Wallach [et al.]. — Curran Associates, Inc., 2019. — P. 8024–8035. — URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

13 *Radford A.*, *Metz L.*, *Chintala S.* Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. — 2016. — arXiv: 1511.06434 [cs.LG].

14 *Rice J. R.* The Algorithm Selection Problem //. Vol. 15 / ed. by M. Rubinoff, M. C. Yovits. — Elsevier, 1976. — P. 65–118. — (Advances in Computers). — DOI: https://doi.org/10.1016/S0065-2458(08)60520-3. — URL: https://www.sciencedirect.com/science/article/pii/S0065245808605203.

15 Scikit-learn: Machine Learning in Python / F. Pedregosa [et al.] // Journal of Machine Learning Research. — 2011. — Vol. 12. — P. 2825–2830.

16 *Vanschoren J.* Meta-Learning: A Survey // CoRR. — 2018. — Vol. abs/1810.03548. — arXiv: 1810.03548. — URL: http://arxiv.org/abs/1810.03548.