МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УО «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра математических методов в экономике
Специальность «Экономическая кибернетика»

Допущена к защите
Заведующий кафедрой
д-р экон. наук, доц.
_____ Г.О. Читая
31.05.2022

ДИПЛОМНАЯ РАБОТА

на тему: **Разработка и использование моделей классификации в кредитном скоринге(на примере ОАО «Белинвестбанк»)**

Студент
ФЦЭ, 4-й курс, ДКК-1 _____ Ф.А. Кобак

Руководитель
доктор экон. наук,
профессор _____ Э.М. Аксень

Нормоконтролер _____ И.В. Денисейко

МИНСК 2022

РЕФЕРАТ

Дипломная работа: 108 с., 15 табл., 33 рис., 7 лист., 21 ист., 11 прил.

КЛАССИФИКАЦИЯ, КРЕДИТНЫЙ СКОРИНГ, ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, АЛГОРИМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ, ВАЛИДАЦИЯ МОДЕЛИ, ОТБОР ПОКАЗАТЕЛЕЙ, AUC, KS, ТЕСТ КОЛМОГОРОВА-СМИРНОВА

Объект исследования – правило классификации кредитополучателей в ОАО «Белинвестбанк».

Предмет исследования – модели искусственных нейронных сетей в приложении к задаче классификации.

Цель работы: построение нелинейного классификатора для целей кредитного скоринга.

Методы исследования: компьютерный анализ данных, машинное обучение.

Исследования и разработки: при отборе показателей и преобразованиях данных для модели задействован особый метод, использующий ROC анализ, подготовлен ряд инструментов для автоматизации процесса подготовки данных.

Элементы научной новизны: новый метод оценки индивидуальной классифицирующей способности каждого показателя.

Область возможного практического применения: задачи требующие проведения классификации клиентов по собранным данным.

Технико-экономическая и социальная значимость: построение моделей, подобных рассмотренным в работе, позволит решить задачи в которых производительности классических методов классификации недостаточно.

Автор работы подтверждает, что приведенный в ней расчетно-аналитический материал правильно и объективно отражает состояние исследуемого процесса, а все заимствованные из литературных и других источников теоритические, методологические и методические положения и концепции сопровождаются ссылками на их авторов.

ABSTRACT

Term work: 108 p., 15 tables, 33 fig., 7 listings, 21 res., 11 supp.

CLASSIFICATION, CREDIT SCORING, ARTIFICIAL NEURAL NETWORKS, BACKWARD ERROR PROPAGATION ALGORITHM, MODEL VALIDATION, FEATURES SELECTION, AUC, KS, KOLMOGOROV-SMIRNOV TEST

The object of study – borrowers classification rule of JSC «Belinvestbank».

The subject of research – models of artificial neural networks attached to classification task.

Objective: fitting of nonlinear classifier for purposes of credit scoring.

Methods: computer data analysis, machine learning.

Research and development: when selecting features and transforming data for the model, a special method is used, the method uses ROC analysis; a number of tools have been prepared to automate the data preparation process.

Elements of scientific novelty: developed evaluation method of individual classifying ability of each feature.

The area of possible practical applications: tasks required classification of clients based on data.

Technical, economic and social significance: fitting models, closed to researched in work, gives an opportunity to solve problems which requires more performance than classic classification algorithms can give.

The author of the work confirms that the calculation and analytical material presented in it correctly and objectively reflects the state of the process under study, and all theoretical and methodological provisions and concepts borrowed from literary and other sources are accompanied by references to their authors.

.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5
1 Теория моделей искусственной нейронной сети в задаче классификации 7
1.1 Постановка задачи 7
1.2 Модель логистической регрессии – линейный классификатор 9
1.3 Модель искусственной нейронной сети в классификации 16
1.4 Целевые функции и алгоритм обратного распространения ошибки 22
2 Анализ и подготовка данных 28
2.1 Начальный анализ наблюденных данных 28
2.2 Оценка классифицирующей способности и отбор предикторов 34
3 Построение и валидация модели 48
3.1 Программное описание модели и алгоритма обучения 48
3.1.1 Основные элементы модели в pytorch 48
3.1.2 Реализация алгоритма обучения 50
3.2 Обучение и валидация модели 54
3.1.1 Подбор параметров обучения 54
3.2.2 Более тонкая подгонка и валидация финальной модели 58
ЗАКЛЮЧЕНИЕ 65
ПРИЛОЖЕНИЕ А Визуализация сигмоиды двух переменных 70
ПРИЛОЖЕНИЕ Б Архитектуры с сигмоидой на выходном слое 71
ПРИЛОЖЕНИЕ В Обобщенные программные функции процессинга данных . 72
ПРИЛОЖЕНИЕ Г Данные на разных этапах обработки 75
ПРИЛОЖЕНИЕ Д Графическая интерпретация TP, FP 88
ПРИЛОЖЕНИЕ Е ROC анализ для номинативной переменной 90
ПРИЛОЖЕНИЕ Ж ROC анализ данных для модели 92
ПРИЛОЖЕНИЕ И Алгоритм оценки параметров 101
ПРИЛОЖЕНИЕ К Исследование обучения по эпохам 106
ПРИЛОЖЕНИЕ Л Характеристики модели при обучении 107

ВВЕДЕНИЕ

Задача классификации – одна из центральный задач решаемых с помощью алгоритмов машинного обучения. Именно к задаче классификации сводиться целый ряд задач из самых разных прикладных областей: в медицине – определение заболевания по набору симптомов, в кибербезопастности – определение потенциально мошеннических сообщений, в компьютерном зрении – по набору пикселей отличать один объект от другого и, наконец, в прикладной экономической науке – классификация клиентов. В данной работе рассмотрена еще более узкая область – кредитный скоринг. Это направление которое ставит своей задачей формирование системы оценки клиентов относительно их способности выполнять обязательства перед банком.

В самом деле, в любой сфере деятельности при работе с клиентами было бы очень полезно знать наперед, как поведет себя тот или иной клиент. Свое решение предлагают статистические методы. Если обобщить и упростить, предлагается накопить большой объем данных описывающий каждый частный случай, а затем отследить как в каждом конкретном случае проявлялось исследуемое явление, а, при принятии решений о новых случаях, учитывать выводы полученные на тех данных. Применительно к кредитному скорингу это обычно реализовано так: при подаче заявки на получение кредита потенциальный кредитополучатель должен указать некоторые данные, по этим данный принимается решение о выдаче или удержании кредита, эти же данные учитываются для того, чтобы сделать новые выводы.

Конечно, можно и без накопленной информации рассматривать каждую заявку в ручную и принимать решение о выдаче или удержании кредита. Но касательно кредитов физическим лицам и, даже, малому бизнесу, в крупный банк может поступать до ста тысяч заявок за год. Не возникает сомнений, что для обработки такого объема информации потребуются просто недопустимые людские затраты. Кроме того, такая ситуация ведет к накоплению большого объема информации, что является отличной почвой для реализации статистических методов. Этим объясняется актуальность выбранной темы.

Студент-дипломник проходил преддипломную практику в ОАО «Белинвестбанк». Там, как раз, проводили плановые обновления и валидации моделей кредитного скоринга клиентов физических лиц. Потому объектом исследования станет правило классификации кредитополучателей в ОАО «Белинвестбанк».

В процессе прохождения практики были предприняты попытки построения линейного классификатора идентификационной формы логистической регрессии. Качества построенного классификатора едва хватало для того, чтобы допустить модель к работе. Было решено усложнять форму модели для преодоления линейности и перейти к классу моделей искусственных нейронных сетей. Таким образом предметом исследования станут модели искусственных нейронные сети в приложении к задаче классификации.

От сюда сформируем задачи дипломной работы. В первой главе займемся теорией моделей искусственных нейронных сетей. Перейдем от самых простых идей к более сложным по ходу усложнения поступающих задач. Раскроем чем плоха привычная линейная регрессия и как ее проблемы решаются в модели логистической регрессии. Объясним, почему модель логистической регрессии является линейным классификатором несмотря на очевидную нелинейность идентификационной формы. Введем терминологию и, в развитии идей логистической регрессии, перейдем к искусственным нейронным сетям. Подробно рассмотрим принципы работы искусственных нейронных сетей и разберемся в особенностях построения данного вида моделей.

Во второй главе познакомился с имеющейся задачей и поработаем с массивом данных полученным для построения модели. Затронем вопросы преобразования данных, очистки от выбросов и невозможных значений, борьбу с пропущенными значениями. При построении моделей лучше избавляться от неинформативных данных путем понижения размерности. Это может заметно упростить процесс построения модели и снизить вычислительную нагрузку при обучении модели. При отборе наиболее информативных критериев был использован особый метод, потому одной из задач ставим его обоснование и понятное разъяснение.

Для проведения вычислений в работе использован язык программирования python3. Относительно простой синтаксис, развитое сообщество генерирующее бесплатные инструменты, особенно применительно к обработке данных и моделированию, и доступная документация делают его идеальным кандидатом для использования в подобных задачах. Весь код приведен в работе не будет. Безусловный плюс в пользу использования языков программирования для моделирования, по сравнению с инструментами не предполагающими программирования, состоит в возможности создания абстрактных инструментов которые могут быть множество раз повторно использованы. Такие инструменты были созданы и при реализации практической части этой работы мы приведем только их и результаты их выполнения.

Располагая теоритическими выводами первой главы и набором данных, подготовленным методами, описанными во второй главе, попробуем построить соответствующую модель. Результаты найдут отражение в третьей главе. Обязательно применим и опишем опыт представленный в специальной литературе, сконцентрируемся на технических моментах связанных с pytorch – одной из самых популярных библиотек для создания моделей искусственных нейронных сетей.

Уже упоминали ОАО «Белинвестбанк» (далее Банк), это организация благодаря которой и для которой были рассмотрены данные методы. Банк является одним из ведущих банков Республики Беларусь и специализируется на кредитовании конечного потребителя, потому для него такая разработка может быть особенно полезна.

1 Теория моделей искусственной нейронной сети в задаче классификации

1.1 Постановка задачи

Рассматриваемые в этой работе методы, на ряду с некоторыми другими, предназначены для решения задач классификации. Задачей классификации называется задача в которой требуется определить способ отнесения некоторых объектов к некоторым группам (классам).

Такие задачи разбиваются на два вида – те для которых известно число классов K и те для которых число классов заранее неизвестно. В этой работе рассматриваются только задачи первого вида.

Особенностью описного типа задач является наличие предсказываемого фактора Y. То есть, для каждого $i-$го объекта из изучаемой совокупности имеется признак y_i который может принимать значения $O_1, O_2, \dots, O_K$:

$$y_i = \begin{cases} O_1, \\ O_2, \\ \dots, \\ O_K. \end{cases} \quad (1.1)$$

Предположим, имеются наблюдения за некоторым явлением или процессом которые можно представить подобно таблице 1.1.

Таблица 1.1 – Исходная система данных

i	Y	X_1	X_2	…	X_m
1	y_1	x_{11}	x_{12}	…	x_{1m}
2	y_2	x_{21}	x_{22}	…	x_{2m}
…	…	…	…	…	…
n	y_n	x_{n1}	x_{n2}	…	x_{nm}

Примечание – Источник: собственная разработка.

Где по строкам расположились наблюдения, а по столбцам некоторые переменные для этих наблюдений. На основе этих данных формируется правило, которое позволяет, получив произвольный набор значении переменных $\hat{X} = \{\hat{X}_1, \hat{X}_2, \dots, \hat{X}_m\}$, наиболее точно, в некотором смысле, предсказать значения $\hat{Y}$. В литературе это правило нередко обозначается так:

$$a: X \to Y.$$

Это очень распространенная задача для прикладной статистики и машинного обучения, встречающаяся во многих сферах жизни: задача определения диагноза по симптомам и анализам; разбиение электронной почты на действительную и спам; задачи распознавания рукописного текста и множество других.

Итак, была получена задача, в которой имеется набор наблюдений с откликом Y, аналогичная задача решается классическим регрессионным анализом, разница лишь в типе отклика – для обычной регрессионной модели он численная переменная, в нашем же случае номинативная (категориальная).

Номинативной называют переменную, для которой не определены ни порядок, ни шкала [1 с. 316]. Под отсудившем шкалы понимается, что мы не можем как-либо определить расстояние между двумя различными значениями переменной, например, если предсказывается факт возврата или невозврата кредитополучателем задолженности, нет никакой возможности показать на сколько возврат «выше» или «ниже» невозврата. Понятие порядка переменной будет более подробно раскрыто в третьем подразделе, когда речь пойдет о упорядоченной модели логистической регрессии.

Описанная разница в типе отклика и порождает непригодность использования классической регрессионной модели.

Рассмотрим задачу бинарной классификации – отклик может принимать лишь два значения ($K=2$), для простоты обозначим события (1.1) каким-либо числами, обычно используется:

$$Y = \begin{cases} 0; \\ 1. \end{cases}$$

После оценивания будет получено регрессионное уравнение следующего вида:

$$\hat{y}_i = a + \sum_{j=1}^{m} x_{ij} b_j + \varepsilon_i,$$

где b_i – оценка коэффициента регрессии;

a – оценка свободного члена;

ε_i – случайная ошибка.

Отдельного обсуждения достойна переменная $\hat{y}_i$, она представляет собой оценку вероятности того, что исследуемый объект принадлежит к классу советующему числу 1. Тут возникает первая проблема такого подхода – нет никаких оснований чтобы $\hat{y}_i \in [0; 1]$, что в корне неверно для понятия вероятности. Наглядная иллюстрация этой проблемы представлена на рисунке 1.1 слева – две выборки распределены вдоль некоторой переменной x, притом для тех для которых $y = 1$, значения x заметно выше, построив регрессионную прямую наглядно убеждаемся в том, что ряд наблюдений получают оценки вероятности за границами нуля и единицы.

На рисунке 1.1 справа наглядно представлена еще одна проблема описанного подхода – к той же выборке, которая обсуждалась выше, был добавлен ряд наблюдений с уровнем исследуемого фактора соответствующим $y = 1$, с заметным смещением в большую сторону по фактору x. Регрессионная прямая в таком случае сместилась и заметно хуже предсказывает вероятности

для старых наблюдений – вся группа советующая $y = 0$, получила оценки вероятностей того, что они принадлежат группе $y = 1$, в районе 0,4, что вообще говоря достаточно плохо для такого простого примера. Пример того как с аналогичной задачей справиться логистическая регрессия будет представлен в следующем подразделе.

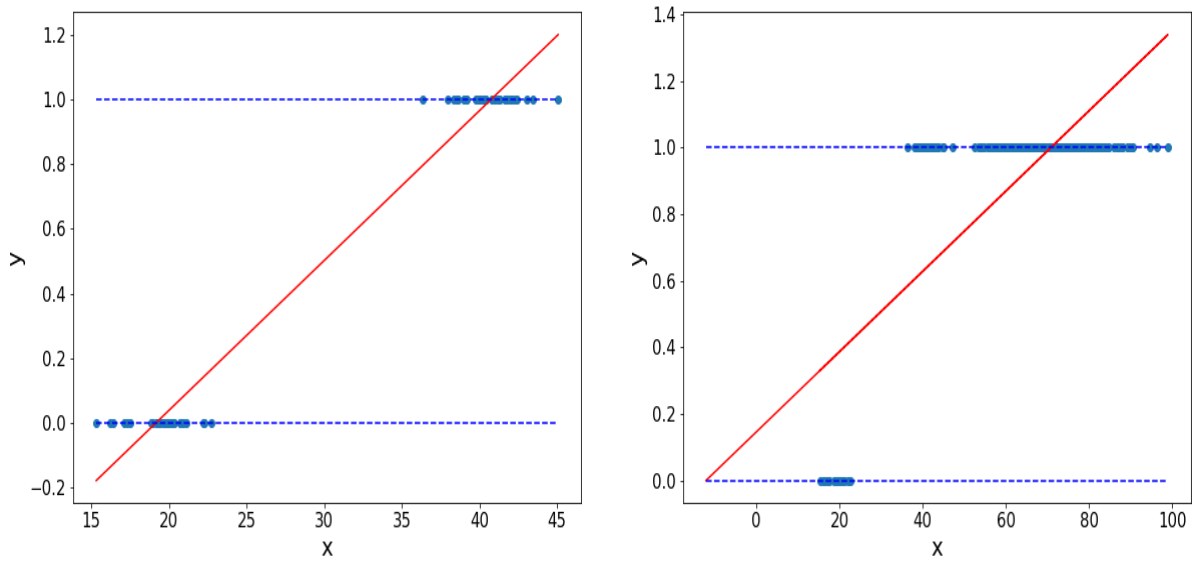

Рисунок 1.1 – Оценки вероятностей через линейную регрессионную модель

Примечание – Источник: собственная разработка.

Более того, такая модель плохо обобщается для случая, когда требуется решение для задачи не бинарной классификации. [2 с. 145]

Наличие всех описанных проблем при использовании регрессионного анализа для моделирования процесса с номинативным откликом и вызвало потребность в разработке специальных методов классификации, одним из которых является логистическая регрессия, рассматриваемая в следующем подразделе.

1.2 Модель логистической регрессии – линейный классификатор

Отталкиваясь от того, что было сказано в прошлом разделе, проведем ряд рассуждений, которые приведут к модели логистической регрессии, для задачи бинарной классификации.

Обозначим вероятность того, что исследуемый признак примет значение $y = 1$ как $P(y = 1)$. Одной из ключевых проблем в данном случае является то что $P(y = 1) \in [0,1]$, в то время как отклик в модели регрессионного анализа принимает значение $\hat{y}_i \in (-\infty, +\infty)$.

Введем понятие шанса события, шансом появления некоторого события называется отношение вероятности появления этого события к вероятности появления любого другого совместного события. В нашем случае справедливо:

$$odds(y = 1) = \frac{P(y = 1)}{1 - P(y = 1)}.$$

Несложно показать, что $odds(y = 1) \in [0, +\infty)$. Пользуясь свойствами логарифма получим, что $\ln[odds(y = 1)] \in (-\infty, +\infty)$, а такая переменная уже хороший кандидат, для того чтобы быть описанной методом линейной регрессии. Таким образом идея логистической регрессии предлагает предсказывать не вероятность того что $y = 1$, а логарифм отношения шансов этого события. Логарифм отношения вероятностей в дальнейшем будет называть логит функцией.

Для краткости последующих записей обозначим:

$$\ln[odds(y = 1)] = y_{odds}.$$

Получим аналитическую запись рассматриваемой модели. Запишем теоретическую линейную модель предсказывающую логарифм отношения шансов:

$$y_{odds} = \alpha + \sum_{j=1}^{m} x_j \beta_j.$$

Имея значение логарифма отношения шансов легко получить искомую вероятность. Сначала проведем потенцирование рассматриваемого выражения:

$$e^{y_{odds}} = e^{\alpha + \sum_{j=1}^{m} x_j \beta_j}.$$

Используя свойства натурального логарифма, получим:

$$\frac{P(y = 1)}{1 - P(y = 1)} = e^{\alpha + \sum_{j=1}^{m} x_j \beta_j}.$$

Решив это уравнение относительно $P(y = 1)$, получим общую запись модели логистической регрессии:

$$P(y = 1) = \frac{e^{\alpha + \sum_{j=1}^{m} x_j \beta_j}}{1 + e^{\alpha + \sum_{j=1}^{m} x_j \beta_j}}. \quad (1.2)$$

Выражение (1.2) и есть модель логистической регрессии для случая бинарной классификации[3 с. 32]. Функция, лежащая в основании этой модели, соответствует функции логистического распределения и имеет два ключевых полезных для рассматриваемой задачи свойства: во-первых, она принимает значения в диапазоне от нуля до единицы, во-вторых она принадлежит к классу

сигмовидных функций, то есть это гладкая, возрастающая функция, имеющая на графике форму буквы «S». В литературе распространено название – сигма функция или сигмоида и в общем она записывается так:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Вернемся к примеру, из предыдущего раздела и посмотрим, как логистическая регрессия справиться с поставленной задачей. На рисунке 1.2 синими точками по-прежнему обозначены сгенерированные наблюдения, красной линией теперь обозначаются оценки вероятностей для различных значений предиктора x, полученные по модели подобной (1.2). Как видно обе обозначенные проблемы решены, предсказания лежат строго в пределах от нуля до единицы, и смещенная выборка почти не влияет на качество модели.

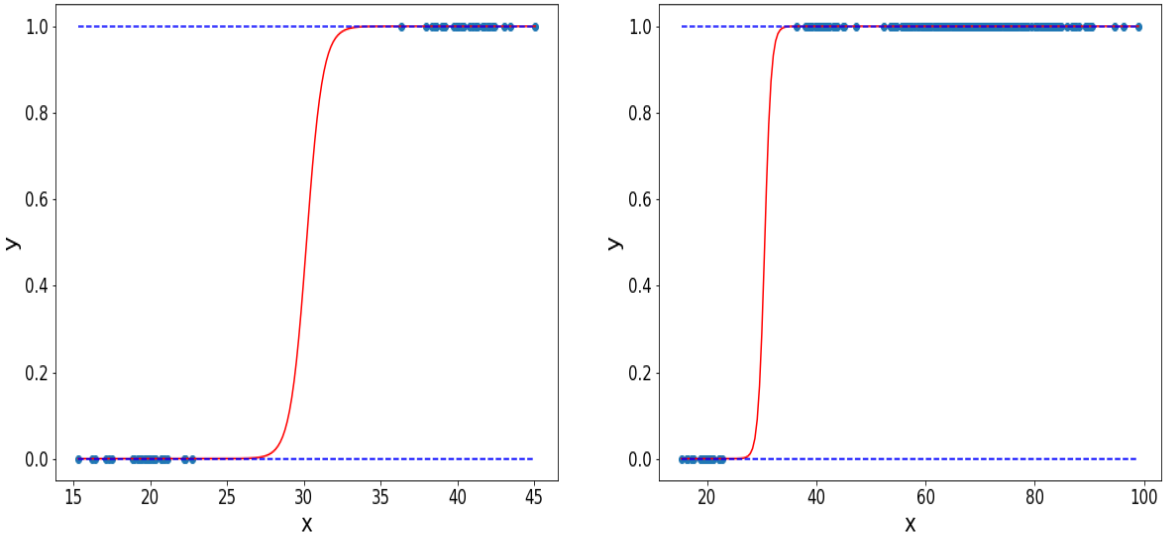

Рисунок 1.2 – Оценки вероятностей через логистическую регрессионную модель

Примечание – Источник: собственная разработка.

Рассмотрим более обобщенный вариант – когда предсказываемая переменная не бинарна, а имеет более двух уровней. Такую модель принято называть мультиномиальной логистической регрессией.

В данном случае предсказываемый фактор будет закодирован, так:

$$Y = \begin{cases} 0; \\ 1; \\ \dots \\ K - 1. \end{cases}$$

В данном случае нам понадобиться K-1 логит функции. Кроме того, надо выбрать базовый уровень выходной переменной, пусть, не нарушая общности, это будет Y=0. Для такого случая логиты можно записать так:

$$g_k(x) = \ln\left(\frac{P(y=k)}{P(y=0)}\right), k = \overline{1, K-1}. \quad (1.3)$$

Притом, каждый из них будет описываться предикторами в соответствии со следующим правилом:

$$g_k(x) = \alpha_k + \sum_{j=1}^{m} x_j \beta_{kj}, i = \overline{0, K-1}.$$

Чтобы получить в данном случае запись, подобную выражению (1.2) для биномиальной регрессии, потребуется поработать с системой (1.3). Для начала проведем потенцирование каждого уравнения системы:

$$e^{g_k(x)} = \frac{P(y=k)}{P(y=0)}, k = \overline{1, K-1}. \quad (1.4)$$

Теперь из одного уравнения выразим $P(y=0)$, пусть, не нарушая общности, это будет первое уравнение:

$$P(y=0) = \frac{P(y=1)}{e^{g_1(x)}}.$$

Используя определение полной вероятности, получим:

$$P(y=0) = \frac{1 - P(y=0) - \sum_{k=2}^{K-1} P(y=k)}{e^{g_1(x)}}. \quad (1.5)$$

Из оставшихся уравнений системы (1.3) получим:

$$P(y=k) = P(y=0)e^{g_k(x)}, k = \overline{2, K-1}. \quad (1.6)$$

Перенеся левую часть выражения (1.5) приведя и общему знаменателю, положив его неравным и вынеся $-P(y=0)$ за скобки, получим:

$$0 = 1 - P(y=0)\left(1 + \sum_{k=1}^{K-1} e^{g_k(x)}\right).$$

Выражая от сюда $P(y=0)$ получим:

$$P(y=0)=\frac{1}{1+\sum_{k=1}^{K-1}e^{g_k(x)}}. \quad (1.7)$$

Вообще говоря, выражение (1.6) будет справедливо и при k=1, потому подставляя туда (1.7), легко получим вероятности появления других уровней исследуемого признака:

$$P(y=k)=\frac{e^{g_k(x)}}{1+\sum_{j=1}^{K-1}e^{g_j(x)}}, k=\overline{1,K-1}. \quad (1.8)$$

Выражения (1.7) и (1.8) – самая общая запись логистической регрессии. Для удобства последующих записей, предполагая что $g_0(x)=0$, запишем одной формулой:

$$P(y=k)=\frac{e^{g_k(x)}}{1+\sum_{j=1}^{K-1}e^{g_j(x)}}, k=\overline{0,K-1}. \quad (1.9)$$

Сразу повторюсь, для заострения внимания – вероятность проявления каждого возможного уровня отклика имеет свою функцию от параметров и входных данных X. Заметим, что оценки коэффициентов α_k и $\beta_{kj}, k=\overline{0,K-1}, j=\overline{1,m}$, получают методом максимального правдоподобия.

Теперь поговорим о том, почему методы машинного обучения в задачах классификации не остановились на логистической регрессии. В литературе можно встретить утверждение: «модель логистической регрессии – линейный классификатор». На первый взгляд может показаться, что это утверждение неверно, ведь выражение (1.2) никак не назвать линейным входного набора данных и сигмоида на рисунке 1.2 в целом кривая. Дело тут кроется в принципе принятия решения о отнесении наблюдения к тому или иному классу. Данное явление лучше всего рассматривать на примере двумерной задачи.

На рисунке 1.3 представлена диаграмма рассеяния опять же сгенерированных случайных данных. Данные двумерные и разделяются на два класса.

По прежнему, нет никакой сложности в том, чтобы провести вручную линию с полной точностью отделяющую один класс от другого (хотя, в отличии от одномерно примера, уже понадобиться знания аналитической геометрии чтобы записать классифицирующее правило). Однако, в целях изучения метода, взглянем на то как с этой задачей справиться логистическая регрессия. На рисунке А.1 двумерный аналог рисунка 1.2 – соответствующая рассматриваемому примеру двумерная сигмоида. На нее нанесены наблюдения с рисунка 1.3.

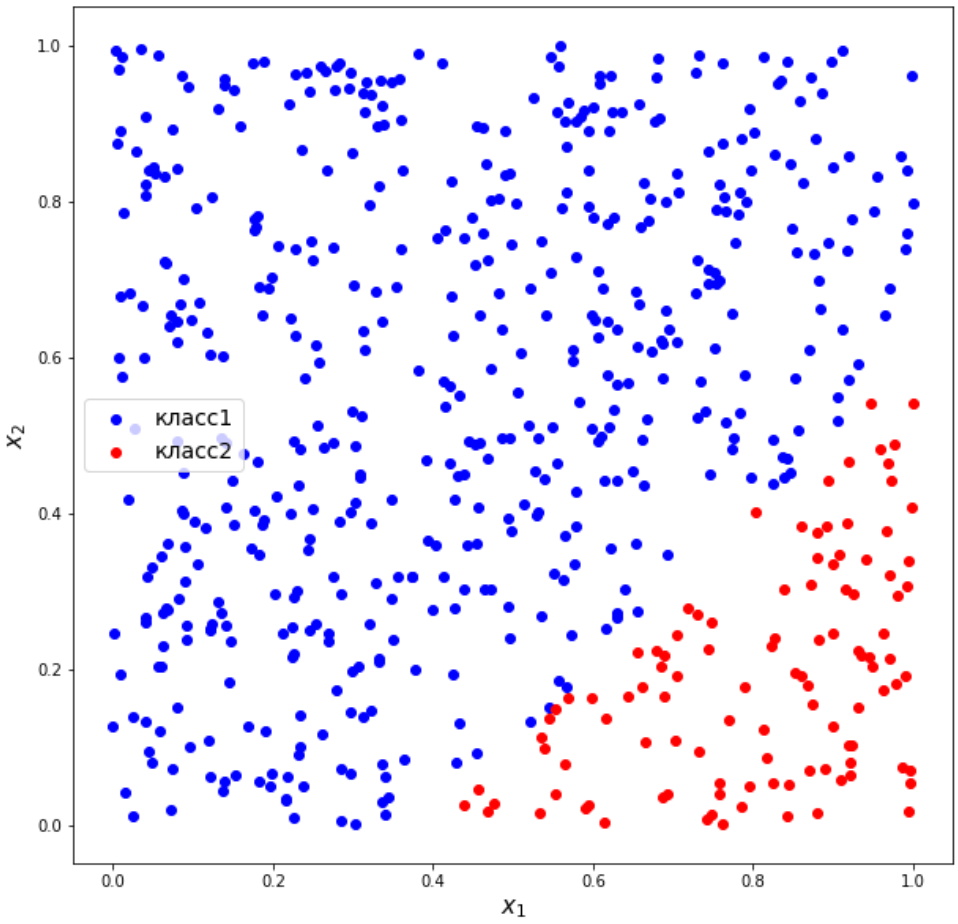

Рисунок 1.3 – Диаграмма рассеяния двумерных классифицированных данных
Примечание – Источник: собственная разработка.

Из рисунка понятно, что можно подобрать такую высоту p' что бы точки принадлежащие отдельным классам были разделены, а соответствующая линия уровня «упадет» на рисунок 1.3 так что идеально разделит классы и в отрыве от высоты сигмоиды. Покажем, что такая линия уровня будет линейной – она возникает при сигмоиде равной p':

$$p' = \frac{1}{1 + \exp(-\alpha - \sum_{j=1}^{m} x_j \beta_j)}. \tag{1.10}$$

Требуется разрешить уравнение относительно выражения под сигмоидой. Такую операцию мы уже продевали раньше, только наоборот, потому просто запишем результат:

$$\alpha + \sum_{j=1}^{m} x_j \beta_j = -\ln\left(\frac{1 - p'}{p'}\right).$$

Заметив, что правая часть выражения константа, скажем, что разделяющая линия уровня линейна.

Тут и рождается «почва» для дальнейшей эволюции методов классификации связанных с логит функцией. Как и ранее рассмотрим пример с которым данный метод справится плохо – рисунок 1.4.

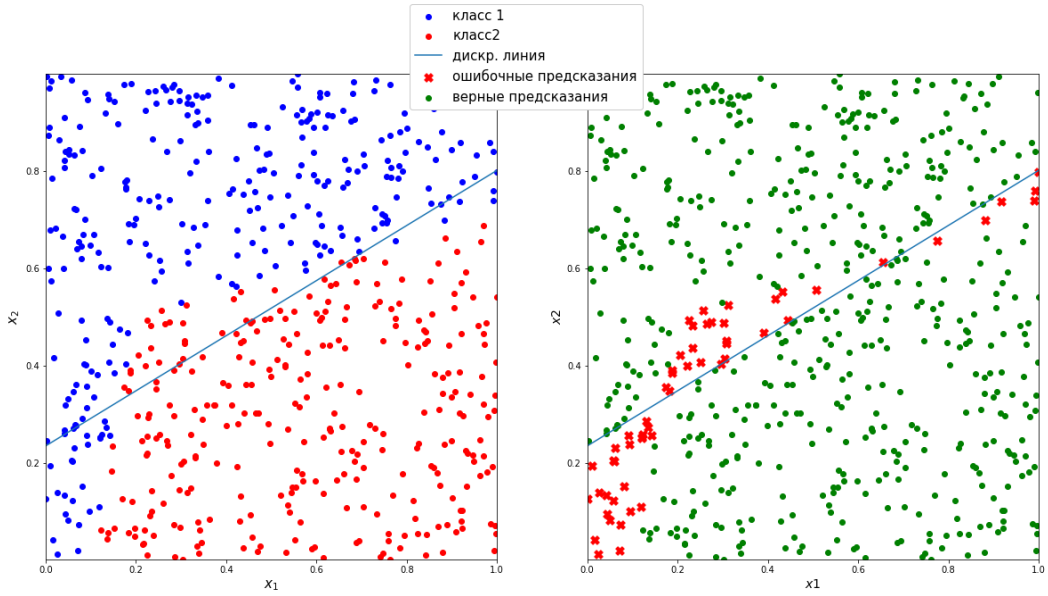

Рисунок 1.4 – Диаграмма рассеяния двумерных классифицированных данных при нелинейном принципе классификации

Примечание – Источник: собственная разработка.

Слева все та-же диаграмма рассеяния, только несколько изменены принципы классификации. По по-прежнему можно вручную записать правило разделяющее классы, только в этот раз это будет не единственные уравнение прямой, но кусочно-линейная функция.

На этих данных была построена модель логистической регрессии. Выбрав p' по принципу наибольшей разности между распределениями классов по p (этот принцип будет раскрыт в последующих главах) и использовав (1.10) мы получили уравнение описывающее дискриминирующую прямую, так же нанесенную на график.

На рисунке справа более броско выделены ошибки модели. Как видно модель логистической регрессии, как и любая другая линейная модель с таким случаем справляется не идеально (хотя задача достаточно простая). Решение заключается в том, чтобы включить в модель некоторую нелинейность.

Этот подраздел посвящен логистической регрессии – модели которая не является целевой для данной работы, однако идеи в ней лежащие важны для полного раскрытия темы. Описаны плюсы по сравнению с линейной регрессией и выведены формулы позволяющие записать модель аналитически. Особо важной для дальнейшего повествования является логит функция (сигмоида, обратная функция логистического распределения). Показано на примере, почему логистическая регрессия является линейным классификатором и описаны причины дальнейшей эволюции методов классификации.

1.3 Модель искусственной нейронной сети в классификации

Как и любая другая математическая модель, нейронная сеть может быть записана как система уравнений, но куда проще и понятнее рассматривать ее как своеобразный граф по ребрам которого «текут» данные и преобразуются в узлах.

Базовой единицей выступает нейрон – узел графа. Раскроем его сруктуру. На рисунке 1.5 схематично представлен искусственный нейрон.

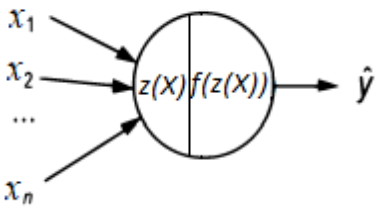

Рисунок 1.5 – Схема искусственного нейрона

Примечание – Источник: собственная разработка на основе [4 с. 52].

Стрелки входящие в нейрон называют входными активациями. По сути $x_i, i = \overline{1,n}$ просто число для отдельного примера с которым работает модель. Функция $z(X), X = (x_1, x_2, \dots, x_n)$ называется сумматорной, она отвечает за восприятие нейроном входных активаций. Функция $f(z(x))$ – активационная (придаточная функция) отвечает за формирование выходной активации. Обозначение $\hat{y}$ на описываемой диаграмме представляет выходную активацию нейрона.

Фактически нет ограничений на вид сумматорной и активационной функций, но в качестве сумматорной функции, как правило, используется просто линейная комбинация входных активаций:

$$z(X) = \sum_{i=1}^{n} \omega_i x_i + b. \qquad (1.11)$$

где x_i – *i-я* входная актвация;

ω_i – вес *i-й* входной активации;

b – свободный член.

В данной работе не рассматривается моделей с сумматорной функцией другого вида, потому, впереть, при любом упоминании сумматорной функции имеется ввиду выражение (1.11). Давайте, для краткости записи за таким выражением по умолчанию, просто закрепим обозначение z. А рисунок 1.5 примет вид как на рисунке 1.6.

Даже в базовой литературе в области нейронных сетей разнообразие активационных функций куда шире нежели используется в этой работе. Можно сказать, что в случае схемы 1.6 вид нейрона полностью определяется видом его активационной функции. В этой работе используются лишь два вида нейронов: сигмоидальный и ReLU.

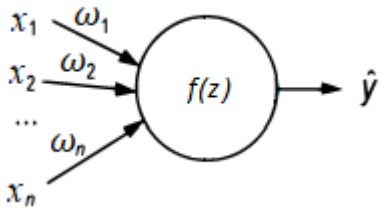

Рисунок 1.6 – Схема искусственного нейрона при линейной сумматорной функции
Примечание – Источник: собственная разработка.

С активационной функцией сигмоидального нейрона мы уже встречались раньше, это ничто иное как логит функция:

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

ReLU (Rectified linear unit) нейрон имеет активационную функцию вида:

$$f(z) = \begin{cases} 0, z < 0 \\ z, z \geq 0 \end{cases}. \quad (1.12)$$

Набравшись некоторой теории касательно аналитической записи отдельных нейронов, перейдем к рассмотрению того как нейроны объединяют в нейронные сети.

Выделяют ряд нейросетевых архитектур, но самая простая – архитектура прямого распространения. Слоем нейронной сети, в контексте сетей прямого распространения, будем называть множество нейронов одного типа, получающих информацию только от нейронов предыдущего слоя и передающих информацию только нейронам следующего слоя. Слои выстраиваются один за другим и формируют цепочку преобразований входных данных. Далее любая сеть о которой пойдет речь будет предполагаться сетью с архитектурой прямого распространения.

В общем такая нейросетевая архитектура может быть представлена в виде рисунка 1.7.

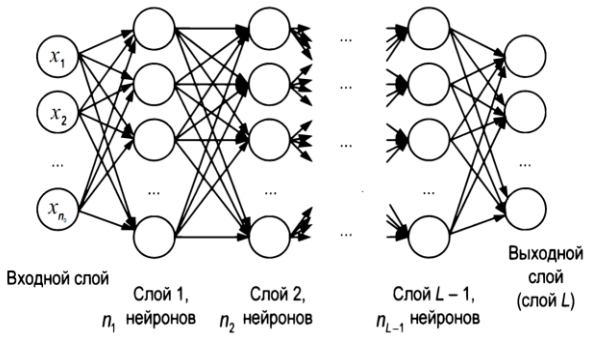

Рисунок 1.7 – Обобщенная нейросетевая архитектура
Примечание – Источник: собственная разработка на основе [4 с. 95].

Входной слой (нулевой слой, слой с номером 0) это даже не в полной мере нейроны – это представление на рисунке, того как в модель попадают входные данные. Все последующие слои с номерами от *1-го* до *(L-1)-го* называют скрытыми слоями нейронной сети. Активации выходного (слой с номером L) слоя представляют собой предсказания нейронной сети, для данных заявленных во входном слое.

Более детально рассмотрим взаимодействие соседних слоев сети в терминах введенных выше. Обозначим $\omega_{ji}^{l}, j = \overline{1, n_l}, i = \overline{1, n_{l-1}}$ – вес выходной активации *i-го* нейрона *(l-1)-го* слоя в суммарной функции *j-го* нейрона *l-го* слоя. В этом обозначении легко запутаться, потому при необходимости можно посматривать на рисунок 1.8.

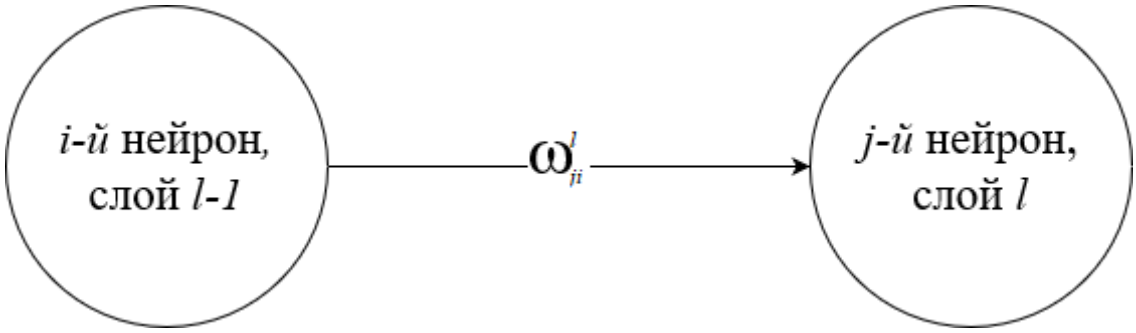

Рисунок 1.8 – Обозначения связывания соседних слоев
Примечание – Источник: собственная разработка.

В целом, процесс идентификации нейронной сети и заключается в выборе последовательности слоев нейронов различных видов и числа нейронов входящих в каждый из слоев. Не возникает сомнений, что архитектура искусственной нейронной сети может принимать очень разный вид, особенно для различных задач, но даже одна и та-же задача может иметь несколько обоснованных архитектур.

В этой работе будет рассмотрена архитектура, которую можно рассматривать как развитие идеи логистической регрессии, перейдем к ее описанию.

В предыдущем разделе было показано, почему логистическая регрессия является линейным классификатором и обоснована необходимость добавления некоторой нелинейности в названную модель.

Возвращаясь к сигмоидальному нейрону получается, что модель логистической регрессии может быть представлена как нейронная сеть без скрытого слоя. Для биномиальной логистической регрессии такая архитектура представлена на рисунке Б.1. Для перехода к мультиномиальной достаточно добавить в выходной слой столько нейронов сколько имеется классов.

Основная идея рассматриваемой в этой работе архитектуры состоит в том, чтобы добавить в модель скрытые слои содержащие ReLU нейроны. Схематично такая архитектура представлена на рисунке Б.2. Получится, что если выразить сумматорную функцию выходного слоя через веса соединяющие различные слои

и активации входного слоя, то под сигмоидой станет кусочно-линейная функция, что обеспечит некоторую нелинейность при принятии решения.

Для простоты рассмотрения механизмов запрятанных в этой модели вернемся к примеру из предыдущего раздела с которым логистическая регрессия справилась плохо. На рисунке 1.9 представлена та-же задача, пока сконцентрируемся на графике слева.

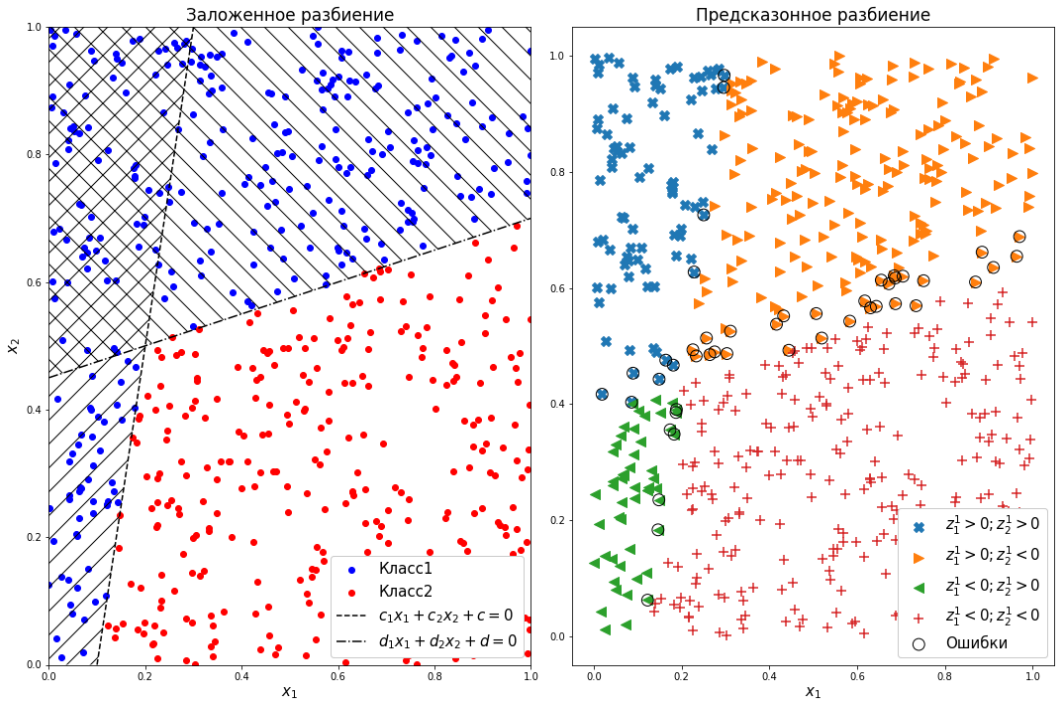

Рисунок 1.9 – Решение задачи с помощью нейронной сети

Примечание – Источник: собственная разработка.

Теперь на график нанесены линии соответствующие уравнениям которые я положил в принцип распределения по классам. Заметим, что таких линии всего две. Забегая вперед, скажем, что для решения такой задачи, достаточно нейронной сети с одним скрытым слоем и всего двумя нейронами в нем, как на рисунке 1.10.

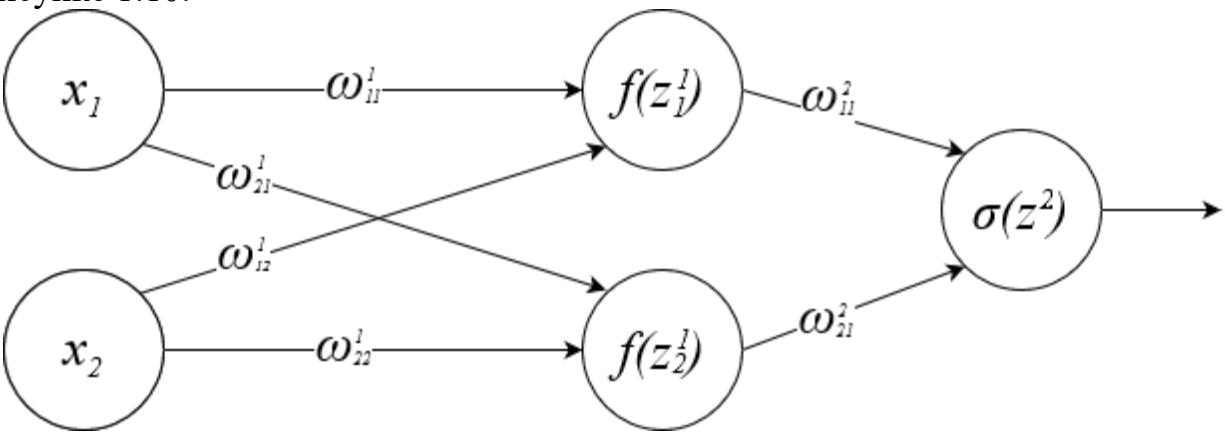

Рисунок 1.10 – Нейронная сеть для кусочно-линейной классификации

Примечание – Источник: собственная разработка.

Распишем аналитически, эту модель. Сигмоиду, которая является последним преобразованием этой модели, подробно обсудили выше, потому сразу начнем с записи сумматорной функции выходного слоя z^2.

$$z^2 = \omega_{11}^2 f(z_1^1) + \omega_{21}^2 f(z_2^1) + b^2,$$

где z_i^j – значение сумматорной функции *i-го* нейрона *j-го* слоя;

b^l – свободный член *l-го* слоя.

Используя определение ReLU функции (1.12) можно последнюю формулу разложить на четыре случая:

$$z^2 = \begin{cases} b^2, & z_1^1 < 0 \text{ и } z_2^1 < 0; \\ \omega_{11}^2 z_1^1 + b^2, & z_1^1 \geq 0 \text{ и } z_2^1 < 0; \\ \omega_{21}^2 z_2^1 + b^2, & z_1^1 < 0 \text{ и } z_2^1 \geq 0; \\ \omega_{11}^2 z_1^1 + \omega_{21}^2 z_2^1 + b^2, & z_1^1 \geq 0 \text{ и } z_2^1 \geq 0. \end{cases} \quad (1.13)$$

Для решеная задачи классификации будет хорошо если каждая область, по штриховке на рисунке, получит свой вид решающего правила – получиться так, что, например, область синих точек отделенная обеими дискриминирующими линиями будет получать вероятности принадлежности ко второму классу по своему отдельному уравнению которое сформируется так чтобы дать наименьшие вероятности. Аналогично, но менее однозначно, вероятности будут формироваться для областей с единственной штриховкой. Для области без штриховки, очевидно, правило должно выстроиться так, чтобы давать наибольшие вероятности.

Зная законы по которым формировалась выборка из примера, можно согласовать веса первого слоя так, чтобы каждому случаю из формулы (1.13) соответствовала своя область рисунка. Это можно сделать несколькими способами, но пусть незаштрихованной области соответствует первый, области с наклоненной влево штриховкой второй, вправо – третий и области с двумя штриховками остается четвертый.

Покажем, как провести данное согласование для такого рисунка, хотя в общем это может выглядеть по другому – все зависит от того как проведены дискриминирующие линии.

На легенде рисунка обозначены уравнения дискриминирующих линий, начнем с первого. В данном случае области с наклоненной влево штриховкой соответствует неравенство:

$$c_1 x_1 + c_2 x_2 + c > 0.$$

Потому присвоив весам нейронной сети $\omega_{11}^1 := c_1$, $\omega_{12}^1 := c_2$ и свободному члену первого нейрона первого слоя $b_1^1 := c$, получим, что второй случай

формулы (1.13) действительно соответствует области с наклоненной влево штриховкой.

Области с наклоненной вправо штриховкой соответствует неравенство:

$$d_1 x_1 + d_2 x_2 + d < 0.$$

Потому присвоив весам нейронной сети $\omega_{12}^1 \coloneqq -d_1$, $\omega_{22}^1 \coloneqq -d_2$ и свободному члену второго нейрона первого слоя $b_2^1 \coloneqq -d$, получим, что третий случай формулы (1.13) действительно соответствует области с наклоненной вправо штриховкой.

При описанном выше распределении весов области с двумя штриховками автоматически будет соответствовать последний случай формулы (1.13). Так получиться, что первый слой идеально разделит область наблюденных данных на четыре части, что обеспечит такую ситуацию, что сигнал в выходной нейрон будут подавать только наблюдения обозначенные синим.

Выше было обозначено, что в рамках этого примера мы будем моделью оценивать вероятность того, что конкретное наблюдение принадлежит к второму классу обозначенному красным. Для того, чтобы оценки вероятностей области без штриховки были выше нежели любые другие, можно весам второго слоя присвоить любые отрицательные значения – положительный сигнал выходящий из первого слоя в результате попадания в модель любого «синего» наблюдения будет помножаться на отрицательное число и занижать сумматорную функцию z^2 и, как следствие, занижать оценки вероятностей первого класса, чего мы и добивались. В данном случае свободный член сумматорной функции выходного нейрона может принимать любое значение – все равно механизм описанный выше будет занижать оценки вероятностей для любого наблюдения из первого класса относительно наблюдений из второго класса, что обеспечит нам стопроцентную точность классификации.

Все это мы вели к тому, что можно моделью нейронной сети архитектуры, представленной на рисунке 1.10, добиться идеальной классификации рассматриваемой задачи – ограничения логистической регрессии преодолены.

Однако, в любой реальной задаче классификации принципы разделения на классы, конечно, неизвестны, потому веса и свободные члены каждого слоя оцениваются статистически. На рисунке 1.9 справа показана работа реально обученной только на статистических данных нейронной сети. Для каждого наблюдения мы вычислили сумматорные функции нейронов первого слоя и обозначили на рисунке формой и цветом как они распределились по знаку названной функции. Присмотревшись к такой диаграмме рассеяния мы заметили, что без ошибок не обошлось – они выделены на рисунке черными кружками.

При построении моделей опирающихся на искусственные нейронные сети, кроме множества возможностей для идентификации нейронной сети, существуют возможности настройки алгоритма обучения, что может вести вообще говоря, к различным моделям. Модель для поставленной задачи, точно

можно привести к идеальной классификации, но этот пример нам даже на руку, можно показать роль, которую в финальном решении играют веса выходного слоя.

Как зависят предсказываемые вероятности от конкретной точки в системе координат представлено на рисунке А.2 – сигмоида состоящая из двух участков под разными углами. И несмотря на то, что ряд пограничных точек приводят к формированию сигнала в выходной слой (хотя по заложенным закономерностям недолжны), веса и свободный член выходного слоя придали им большую сумматорную функцию выходного слоя, нежели для наблюдений «синего» класса. Это привело к тому что все точки относящиеся ко второму классу на графике А.2 выше, нежели точки первого класса – от сюда, правильно выбрав высоту p', можно добиться 100% точности классификации.

Этот подраздел работы вводит понятие искусственной нейронной сети, с краткими наиболее общими идеями лежащими в основе этой группы моделей. Нейронная сеть представляет собой последовательность преобразований входных сигналов, каждое из которых в своей сумматорной функции сочетает выходные сигналы предыдущего слоя и к этому сочетанию применяет активационную функцию результат которой отправляется в сумматорные функции следующего слоя или, для выходного слоя, формирует предсказание модели. Далее вводиться архитектура нейронной сети в которой скрытые слои представляют собой ReLU функции а выходной слой содержит сигмоидальный нейрон. На примере подробно раскрыты механизмы почему эта модель работает, как она связана и развивает идеи модели логистической регрессии.

В этом разделе основное внимание было уделено тому как названная модель применяется для известной закономерности, но на практике закономерности неизвестны, а имеются лишь наблюденные данные и идентификацию и оценку параметров надо производить в этих условиях. В следующем подразделе описаны идеи, как производиться оценка параметров, а более детальная идентификация модели в контексте нейронных сетей лучше всего раскрывается на практике, и будет представлена в третьей главе.

1.4 Целевые функции и алгоритм обратного распространения ошибки

Процесс оценки коэффициентов в контексте нейронных сетей принято называть обучением модели.

Ключевым пунктом является выдвижение некоторого правила которое оценивает насколько модель соответствует наблюденным данным – целевую функцию. В общем, требуется ввести функцию:

$$F\left(Y, \hat{Y}\right),$$

где Y – реально наблюденный вектор предсказываемого явления;

$\hat{Y}$ – вектор текущих предсказаний.

Очевидно, что вектор предсказаний формируется данными, проходящими через модель т.е. правомерна запись:

$$\hat{Y} = \hat{Y}(X, W),$$

где X – реальное множество наблюдений за факторами для которых предполагается влияние на Y;

W – множество коэффициентов которые на разных этапах оказывают влияние на отклик вычисляемый моделью.

Итак, окончательно, обобщенная целевая функция примет вид:

$$J(Y, X, W).$$

Её значение должно быть тем больше, чем сильнее отличаются наблюденные и предсказанные значения. Нам выгодно, чтобы полученные предсказания были максимально похожи на наблюденные, потому нам тем лучше, чем меньше эта функция.

В контексте рассматриваемого вопроса можно сказать, что множества X и Y неизменны (хотя некоторые продвинутые обучающие алгоритмы могут использовать на разных стадиях обучения разные подмножества этих множеств).

Приходим к тому, что необходимо подобрать такие коэффициенты W, чтобы целевая функция J была минимальна, или более формально:

$$W^* = argmax_W[J(Y, X, W)].$$

По выводам предыдущего подраздела очевидно, что в случае нейронной сети $\hat{Y}(X, W)$ – нелинейная функция, да и все широко применяемые целевые функции также не линейны, потому очевидно, что перед нами стоит задача нелинейной оптимизации.

Для решения таких задач широко распространена группа методов основанных на градиентном спуске. Основная идея этих методов заключается в том, чтобы постепенно от некоторой выбранной начальной точки двигаться в направлении наискорейшего убывания J.

В методах градиентного спуска используется свойство антиградиента функции, которое утверждает, что функция в каждой точке убывает быстрее всего в направлении ее антиградиента, который определяется так:

$$-\nabla J = -\frac{dJ}{dW} = -\left(\frac{dJ}{d\omega_1}, \frac{dJ}{d\omega_2}, \dots, \frac{dJ}{d\omega_n}\right),$$

где ω_i – некоторый отдельный коэффициент модели $i = \overline{1, n}$.

Возвращаясь к искусственным нейронным сетям, приходим к тому, что единственная сложность реализации этого алгоритма заключается в том, чтобы получить частные производные по всем весам. К решению этой проблемы

призван метод обратного распространения ошибки. Будем его рассматривать как надстройку над методами градиентного спуска.

В процессе разъяснения принципа метода обратного распространения ошибки нам пригодится рисунок 1.11.

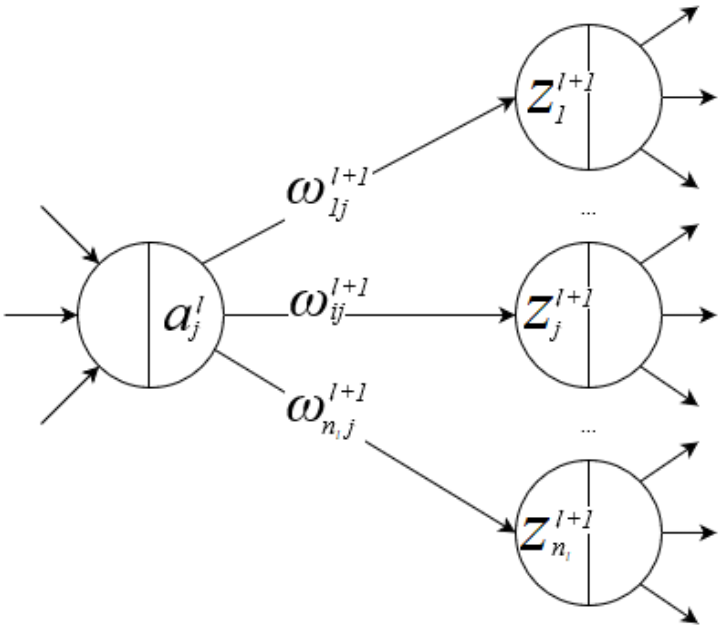

Рисунок 1.11 – Связь соседних слоев
Примечание – Источник: собственная разработка.

И некоторые обозначения. Матрица весов *(l+1)-го* слоя

$$W^{l+1} = (\omega_{ij}^{l+1})_{n_{l+1} \times n_l}.$$

Вектор из весов помножаемых на *j-ю* активацию *l-го* слоя, это-же столбец *j* матрицы весов *(l+1)-го* слоя

$$(W^{l+1})_j. \tag{1.14}$$

И так называемая, ошибка *j-го* нейрона *l-го* слоя, определяемая так:

$$\delta_j^l = \frac{\partial J}{\partial z_j^l}. \tag{1.15}$$

Выясним взаимосвязь ошибок *l-го* и *(l+1)-го* слоев. Для того придадим некоторое малое приращение Δ активации a_j^l и посмотрим какое приращение при этом получат сумматорные функции *(l+1)-го* слоя. В силу линейности сумматорных функции приращение для z_j^{l+1} составит $\Delta\omega_{ij}^{l+1}, j = \overline{1,n}$.

Пользуясь свойством, что при достаточно малом приращении аргументов соответствующее приращение функции близко к скалярному произведению градиента и вектора приращений аргументов получаем:

$$\Delta J\left(z_1^{l+1}, \dots, z_j^{l+1}, \dots, z_{n_{l+1}}^{l+1}\right) = \\ = \left(\frac{\partial J}{\partial z_1^{l+1}}, \dots, \frac{\partial J}{\partial z_j^{l+1}}, \dots, \frac{\partial J}{\partial z_{n_{l+1}}^{l+1}}\right)\left(\Delta\omega_1^{l+1}, \dots, \Delta\omega_j^{l+1}, \dots, \Delta\omega_{n_{l+1}}^{l+1}\right).$$

Или, используя свойство ассоциативности скалярного произведения, обозначения (1.14) и (1.15):

$$\Delta J(z^{l+1}) = \Delta\delta^{l+1}(W^{l+1})_j,$$

где z^{l+1} – вектор из значений сумматорной функции *(l+1)-го* слоя;

δ^{l+1} – вектор ошибок *(l+1)-го* слоя.

Разделив обе части уравнения на Δ, устремив его к нулю и вспомнив, что это приращение активационной функции a_j^l запишем:

$$\frac{\partial J}{\partial a_j^l} = \delta^{l+1}(W^{l+1})_j.$$

Далее вспоминая, что a_j^l, функция от z_j^l и, используя правило дифференцирования сложной функции:

$$\frac{\partial J}{\partial z_j^l} = \frac{\partial J}{\partial a_j^l}\frac{da_j^l}{dz_j^l} = \delta^{l+1}(W^{l+1})_j\frac{da_j^l}{dz_j^l}.$$

Опять обратившись к (1.15) и переходя к матричной форме записи, получим:

$$\delta^l = (W^{l+1})^T\delta^{l+1} \odot \frac{da_j^l}{dz_j^l}. \qquad (1.16)$$

Формула (1.16) самый важный вывод метода обратного распространения ошибки, собственно она отражает название метода – каждая предыдущая ошибка итеративно вычисляется из следующей, конечно, кроме ошибки выходного слоя, которую можно записать так:

$$\delta_j^L = \frac{\partial J}{\partial z_j^L} = \frac{\partial J}{\partial a_j^L}\frac{da_j^L}{dz_j^L}.$$

Или переходя к матричной записи:

$$\delta^L = \frac{dJ}{da^L} \odot \frac{da_j^L}{dz_j^L}. \tag{1.17}$$

Заметим также, что основной целью было названо выделение частных производных целевой функции по весам и свободным членам. Учитывая (1.15) и то что z_j^l функция от весов и свободных членов, не составляет труда очередной раз использовать правило дифференцирования сложной функции и получить:

$$\begin{aligned} \frac{\partial J}{\partial b_j^l} &= \frac{\partial J}{\partial z_j^l}\frac{\partial z_j^l}{b_j^l} = \delta_j^l; \\ \frac{\partial J}{\partial \omega_{ji}^l} &= \frac{\partial J}{\partial z_j^l}\frac{\partial z_j^l}{\partial \omega_{ji}^l} = a^{l-1}\delta_j^l. \end{aligned} \tag{1.18}$$

К сожалению, ни в одном из источников мы не нашли причины, по которой, используется именно такой подход, а не просто аналитически выводятся производные по весам и смещениям. Но можно предположить, что такой подход однозначно проще подлежит программированию – достаточно знать лишь производные активационных функций по их аргументам, и вне зависимости от слоя в котором находиться нейрон, зная его активацию, для текущих параметров модели можно получить производную по весам и смещениям используя одни и те-же рекуррентные формулы (1.18).

Заметим также, что рассуждения, касающиеся этого метода, мы строили в отрыве от наблюденных данных *X, Y*. Все выше сказанное будет справедливо для каждого конкретного примера по отдельности, а должны быть учтены все примеры участвующие в обучении. Благо все целевые функции построены на том, что суммируют ошибки отдельных примеров, а производная суммы, как известно, сумма производных потому результаты полученные из (1.18) надо просто просуммировать для каждого отельного примера и будет получена действительно производная целевой функции.

И так, теперь, на основе метода обратного распространения ошибки сформируем одноименный алгоритм оценки параметров нейронной сети:

1. Выполнить прямое распространение активации для каждого примера (подставить *X* в модель), сохраняя промежуточные активации;
2. Использовать формулу (1.17) для вычисления ошибок выходного слоя;
3. Произвести обратное распространение ошибки используя формулу (1.16) – будут получены ошибки всех нейронов сети;
4. Используем формулы (1.18) просуммировав их значения для каждого примера – будут получены градиенты по весам и смещениям;
5. Обновление параметров вдоль антиградиента;

Описанные пункты повторяются до тех пор пока не будут выполнены условия остановки алгоритма выбранной вариации градиентного спуска. [5 с. 243]

В целом, эта глава была посвящено методам классификации. Изначально была раскрыта подстановка задачи – требуется на основе наблюденных данных сформировать правило, которое получив некоторую информацию об объекте сможет отнести его к нужному классу. В приложении к кредитному риску: на основе данных предоставляемых клиентом в заявке на получение кредита, требуется принять решение о выдаче или удержании займа.

Описанная задача не нова и для ее решения широко используются, уже ставшие классическими, методы: логистическая регрессия и дискриминантный анализ. В начале главы описаны предпосылки использования логистической регрессии – невозможность решения поставленной задачи обычными регрессионными моделями объясняется, тем что эта группа моделей предсказывает число, в то время как нам требуется получить предсказание класса. Логистическая регрессия обходит это ограничение благодаря свойствам логит функции, которые позволяют предсказанное значение интерпретировать как вероятность отнесения к некоторому классу. В подразделе 1.2 подробно описывается процесс перехода к модели именно такой формы.

В том же разделе мы указали и на ограничения модели логистической регрессии – несмотря на очевидную нелинейность используемой формулы модель логистической регрессии остается линейным классификатором.

Для преодоления этого ограничения мы использовали концепцию нейронной сети, рассмотренной как развитие модели логистической регрессии через усложнение формы функции от показателей лежащей под логит функцией. По сути вместо линейной функции под логит функцию была положена кусочно-линейная функция. В подразделе 1.3 мы постарались раскрыть как этот механизм приводит к улучшению классифицирующих свойств модели в некоторых случаях.

Завершается глава исчерпывающим описанием математики используемой для формирования алгоритма оценки коэффициентов нейронной сети – метода обратного распространения ошибки. В целом, это тот же численный метод оптимизации как градиентный спуск, но частные производные целевой функции вычисляются особым способом, очевидно, особенно удобным для программирования.

На практике, кроме оценки коэффициентов, для модели требуется дополнительно еще целый комплекс мер связанных подготовкой данных. Кроме того в идентификация модели нейронной сети это отдельное «искусство» неразрывно связано с методом подбора решения – хитрости и уловки используемые соответствующими специалистами лучше всего раскрывать на практике. Этим вопросам и посвящены следующие разделы этой работы.

2 Анализ и подготовка данных

2.1 Начальный анализ наблюденных данных

Сразу укажем, что все действия связанные с вычислениями и обработкой информации мы производили на языке программирования python3 – одном из самых популярных и востребованных, на сегодняшний день, инструментов для математического моделирования и обработки данных. В работе использованы библиотеки-расширения pandas [6], numpy [7], scipy [8], sklearn [9], pytorch [10]. Весь исходный код в формате jupyter notebook или исполняемых «.py» файлов, результаты его выполнения в необработанном виде и даже текст этой работы можно найти на предварительно созданном репозитории[1] в сети интернет.

В связи с коммерческой тайной мы не можем предоставить данные использованные для построения модели в открытый доступ, приходиться принять, вызванную этим фактом, непрозрачность описываемого исследования. Тем не менее, мы на словах и промежуточных результатах вычислений постараемся по максимуму раскрыть полный цикл разработки модели – от «сырой» таблицы показателей и до валидации готовой модели.

Этот раздел посвящен анализу полученной из баз данных банка информации и, на его основании, обоснования и применения решений связанных с преобразованием данных способствующих дальнейшему успеху при моделировании.

Для преобразования данных зачастую пишут отдельную программу со своими настройками, так как по результатам моделирования может потребоваться подгонка еще и параметров предварительной обработки данных – в этом случае следует просто поменять некоторые настройки процессинга данных и забрать по сути новый набор данных. Такую программу в западных источниках называют «pipeline» (пайплайн), мы будем называть конвейер данных. Учитывая, что конвейер данных собирается под конкретную структуру данных и то, что нет возможности предоставить данные в свободный доступ, не вижу никакого смысла, в техническом представлении этой программы в работе. Периодически лишь будут мелькать обобщенные методы для конкретных преобразований и представления результатов вычислений.

Та часть практики которая связана с аналитикой и подготовкой данных может быть найдена в папке «data_processing» названного репозитория. Особое внимание следует обратить на файл «processing1.ipynb», его можно открыть прямо в браузере.

Изначально входной файл идет в формате «.xlsx» – его, при соблюдении внутри структуры как в таблице 1.1 можно в одну строку открыть с помощью метода «read_excel» библиотеки pandas. На выходе будет получен «padnas.DataFrame» – объект представляющий собой данные вида таблицы 1.1. Заметим, что большой объем данных из формата «.xlsx» загружается достаточно долго, потому его лучше преобразовать в формат «.csv», который значительно

[1] https://github.com/Dranikf/diplom_project

проще. Сделать это можно прямо внутри pandas используя метод «to_csv» объекта «padnas.DataFrame», важно в аргумент «index» передать значение «none» для того, чтобы не был сохранен еще и столбец с индексом который был автоматически создан при загрузке. После этой процедуры можно пользоваться созданным «.csv» файлом, загружая его функцией «read_csv» библиотеки «pandas».

И так, изначальный набор данных имеет 247062 записей и 45 столбцов-показателей. Для начала нужно разобраться с составом столбцов, их типом данных и структурой. Все эти операции можно провести используя только возможности pandas, но удобнее, чтобы это можно было сделать в одну строку кода. Для решения такой задачи создана функция языка программирования python «get_data_frame_settings» исходный код представлен на листинге В.1. Применив ее к объекту с данными, на выходе будет получен, также «pandas.DataFrame». Для наших данных результат представлен в таблице Г.1.

Первый столбец содержит название показателя. Второй столбец посвящен типу данных, конечно в pandas типы данных другие – но в листинге В.1 объявлен словарь «types_natural_names» в котором реальным названиям типов данных pandas поставлены в соответствие русскоязычные названия. Можно заметить, что перечислены не все типы данных pandas и те которые не упомянуты будут отображаться так, как они бы выглядели в pandas, в случае необходимости можно добавить свой пункт в этот словарь. И так, у нас в таблице имеются данные следующих типов – дата, целое число, действительное число и номинативная переменная.

Третий столбец содержит информацию о возможных значениях которые принимает соответствующий показатель. Для дат и чисел это будет интервал, для номинативных показателей перечисленные через запятую уровни наблюденные в этом столбце. Сразу заметим, что показатели «Адрес проживания - Населенный пункт», «Вид деятельности по ОКЭД», «Кредитный продукт» и «Место работы» принимали настолько много уровней, что все их выписать в эту работу не представлялось возможным, из соображений оформления, потому они были заменены.

Четвертый столбец, для номинативных переменных содержит число уровней которые они принимают.

Отдельную сложность в подготовке и анализе данных представляют ячейки с пропущенными значениями, в последнем столбце таблицы Г.1 выписаны количества таких наблюдений. Некоторые столбцы содержат до 95% пропусков.

Теперь, отталкиваясь от этой таблицы попробуем принять некоторые преобразования данных.

Особую ценность для нас составляет столбец «Дефолт», в нем содержится, целевая для нас информация – сколько дней, на момент среза, таблицы каждый контракт находиться в состоянии просрочки платежа. По указанию начальства, для целей моделирования, вышедшим в дефолт считается контракт, у которого в этом столбце наблюдается значение свыше 59 дней. Произведем перекодировку в новый столбец «*Y*» по правилу:

$$Y = \begin{cases} 1, \text{Дефолт} < 60; \\ 0, \text{в противном случае.} \end{cases}$$

Исходный столбец «Дефолт» подлежит удалению.

Выше уже упоминались столбцы «Вид деятельности по ОКЭД», «Кредитный продукт» и «Место работы». Было принято решение произвести их удаление. Все они содержат очень много уровней, что в условиях модели привело бы к колоссальному росту объема вычислений, хотя на решения модели это вряд ли как-либо повлияло – если некоторый уровень редко встречается, даже если все восхождения отметились как дефолт, нет оснований считать, что отловлена статистическая закономерность. Можно было бы попытать произвести укрупнение этих столбцов, но это было осложнено но названным ниже причинам.

Столбец «Вид деятельности по ОКЭД» идет в достаточно странном формате. Опираясь на общегосударственный классификатор видов экономической деятельности [11] удалось выяснить, что в уровнях присутствуют, как секции (буквенные обозначения), так и разделы, которые вообще говоря входят в секции. Непонятны причины такого разделения, потому столбец и было решено удалить.

Столбцы «Кредитный продукт» и «Место работы» подавались в строковом формате. Кончено, были «зацепки» для проведения некоторого парсинга, но из соображений экономии времени и не перспективности данных направлений работы было решено этого не делать.

Перейдем к показателю «овердрафт»: бинарный показатель который показывает является ли каждый конкретный кредит овердрафтом. Овердрафт – особый вид кредита, он предполагает открытие некоторого счета, в котором можно уйти в отрицательную сумму – только тогда появляется задолженность. Дело в том, что момент появления задолженности может не совпадать с моментом, когда была подана заявка на открытие такого счета и, как следствие, этот вид кредита имеет особую специфику. Кончено было бы идеально, чтобы модель автоматически учитывала этот факт – но это может значительно осложнить процесс моделирования, потому пока модель будет построена только для случаев без овердрафта. Соответственно предполагается удаление всех наблюдений содержащих уровень «да» в рассматриваемом столбце и сам столбец вырождается, потому подлежит удалению.

Основная цель моделирования в данном случае – построение некоторого правила, которое поможет оценить способность каждого конкретного кредитополучателя к возврату долга. Потому, все показатели должны быть известны на момент выдачи кредита. Мы заметили, что исходной таблице присутствуют показатели которые не могут быть известны заранее: «Отношение факт срока к плановому при прекращении КД», «Причина прекращения договора» и «Дата фактического закрытия»; такие показатели также подлежат удалению.

Нередко показатель в чистой форме имеет слабую взаимосвязь с исследуемым явлением или не имеет взаимосвязи вовсе, но если к нему применить некоторые преобразования – то он может быть куда более полезен. Сейчас мы просто покажем процесс создания ряда показателей, а в следующем подразделе займемся выбором наилучших.

В таблице Г.1 сразу попадаются на глаза показатели связанные с годом выпуска автомобиля, притом они очень невнятно подписаны и мне так и не удалось выяснить, чем они между собой отличаются. Можно лишь выдвигать предположения результатом которых станет некоторое комбинирование этих показателей. Так были созданы показатели «Автомобиль год выпуска» который представлял собой просто самую позднюю дату из исходных трех, «Число авто» в котором подчитывалось количество непустых значений исходных трех, есть ли авто который принимал значение «да» если находился хотя бы одно непустое значение в исходных.

Продолжая разговор о показателях которые содержат дату, можно сказать, что показатели «Дата планируемого закрытия» и «Дата регистрации договора» не могут быть использованы в модели, так как уже наблюденные даты никогда больше не повторяться в новых заявках. Но вот их разность покажет предполагаемый срок кредита, с этой идеей был создан показатель «Срок кредита в днях». Исходные показатели были удалены из выборки.

Дальнейшим развитием этого показателя послужит его использование вместе с показателем «Сумма договора». Разделив показатель «Сумма договора» на показатель «Срок кредита в днях» получим показатель «Ежедневный платеж».

Модель не воспринимает дату, как формат, потому преобразуем его в число как количество дней от базовой даты – 8 декабря 1991 года.

Заметим, что среди показателей с очень большим числом уровней также оказался показатель «Адрес проживания - Населенный пункт», но он был выше удален подобно «Вид деятельности по ОКЭД», «Кредитный продукт» и «Место работы». Для его относительно легко оказалось провести парсинг. Можно было выделить показатель «Столица», который принимал значение «да» если запись относилась к городу Минску и нет в противном случае. Аналогичные действия относительно легко было провести с областными центрами с результатом в виде бинарного показателя «Областной центр». Исходный показатель был заменен так что все наблюдения не относящиеся к одному из областных центров получили значение «нет информации». Прежде чем проводить описанные процедуры надо было обязательно подготовить исходный столбец – дело в том, что один и тот же уровень мог быть записан по разному в плане некоторых отдельных символов, например в выборке можно было найти 51 запись в которой был указан город «могилев» (с маленькой буквы и буквой «е» вместо «ё»), хотя большинство записей указаны по правилам – «Могилёв». Для того, чтобы обойти такую погрешность нужно все символы столбца перевести в нижний регистр и буквы «е» заменить на «ё».

По завершению этого этапа выборка уже несколько трансформировалась и это не может не отразиться на таблице Г.1. В таблице Г.2 представлена актуальная информация о показателях выборки.

Следующим шагом станет очистка выборки от выбросов. В основном, тут следует уделять внимание столбцу «Область допустимых значений» таблицы Г.2.

Сразу обратим внимание на показатели «Работа последнее место стаж лет», «Работа уровень дохода BYN», «Сумма договора» и «Ежедневный платеж», они по смыслу, очевидно, не могут содержать отрицательные значения, однако по проведенному исследованию получается, что есть записи с отрицательными числами. Очевидно, это некоторая ошибка заполнения, которую нам следует исправить. Будет неправильно удалять целую запись с ошибочным заполнением, так как по результатам анализа классифицирующей способности, проводимого далее, может быть удален целый столбец. Правильнее будет эти позиции заполнить пропусками, так информация, содержащаяся в прочих столбцах, будет сохранена.

Далее нам показались подозрительными размахи некоторых переменных: «Количество фактов просрочки по основному долгу», «Максимальное количество дней просрочки», «Общее количество запросов в КБ», «Сумма кредитных лимитов», «Сумма договора» и «Ежедневный платеж». Для быстрого исследования квантилей числовых показателей в pandas можно использовать следующий механизм – выделяя через оператор «[]» некоторый столбец получим объект типа «pandas.Series» у которого имеется метод «describe». В выводе названного метода будет информация о числе непропущенных наблюдений, среднем, стандартном отклонении, размахе, 25%, 75% персентилях и медиане. Для быстрого получения информации этой информации сразу по диапазону столбцов может быть использована функция «get_describes» нанесенная на листинге В.2. Так краткой записью:

```
get_describes(data[em_research_list]),
```

где data – pandas.DataFrame содержащий все исследуемые показатели;
em_research_list – список численных показателей.

Можно получить таблицу вида 2.1.

Таблица 2.1 – Описательные статистики столбцов с предполагаемыми выбросами

Статистика	Количество фактов просрочки по основному долгу	Максимальное количество дней просрочки	Общее количество запросов в КБ	Сумма кредитных лимитов	Сумма договора	Ежедневный платеж
Количество	77016,00	70482,00	104134,00	90858,00	236496,00	236495,00
Среднее	5,11	18,05	8,35	7719,67	5739,18	2,69
СКО	10,60	132,79	7,48	13222,99	13752,80	2,74
Мин.	0,00	0,00	0,00	0,00	0,00	0,00

Продолжение таблицы 2.1

25%	0,00	0,00	3,00	1187,16	600,00	1,14
50%	1,00	1,00	7,00	3800,00	1500,00	1,86
75%	5,00	8,00	11,00	8900,00	5000,00	3,35
Макс.	284,00	4471,00	222,00	678562,60	600000,00	196,24

Примечание – Источник: собственная разработка.

Особо важная для нас в данном случае информация содержится в стоках «75%» и «Макс.». Заметим, что в каждых из названных столбцов максимальное значение очень отдалено от 75-го персентиля, что, скорее всего, вызвано некоторыми исключительными случаями. Так как основная идея моделей на статистических данных – отловить центральную тенденцию, а наличие таких случаев в выборке может вести к смещению оценок параметров, то их обычно удаляют.

При отделении значений относящихся к выбросам можно пользоваться следующим правилом [11], выбросами считаются наблюдения лежащие вне интервала:

$$[x_{25} - 1{,}5(x_{75} - x_{25}), x_{75} + 1{,}5(x_{75} - x_{25})], \quad (2.1)$$

где x_{25} – 25%-ная персентиль исследуемого набора чисел;

x_{75} – 75%-ная персентиль исследуемого набора чисел.

Но в нашем случае нет проблем с левым хвостом, потому, предлагаю несколько модифицировать правило – выбросами считаются наблюдения лежащие вне интервала:

$$[0, x_{75} + 1{,}5(x_{75} - x_{25})]. \quad (2.2)$$

Для быстрой реализации этой формулы можно воспользоваться функциями «get_selcond_emiss_25_75» и «cut_emissioins», представленными на листинге В.3. Ключевой в данном случае является «get_selcond_emiss_25_75» она для переданного столбца типа «pandas.Series» сформирует бинарное условие выбора (массив бинарных значений, совпадающий, по размерности, с исходным – выбираются те значения исходного массива, для которых соответствующие из бинарного условия имеют значение «Истинна») значений не принадлежащих выбросам. Можно так же указать аргумент «constant» (по умолчанию 1.5) – множитель их формулы (2.1) и аргумент «cut_type» (по умолчанию «both») который покажет какую из сторон проранжированного ряда следует обрубить: «both» – обе; «right» – правую; «left» – левую.

Функция «cut_emissioins» надстройка над «get_selcond_emiss_25_75» и прямая реализация (2.2); получает «pandas.Series», возвращает «pandas. Series» без выбросов.

После применения созданных инструментов к столбцам вызвавшим вопросы таблица 2.1 трансформировалась в таблицу 2.2.

Таблица 2.2 – Описательные статистики после очистки от выбросов.

Статистика	Количество фактов просрочки по основному долгу	Максимальное количество дней просрочки	Общее количество запросов в КБ	Сумма кредитных лимитов	Сумма договора	Ежедневный платеж
Число	67783,00	62987,00	100253,00	83282,00	216364,00	224167,00
Среднее	2,04	3,04	7,42	4684,76	2667,00	2,25
СКО	2,89	4,18	5,28	4643,93	2931,37	1,48
Мин.	0,00	0,00	0,00	0,00	0,00	0,00
25%	0,00	0,00	3,00	1000,00	570,80	1,12
50%	1,00	1,00	6,00	3200,00	1118,88	1,76
75%	3,00	8,00	11,00	7000,00	4000,00	3,01
Макс.	12,00	20,00	23,00	20464,32	11600,00	6,67

Примечание – Источник: собственная разработка.

Дальнейшим шагом могла стать борьба с пропусками, но это лучше делать после того, как получены некоторые характеристики классифицирующей способности показателей, потому, как борьба с пропусками обычно связана с необходимостью исключать некоторые столбцы из выборки, а это лучше делать, учитывая взаимосвязи показателей – некоторый показатель может иметь такие сильные классифицирующие свойства, что лучше потерять часть наблюдений, но оставить этот показатель в модели.

По итогам данного раздела мы перешли от первоначальных неочищенных данных к почти полностью обработанным данным. Используя ряд написанных функций отвечающих за процессинг данных удалось: создать столбец отклика, принять решение о удалении номинативных показателей с очень большим числом уровней не подлежащих обработке быстро, удалению всех контрактов относящихся к категории «овердрафт», создание новых показателей, с целью организации выбора на этапе исследования взаимосвязей, преобразование оставшихся показателей типа «Дата» к числовому типу данных и очистка от невозможных значений и выбросов. Последняя перед процедурой отбора показателей структура данных представлена в таблице Г.3

2.2 Оценка классифицирующей способности и отбор предикторов

ROC анализ является распространенным методом оценки классифицирующей способности моделей. Однако, как будет показано далее, эту методологию можно применить не только как способ оценки качества уже готовой модели, но и как некоторый критерий оценки классифицирующей способности отдельного предиктора.

Рассмотрим общую идею ROC анализа. Пусть у нас получена модель, которая позволяет производить бинарную классификацию. Исследуемый признак Y может относиться к категории у которой не проявилось некоторого признака (Negative – N), или к той, у которой признак проявился (Positive - P). В отношении кредитного скоринга соответственно – клиент без дефолта и с дефолтом.

Тогда интуитивным способом оценить ее производительность, является получение предсказаний на основе модели и построение таблицы следующего вида 2.3.

Таблица 2.3 – Качество бинарного классификатора

Истинный класс	Предсказанный класс		Всего
	N	P	
N	TN	FP	N'
P	FN	TP	P'
Всего	$N*$	$P*$	n

Примечания:

1. Источник: [3 с. 165],
2. TN – число наблюдений для которых модель предсказала отсутствие признака и оказалась права,
3. FN – число наблюдений для которых модель предсказала отсудившие признака, хотя на самом деле он присутствовал,
4. FP – число наблюдений для которых модель предсказала присутствие признака, хотя на самом деле он отсутствовал,
5. TP – число наблюдений для которых модель предсказала наличие признака и оказалась права,
6. $N^* = TN + FN$ – число наблюдений для которых модель предсказала отсутствие признака,
7. $P^* = FP + TP$ – число наблюдений для которых модель предсказала наличие признака;
8. $N' = TN + FP$ – число наблюдений без проявления признака,
9. $P' = FN + TP$ – число наблюдений в которых было замечено наличие признака,
10. $n = N* + P* = N' + P'$ – общее число наблюдений в выборке.

На основании абсолютных чисел сложно делать выводы, потому вводят относительные показатели:

$$FPR = FP/N'$$
$$TPR = TP/P'$$

Соответственно доля ошибок и правильных предсказаний модели при предсказании появления исследуемого признака.

Как правило, модель не позволяет сразу однозначно определить класс, потому, при варьировании точки отсечения PD' будут получены различные показатели FPR и TPR. Для лучшего понимания взгляните на рисунок 2.1.

На оси абсцисс отложены вероятности наличия признака предсказываемые моделью, зеленой площадью представлено предполагаемое распределение клиентов с отсутствием исследуемого признака, красной – с присудившем. Предсказанные вероятности до PD' относят соответствующие наблюдения к

категории без проявления исследуемого признака, после PD' – к категории с проявлением признака.

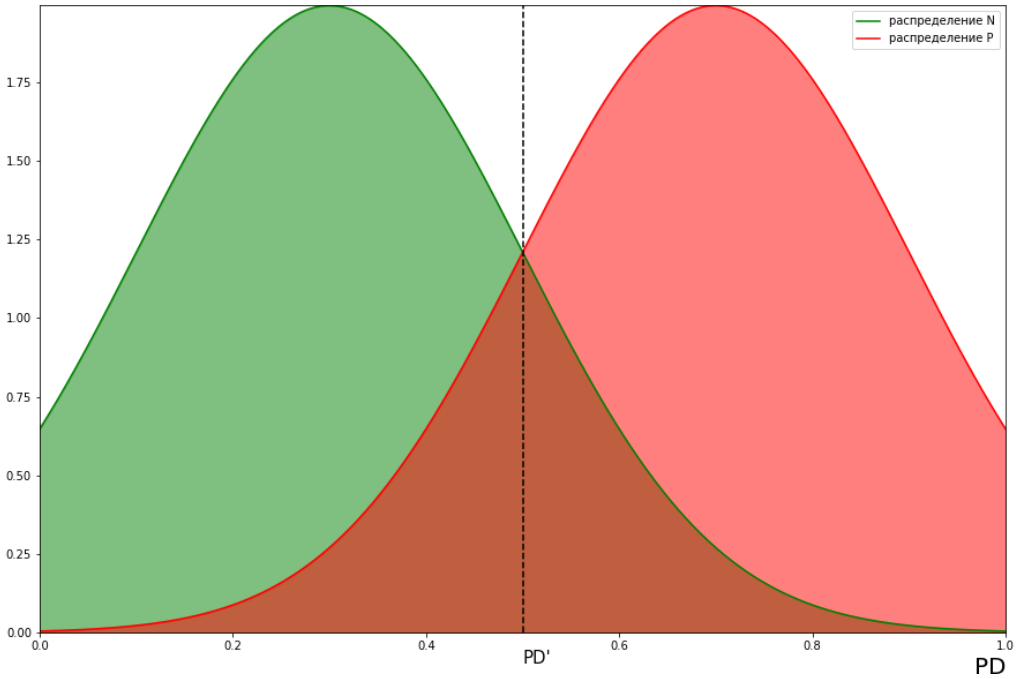

Рисунок 2.1 – Распределение реальных значений P и N

Примечание – Источник: собственная разработка.

Изменяя значение PD' можно влиять на соотношения в таблице 3.1. Увеличение PD' будет приводить к тому, что для большего числа наблюдений будет предсказано отсудившие исследуемого признака (рост N^* и спад P^*). В случае снижения PD' все будет работать наоборот.

Визуализация идей изложенных выше представлена на рисунках Д.1 и Д.2, они представляют строгую и лояльную модели соответственно. Глядя на эти рисунки сделаем некоторые выводы.

Обычно при моделировании используются показатели TP, FP и их относительные аналоги TPR, FPR, потому, сконцентрируемся именно на них.

Начнем из ситуации $PD' = 1$ – совершенно лояльной модели: все наблюдения с проявившимся признаком будут включены в FP, все наблюдения без признака будут включены в TN. Из того следует, что $FPR = 0$, $TPR = 0$.

При постепенном увеличении строгости модели (движении PD' по убыванию) будут возрастать исследуемые показатели. В их возрастании можно выделить два этапа:

1. До пика распределения N – быстро возрастает TP с нарастающим темпом, FP также возрастает с нарастающим темпом но не так быстро как TP;

2. После пика распределения N – рост TP начинает постепенно замедлятся, рост FP ускоряется а FPR постепенно начинает догонять ушедший вперед TPR.

Закачивает этот процесс тем, что TP включает в себя все наблюдения с проявившимся признаком, а FP все наблюдения в которых признака не наблюдалось. Из того следует, что $FPR = 1$, $TPR = 1$.

Теперь, отложим на оси абсцисс FPR а на оси ординат TPR и получим самый популярный способ оценки классифицирующей способности модели – ROC кривую.

Опираясь на рассуждения выше можно сказать, что чем более непохожие распределения по PD у наблюдений с проявившимся признаком и без, тем круче будет ROC кривая и, при том, она проходит через точки (0,0) и (1,1). В подтверждение своих слов приводим рисунок Д.3 – на нем, слева, распределения по PD наблюдений с проявившимся признаком и без представлены в виде нормальных распределений с фиксированной дисперсией, математические ожидания этих распределений постепенно приближаются друг к другу. Справа для каждой из ситуации простроена ROC кривая – видно, как по степени сближения распределений, ROC кривая выпрямляется.

В описанных условиях удобной метрикой, в одно число, классифицирующей способности может выступить AUC ROC кривой – площадь под ROC кривой. Далее, для краткости, описанный показатель, будем называть AUC.

Еще одной связанной метрикой качества модели выступает статистика Колмогорова-Смирнова (KS-статистика). Обычно эта статистика используется для проверки гипотезы о соответствии некоторого наблюденного ряда предполагаемому теоритическому распределению, но в нашем случае сверять мы будем распределение наблюдений с проявлением признака и без. В данном случае статистика определяется по формуле:

$$KS = \max_{PD'} \left| \hat{F}_P(PD') - \hat{F}_N(PD') \right|, \qquad (2.3)$$

где $\hat{F}_P(PD')$ – эмпирическая функция распределения наблюдений с проявлением признака по PD';

$\hat{F}_N(\widehat{PD'})$ – эмпирическая функция распределения наблюдений без проявления признака по PD'.

Графически эту статистику удобно представлять себе в виде рисунка 2.2.

Что из формулы, что из рисунка становиться очевидно – KS статистика показывает на сколько максимум накопленные промежуточным итогом наблюдения одного класса отстают от наблюдений другого класса.

Сразу обозначим следующую закономерность. Из рисунка 2.2 сверху можно сделать вывод:

$$FPR(PD') = 1 - \hat{F}_P(PD'),$$
$$TPR(PD') = 1 - \hat{F}_N(PD').$$

Используя эту закономерность вместе с (2.3), получим:

$$KS = \max_{PD'} |FPR(PD') - TPR(PD')|.$$

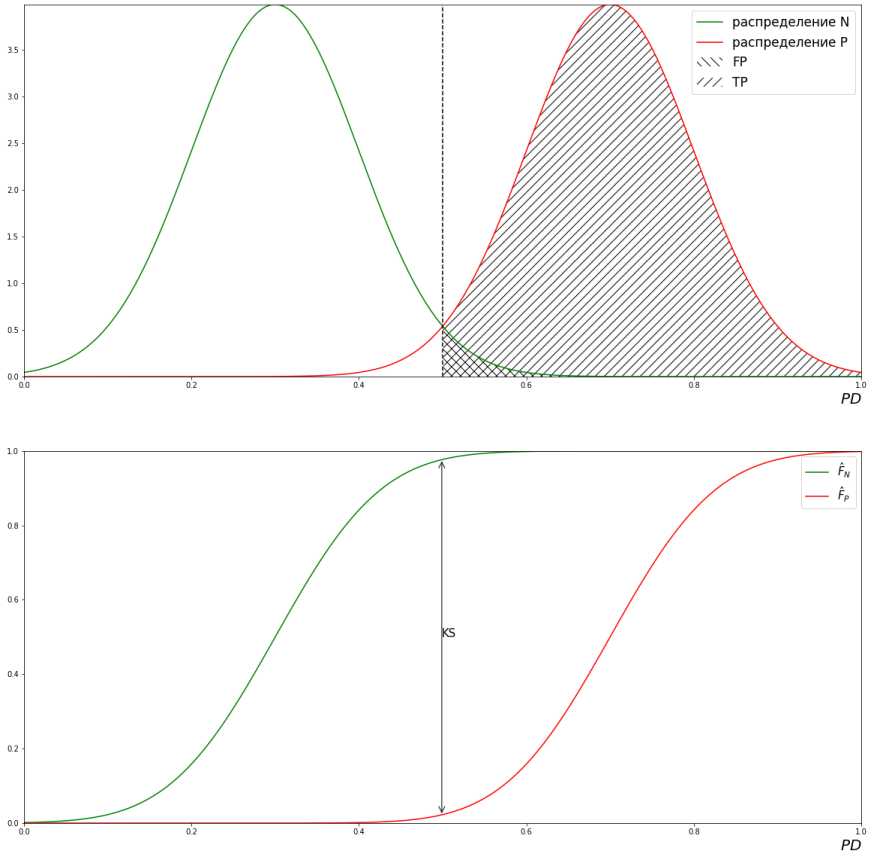

Рисунок 2.2 – Смысл KS статистики
Примечание – Источник: собственная разработка.

Итак, KS это еще и максимальная разница долей истинно положительных и ложно положительных предсказаний модели. От сюда очень важный для нас в дальнейшем вывод – точка PD^*, соответствующая KS статистике, тот вариант точки отсечения который позволяет максимально нарастить долю правильных предсказаний исследуемого признака при минимально допускаемой доле ошибочных предсказаний. Формально такую точку можно записать:

$$PD^* = \text{argmax}_{PD'}\left|\hat{F}_P(PD') - \hat{F}_N(PD')\right| = \\ = arg\text{max}_{PD'}|FPR(PD') - TPR(PD')|. \quad (2.4)$$

Это, кстати, именно тот принцип выбора точки отсечения, который мы упоминали в первом разделе работы когда речь шла построении классификаторов для данных примера. Только там удавалось добиться полной точности классификации, что было следствием $KS = 1$ в формуле (2.3), т.е. находилась точка, в тех обозначениях p' которая оставляла все наблюдения с проявлением исследуемого признака «позади», не достигнув еще ни одного из наблюдений без исследуемого признака.

Важным является вопрос: какую статистику лучше использовать в каком случае? В формуле для *AUC* используются *FPR* и *TPR* для всех точках отсечения, что характеризует эту статистику как описание классифицирующей способности

для всего ряда вероятностей. *KS* же использует всего одну вероятность, и по ней нельзя сделать вывода, есть ли другие точки отсечения с приемлемыми свойствами, тем не менее, она указывает на статистически оптимальную точку отсечения. То, на какую статистику стоит обращать внимание при оценке классифицирующих свойств, сильно зависит от прикладной области, так в кредитном скоринге, при выборе точки отсечения, нередко опираются не столько на статистику, сколько на текущую политику банка, потому важнее иметь много допустимых точек отсечения, от сюда, больше внимания следует уделять статитике *AUC*. За *KS* остается еще одно очень важное преимущество – за ним стоит строгий статистический тест. В рамках выбора готовой модели это малое преимущество, но в рамках оценки классифицирующих свойств отдельных предикторов и, как следствие, их отборе, как будет показано далее, это становиться ключевой возможностью.

Все вышесказанное естественно для готовой модели, а у нас, на данном этапе, задача – отбор показателей в модель. Для оценки классифицирующей способности отдельного предиктора с помощью ROC анализа рассмотрим однофакторную модель вида:

$$\hat{y} = \begin{cases} N, x \in A_N; \\ P, x \in A_P. \end{cases} \quad (2.5)$$

где $\hat{y}$ – предсказанное значение оклика;

$x \in A_N \cup A_P$ – переменная классифицирующая способность которой оценивается;

A_N – множество значений x при котором предсказывается отсутствие исследуемого признака;

A_P – множество значений x при котором предсказывается наличие исследуемого признака.

На данном этапе можно логику разделить на две ветви – исследование числовых и исследование номинативных переменных. Начнем с рассмотрения числовых. Для названного случая формула (2.5) примет вид:

$$\hat{y} = \begin{cases} N, x \leq x' \\ P, x > x'. \end{cases} \quad (2.6)$$

где x' – точка отсечения по переменной x.

В отношении кредитного скоринга получается модель со следующим принципом принятия решений: для всех наблюдений у которых наблюдаемое значение меньше либо равно некоторого порога x' принимаем решение о выдаче кредита, в противном случае, понимается решение об отказе.

Заметим, что рассуждения предыдущего абзаца очевидно справедливы для случая, когда с ростом исследуемого показателя ожидается увеличение проявления исхода с дефолтом (P). В действительности же, даже чаще, встречается обратная ситуация – с ростом исследуемой переменной ожидается

падение проявления исходов с дефолтом. Например, с ростом уровня дохода клиента физического лица, справедливо ожидать от него большей стабильности платежей и, как следствие, меньшую вероятность выхода в дефолт.

В сущности, приведенное замечание не меняет основных идей анализа – если на рисунках Д.1, Д.2 поменять местами распределения P и N, то диаметрально поменяется порядок изменяя *FPT, TPR*, описанный выше. Что приведет к тому, что ROC кривая выгнется в обратную сторону. Такая ситуация представлена на рисунке Д.4.

Очевидно, что, в таком случае, нельзя сказать, что модель не имеет классифицирующей мощности, хотя AUC, указывает на то, что классифицирующая мощность модели ниже чем в случае случайного угадывания (нижняя часть рисунка Д.3).

Если бы такая модель использовалась для предсказаний, то логично было бы поменять знаки в выражении (2.6). Но, в нашем случае, мы лишь оцениваем классифицирующую способность предиктора с использованием AUC. Потому введем следующее окончательное определение AUC:

$$AUC = \begin{cases} AUC', AUC' \geq 0.5; \\ 1 - AUC', AUC' < 0.5, \end{cases}$$

где AUC' – эмпирическая функция распределения наблюдений с проявлением признака по PD';

AUC – эмпирическая функция распределения наблюдений без проявления признака по PD'.

Так же, заметим, что ранее речь шла о распределении наблюдений с различным состоянием признака по *PD*, а сейчас речь идет о распределении вдоль некоторой численной переменной *x*. В сущности, снова нет большой разницы – наблюденный *x* можно нормировать и, по смыслу, будет получен тот-же *PD*.

Более сложным является подобный анализ для номинативной переменной. Множества A_N и A_P становятся дискретными конечными множествами, в объединение которых включены все уровни исследуемой переменной.

Обозначим и пронумеруем все уровни исследуемой переменной как $t_i, i = \overline{1, m}$. Начинаем «движение» от наиболее лояльной ситуации, когда при любом уровне исследуемого предиктора в модели (2.5) принимается решение о том, что $\hat{y} = N$, т.е. $A_N = \{t_1, t_2, \dots, t_m\}$, $A_P = \emptyset$. Далее в множество A_P в обратном порядке вводятся t_i, и выводятся из A_N – модель становиться строже. Так продолжается до тех пор пока мы не приходим к наиболее строгой ситуации, при которой, для любого уровня исследуемого предиктора принимается решение о том, что $\hat{y} = P$, т.е. $A_N = \emptyset$, $A_P = \{t_1, t_2, \dots, t_m\}$. В более понятной форме эта логика описана в таблице Е.1.

Используя описанный механизм можно, строить ROC кривую и вычислять показатель *AUC*. На рисунке Е.1 нанесена ROC кривая в общем виде и различными заполнениями (которые понадобиться в дальнейшем

повествовании) обозначена площадь под ней. ROC кривая состоит из ряда линейных участков, каждый из которых знаменует переход к новому уровню предиктора и образует с осью абсцисс трапецию площадь которой обозначим S_q.

Может показаться, что просто суммируя S_q можно получать AUC для номинативного предиктора, однако, при использовании этого метода на практике, быстро выявляется нюанс описанный в следующем утверждении.

Утверждение 1. При изменении порядка включения предикторов в группу P вид ROC кривой может меняться и, как следствие, может меняться показатель *AUC*.

Это утверждение наталкивает на вопрос – какую последовательность включения уровней использоваться для вычисления статистики *AUC*? Для того, чтобы ответить на этот вопрос введем следующее утверждение.

Утверждение 2. Если строить ROC кривую по логике постепенного уменьшения лояльности модели (так как это записано в таблице Е.1), то при включении уровней в порядке убывания частот проявления исследуемого признака в отклике *AUC* будет максимален.

В виду и так заметной перегруженности этого подраздела теоритическими сведениями не будем приводить строгого доказательства этого утверждения – его можно получить используя сведения школьного курса алгебры и геометрии. Оговоримся лишь, что доказательство строиться на том, что зеленая площадь рисунка Е.1 не зависит от последовательности включения *i-го* уровня с меньшей частотой проявления исследуемого признака и *j-го* уровня с большей частотой проявления. Синяя тоже, но по другим причинам. Любая трапеция красной площади может быть представлена формулой:

$$S_s = \frac{n_s^N}{2N'P'}\left(2\left[\sum_{k=s+1}^{j-1} n_k^p + n_j^P + \sum_{k=j+1}^{m} n_k^p\right] + n_s^P\right), s = \overline{\imath+1, j}.$$

В случае прямого включения уровней. А в случае, если поменять порядок включения *i-го* и *j-го* уровней между собой:

$$S'_s = \frac{n_s^N}{2N'P'}\left(2\left[\sum_{k=s+1}^{j-1} n_k^p + n_i^P + \sum_{k=j+1}^{m} n_k^p\right] + n_s^P\right), s = \overline{\imath+1, j}.$$

Очевидно, что два последних выражения отличаются на второе слагаемое в квадратных скобках. Учитывая вышесказанное, заметим, что $n_i^P < n_j^P$, от сюда и $S_s > S'_s$, что ведет к истинности второго утверждения.

В целом, результат рассуждения такой: для получения показателя классифицирующей способности *AUC* для номинативного предиктора, следует включать урони в порядке возрастания проявления исследуемого признака – в таком случае *AUC* будет максимальный. Нужно это, конечно, не для того, чтобы

завысить показатель *AUC*, а чтобы из множества возможных оценок выбрать единственную.

У *AUC* в данном случае имеется один большой минус – нет критерия при котором можно считать, что показатель связан с объясняемой переменной или придерживаться обратного утверждения. Тут на помощь приходит введенная выше связанная статистика KS – она распределена в соответствии с распределением Колмогорова-Смирнова, от того возможность проведения статистического тестирования.

Нас, в данном случае, очевидно, интересует двухвыборочная вариация теста – сравнение выборок, полученных для наблюдений с проявлением исследуемого признака и без него. Такой тест имеет следующую постановку: пусть X и Y две независимые случайные переменные, чьи функции распределения соответственно F и G неизвестны. Выдвигается нулевая гипотеза [13]:

$$H_0: F(x) = G(x), \forall x \in R^d.$$

Против альтернативной гипотезы:

$$H_1: F(x) \neq G(x), \forall x \in R^d.$$

Представленная выше KS статистика распределена в соответствии с распределением Колмогорова-Смирнова с числом степеней свободы вычисляемым по формуле:

$$\frac{nm}{n+m},$$

где n – число наблюдений в выборке X;

m – число наблюдений в выборке Y.

Нулевая гипотеза принимается в том случае, если реальная статистика меньше теоритической вычисленной с заданной доверительной вероятностью. Или можно посчитать p-значение, как значение функции распределения Колмогорова-Смирнова с указанным числом степеней свободы в точке KS.

Все описанные принципы положены на практику классом python «classification_power_predictor», со всеми сопутствующими методами расположившемся в отдельном репозитории[2] в сети интернет. Он позволяет произвести подсчет *AUC*, *KS* и p-значений двувыборочного *KS* теста, для всех переменных переданного «pandas.DataFrame» и столбца отклика в виде «pandas.Series».

В конструктор этому классу следует первым аргументом предать набор предикторов и столбец отклик, например так:

```
import classification_power_predictor as cpp
```

[2] https://github.com/Dranikf/classification_power_predictor

```
my_cpp = cpp(
    data.drop('Y', axis = 1),
    data['Y'].replace({0:"Нет дефолта", 1:"Дефолт"})
),
```

где data – экземпляр класса «pandas.DataFrame» с структурой данных соответствующей той, что мы организовывали выше.

Заметим некоторую тонкость последней записи – столбец «Y» изначально идет как набор нулей и единиц, класс устроен так, чтобы выводить урони так как они названы в столбце отклике, потому используя метод «replace» мы сложно читаемые 0 и 1 превращаем в «Нет дефолта» и «Дефолт» соответственно.

После получения экземпляра класса «classification_power_predictor» обязательно нужно вызвать его метод «update_predictors». После того в объексте будет проинициализировано поле «result_DF», которое представляет собой «pandas.DataFrame» структуры, подобной той, что представлена в таблице Ж.1. В таблице Ж.1 первое поле указывает название предиктора, поля 2-4 – статистики вокруг которых строилось рассуждение выше: *AUC, KS* и *p*-значение двувыборочного теста Колмогорова-Смирнова. Поля 5-7 дают информацию о пропусках: «Пропуски» указывает количество пропусков в каждом столбце, последние два поля описываю структуру распределения пропусков по уровням отклика.

Заметим, что в целях размещения в работе, таблица была сильно урезана, так в оригинальном варианте такой набор колонок формируется для каждого уровня отклика. В данном случае, информации потеряно не было – в случае бинарной классификации *AUC* и *KS* для обоих классов будут одинаковые.

В поле «result_DF» располагается также информация о показателе *Gini*, который составляет альтернативу *AUC* и вычисляется по формуле:

$$Gini = (AUC - 0{,}5)/2.$$

Еще, по умолчанию, можно получить боле подробную информацию о наполненности данных и структуре пропусков.

Еще одно важное замечание – при вычислении статистик классифицирующей способности для номинативных переменных пропуск воспринимается как отдельный уровень. Это сделано, для того, чтобы не терять данных. Предполагается, что стремление кредитополучателя скрыть о данные о себе может быть признаком того, что эти данные могут занизить его шанс на получение кредита.

При более подробном рассмотрении таблицы Ж.1 можно найти ряд предикторов, для которых двувыборочный тест Колмогорова-Смирнова ведет к принятию нулевой гипотезы об отсутствии различий распределений наблюдений с дефолтом и без него, при доверительной вероятности в 0,01. В список таких показателей вошли: «Автомобиль год выпуска3», «Количество детей», «Отношение к банку», «Работа последнее место стаж лет», «Работа уровень

дохода BYR», «Количество действующих договоров обеспечения», «Число авто» и «Есть авто».

Сразу было произведено удаление этих столбцов. Далее проводился более тонкий анализ для каждого из предикторов по отдельности. По результатам было принято решение произвести удаление столбца «Автомобиль год выпуска», так как он имел меньший показатель *AUC* по сравнению с базовыми для него столбцами «Автомобиль год выпуска 1» и «Автомобиль год выпуска 2». Аналогично «Ежедневный платеж» как отношение «Сумма договора» к «Срок депозита в днях» уступает базовому показателю по значению *AUC*.

Похожая ситуация с показателями «Столица», «Областной центр», «Адрес проживания – Населенный пункт». Несмотря на то, что они были созданы в отрыве от показателя «Тип населенного пункта», очевидно, что они неразрывно с ним связаны и, вместе с тем, заметно уступают ему по классифицирующей способности, потому были удалены.

Следующим шагом, после удаления показателей со слабой описательной способностью, будет борьба с пропусками, перечислим возможные пути решения и проблемы связанные с каждым из них:

- Удаление строк содержащих пропуски по каждому показателю. Если некоторый показатель слабо заполнен, то для его включения в модель понадобиться удалить много сток, и будет потеряна информация указанная по удаленным записям в других столбцах;
- Удаление столбцов с большим числом пропусков. Так же ведет к недостаточности данных, но в другом смысле – теряются «ракурсы» с которых мы можем смотреть на исследуемую проблему;
- Заполнение пропусков данных альтернативными значениями – ведет к искажению реальных закономерностях лежащих в данных может привести к улучшению модели «на бумаге», но в реальных условиях толку от нее будет мало;
- Рассмотрение пропусков в номинативных показателях как отдельных уровней. Так же может вести к искажениям в данных, но лучше применять такой подход чем выбрасывать показатели.

В общем нет универсального рецепта – всегда приходиться принимать решения опираясь на конкретные данные и, даже, возвращаться к этому этапу в процессе построения модели.

Логически получилось связать показатели «Воинская служба» и «Пол». Очевидно, что у большинства женщин в графе «Воинская служба» должно быть проставлено «невоеннообязанный» и если соответствующие пропуски заполнить этим значением в подавляющем большинстве случае это будет правильно.

Для любой числовой переменной, превращение ее в номинативную переменную и, как результат, сохранение наблюдений с пропуском в этом столбце можно было осуществить двумя путями.

Можно превратить в бинарные переменные с уровнями «нет данных»/ «есть данные».

Второй подход несколько сложнее – для каждой переменной подсчитаны статистики *KS*. Выше описывалось, как выбрать точку отсечения для готовой модели с использованием *KS* статистики по формуле (2.4). Можно попробовать использовать такой подход для того, чтобы определиться с точкой разделения численной переменной на два уровня номинативной переменной, и отдельно будет использован уровень «нет данных», таким образом получается номинативный показатель с тремя уровнями.

Еще один способ обойти проблемы связанные с пропусками в числовых предикторах – записать вместо пропусков число ноль. В некоторых случаях это может быть оправдано, например, в случае показателя «Работа последнее место стаж лет» некоторые кредитополучатели при заполнении анкеты могли решить, что в случае отсутствия стажа (0 лет) следует в этом поле ничего не указывать.

Применим все три операции к каждой переменной численного типа из оставшихся на данный момент в выборке. И потом, опираясь на здравый смысл и подсчитанные статистики классифицирующей способности, примем решения о том в какой форме та или иная переменная должна участвовать в модели и должна ли участвовать вообще. В таблице Ж.2 результаты применения класса «classification_power_predictor» к таким переменным. В ней каждая численная переменная исходного набора переменных содержит еще 3 дочерние переменные. Первая группа с припиской «(бинарная)» представляет разделение на «нет данных»/«есть данные», вторая группа с припиской «(пропуск => 0)» предполагает замену пропусков нулями и последняя отмечена как «(разбивка по KS)» предполагает, что для разделения использовалась формула (2.4).

Показатель «Автомобиль год выпуска1» по классифицирующей способности сам по себе достаточно хорош, но имеет очень много пропусков. Было принято решение использовать его вариацию с разделением по *KS* так как она в сравнении с остальными производными показателями сохраняет высокий *AUC*. Аналогичное рассуждение можно приложить к показателям: «Количество запросов в КБ за последние 30 дней», «Максимальный срок, на который заключался договор, в годах», «Общее количество запросов в КБ» и «Срок кредита в днях».

Показатель «количество действующих кредитных договоров» даже улучшил свой *AUC* при замене пропусков нулями. Такое преобразование имеет за собой логическое основание – при отсутствии кредитных договоров многим потенциальным кредитополучателям может показаться логичным не заполнять соответствующее поле, потому оставим именно этот вариант показателя. Аналогичное рассуждение можно приложить к показателям: «Максимальное количество дней просрочки» и «Сумма кредитных лимитов (пропуск => 0)».

Отдельного обсуждения заслуживает показатель «Сумма договора». Названный показатель несколько прибавляет в значении *AUC* в случае замены пропусков нулями, но вообще, сомнительно качество записей в которых не указана даже сумма договора – сами информационные системы банка должны сохранить это значение. В результате для этого показателя было принято

решение сохранить исходную форму несмотря на потерю около 10% записей от исходного числа.

Все оставшиеся показатели было решено не использовать в модели – у них недопустимо много пропусков и показатели *AUC* очень слабые.

Последним шагом станет возвращение сохраненным производным показателям имен родительских для них показателей.

В этой главе рассмотрен полный цикл подготовки данных к моделированию: загрузка данных, изучение их структуры и приведение типов данных, исключение невозможных значений и выбросов, анализ классифицирующей способности и отбор предикторов, исключение пропусков.

Изучение структуры данных проводилось с использованием подготовленного инструмента – функции языка программирования python3 «get_data_frame_settings» который позволяет получать таблицы вида таблиц приложения Г. На основе такого анализа были выдвинуты предположения, которые легли в основу создания новых показателей. Развитие некоторых показателей показалось неперспективным направлением, потому они были удалены. Даты получили численный тип данных, как число дней от базовой даты, номинативные показатели с очень большим числом уровней либо были укрупнены либо удалены в связи со сложностью возможных алгоритмов укрупнения и логической неперспективности для решения поставленной задачи. В связи с оторванностью от остальных записей исследуемого набора данных были исключены записи содержащие информацию о кредитах типа овердрафт вместе с показателем позволяющим произвести соответствующее разделение.

Касательно вопроса недопустимых значений можно обратиться к таблице Г.2, а именно к ее строкам использующимся для описания показателей «Работа последнее место стаж лет», «Работа уровень дохода BYN», «Сумма договора» и «Ежедневный платеж», они по смыслу, очевидно, не могут содержать отрицательные значения, однако по проведенному исследованию получается, что есть записи с отрицательными числами. Записи с невозможными значениями были заменены на пропуски.

Так же наше внимание привлекли показатели с подозрительно большими размахами: «Количество фактов просрочки по основному долгу», «Максимальное количество дней просрочки», «Общее количество запросов в КБ», «Сумма кредитных лимитов», «Сумма договора» и «Ежедневный платеж». В подтверждение этой идеи мы приводим таблицу 2.2 – значение 75-ой персетиля и максимальное нередко различаются более чем в 100 раз. Выбросом было принято считать значение не попадающее в интервал (2.2). Выбросы были также заполнены пустыми значениями.

Второй подраздел данной главы был посвящен преимущественно анализу классифицирующей способности предикторов. Были описаны показатели *AUC* и *KS*. Главная цель этих показателей выбор готового классификатора, но мы предложили для каждого показателя рассмотреть однофакторную модель вида (2.5) и рассчитанные для нее показатели воспринимать как оценки классифицирующей способности соответствующих предикторов.

Для того, чтобы в 3 строки кода проводить описанный анализ был создан класс языка программирования python3 «classification_power_predictor». Результаты его работы выражаются в таблицах приложения Ж. В результате были исключены все показатели для которых двухвыборочный тест Колмогорова-Смирнова приводил к принятию нулевой гипотезы о том, что распределения наблюдений с дефолтом и без по исследуемому показателю не различаются. Из всех связанных между собой показателей были выбраны те, которые имеют лучшие оценки классифицирующей способности, остальные были удалены.

В контексте избавления от пропусков в основном использовался метод приведения численных показателей к номинативной форме, и для всех номинативных переменный пропущенное значение воспринималось как новый уровень переменной. Особо интересен подход к разбиению численной переменной с использованием точки соответствующей *KS* статистике (формула (2.4)).

Таким образом, мы пришли к структуре данных описанной таблицей Г.4, а классифицирующие способности отдельных предикторов представлены в таблице Ж.3.

К сожалению невозможно гарантировать, что во всех моделях будет использован именно точь в точь этот набор данных – нередко приходиться подгонять данные в зависимости от ситуации. Но по умолчанию используется именно этот набор данных, все преобразования будут кратко описаны.

3 Построение и валидация модели

3.1 Программное описание модели и алгоритма обучения

3.1.1 Основные элементы модели в pytorch

В этой главе мы сконцентрируемся на прикладных вопросах создания моделей искусственных нейронных сетей. На сегодняшний день нет готовых пакетов которые позволяют полноценно формировать модели нейронных сетей, вызвано это тем, что ходы которые делают инженеры соответствующей квалификации очень разнообразны и зачастую, даже, уникальны. Потому, удобнее всего использовать языки программирования для построения таких формул. Однако нам не придется все писать с нуля, разработаны и распространяются как open source проекты, ряд отличных библиотек языка программирования python3. Мы будем использовать разработанный кампанией facebook пакет-расширение pytorch [10]. Выбор пал на него благодаря его относительной простоте, по сравнению с разработанным в google пакетом tensor flow, и гибкости, по сравнению с надстройкой над tensor flow под названием keras.

Заметим, что эта глава будет содержать куда больше технических деталей, так как в ней описывается центральный для этой работы алгоритм.

Весь и самый актуальный код для построения модели распложен в указанном выше репозитории в папке «model». Ключевые его элементы будут представлены в приложении И и подробно описаны в этом разделе.

Любая искусственная сеть, формируемая в pytorch, обязательно должна содержать следующие элементы:

- Класс, наследник класса «torch.nn.Module» описывающий в общем виде нейронную сеть требуемой идентификационной формы и, соответственно, экземпляр этого класса;
- Целевая функция, в виде экземпляра одного из классов сформированных в модуле «torch.nn». Полный список готовых целевых функций представлен на сайте документации pytorch [14];
- Так называемый оптимизатор – алгоритм оценки параметров модели, в виде экземпляра одного из классов модуля «torch.optim». Полный список возможных оптимизаторов представлен на сайте документации pytorch [15].

В первой главе работы, касаемо идентификационной формы модели мы определились лишь с тем, что выходной слой представляет собой единственный сигмоидальный нейрон, а в срытых слоях содержатся ReLU нейроны. Неопределенными остаются число скрытых слоев нейронной сети и количесво нейронов в них входящее. Эти факты прямо повлияли на описание класса нейронной сети.

Названный класс представлен на листинге И.1. Подробно разберем этот класс. Метод «__init__» в python – зарезервированный метод под конструктор. Конструктором называются методы вызываемые при создании экземпляра класса. В нашем случае конструктор принимает список «neur_counts» в качестве

аргумента. Это должен быть список из целых чисел каждое число представляет собой число нейронов в соответствующем слое начиная с входного слоя. В нем не следует указывать выходной слой, это по умолчанию один нейрон. Так для создания экземпляра описывающего сеть с тремя входами и двумя скрытыми слоями, по 2 и 3 нейронами соответственно надо будет записать:

ResultNet([3,2,3]).

Такой подход обеспечит нам возможность создания моделей различных идентификационных форм «на лету» – можно будет в циклах перебирать различные идентификационные формы не прибегая к изменению кода класса.

Библиотека pytorch использует несколько другой тип формального представления сети. Нейроном называется то, что мы в торической главе называли сумматорной функцией, а потом к ним применяются такие преобразования, которые мы называли активационной функцией. В результате, никакой разницы, но сначала может создавать некоторый диссонанс теории и практики. Так конструктор создает словарь «layers» в котором последовательно указываются слои, как экземпляры класса «nn.Linear», реализующие линейную комбинацию (1.11). Для создания нужно указать число выходных активаций предыдущего слоя и число выходных активаций создаваемого слоя (число нейронов в этом слое). Созданный нами класс все это проделывает автоматически. Кроме того, мы заполнили веса и свободные члены каждого слоя случайными значениями, так формируется точка с которой алгоритм будет начинать процесс оптимизации. Лучшим решением для такой ситуации было бы указать примерное решение, но у нас нет идей как оно должно выглядеть, потому будет заполнение случайными весами.

Класс нейронной сети в pytorch должен содержать в себе метод «forward». Основная цель этого метода применять к матрице X входных данных нейронную сеть в текущем ее состоянии. Тут последовательно, в цикле перебираются слои созданные конструктором (все кроме последнего), на каждой итерации X подставляется в советующий слой, а результат в функцию «torch.nn.functional.relu», которая и представляем собой ReLU преобразование. Полученная матрица заменит X вышедший из предыдущего слоя. Последним действием применяется «torch.sigmoid», который реализует активацию выходного нейрона, полученное значение становиться возвращаемым значением метода «forward».

Что касается целевой функции, мы использовали бинарную кросс-энтропию [17]. Эта функция определяется следующим образом:

$$H(y,p) = \frac{1}{N}\sum_{i=1}^{n} y_i \log(p_i) + (1-y_i)\log(1-p_i),$$

где y_i – реально наблюденное значение признака, $y_i \in \{0,1\}, i = \overline{1,n}$ для i-*го* наблюдения;

p_i – предсказанная вероятность одного из уровней, $p_i \in [0,1]$, $i = \overline{1,n}$ для i-*го* наблюдения.

Эта функция рекомендована документацией pytorch к использованию в случае задач бинарной классификации [14]. Она реализуется как экземпляр класса «torch.nn.BCELoss». При создании не требуется никаких аргументов.

Оптимизаторы в pytoch наследуются от базового класса «torch.optim.Optimizer». Любому наследнику следует передавать первым аргументом параметры нейронной сети, обычно они идут в формате «torch.Tensor» и должны быть извлечены из экземпляра нейронной сети, то есть в нашем случае это должно выглядеть так:

```
model = ResultNet(layers_list)
optimizer = Some_optimizer_class(model.parameters()),
```

где layers_list – предварительно объявленный список слоев;

Some_optimizer_class – выбранный наследник.

Работает это следующим образом: model.parameters() возвращает параметры нейронной сети, а optimizer «запоминает» их и, по вызову его метода «step()», изменяет их в зависимости от ситуации. Более подробно этот механизм будет раскрыт далее, когда мы непосредственно коснемся вопроса обучения нейронной сети.

Конкретные алгоритмы оптимизации реализованы в наследниках. Нигде не удалось найти рекомендаций или указаний, какой вид алгоритма лучше всего использовать в нашем случае, мы всегда используем «torch.optim.Adam», потому как он нередко приводиться в примерах к задачам классификации. Важным параметром для оптимизатора является learning rate – это число на которое умножается градиент целевой функции, таким образом можно регулировать величину шага нелинейной оптимизации. Во всех готовых оптимизаторах и, в частности, в оптимизаторе Adam он указывается через именованный аргумент «lr» конструктора.

3.1.2 Реализация алгоритма обучения

В самом базовом случае для обучения модели нужны следующие шаги:

- Обнулить градиент оптимизатора с прошлой итерации используя метод «torch.optim.Optimizer.zero_grad»;
- Произвести прямое распространение активации используя описанный выше метод «forward» экземпляра модели. Нет необходимости вызвать метод напрямую, для «torch.nn.Module» произведено переопределение оператора «()», так что он вызывает «forward» автоматически для переданных аргументов. Соответственно, все наследники, если в них, конечно, оператор не переопределен повторно будут действовать также;

- Предсказанные результаты с реальными откликами передаются в экземпляр целевой функции, для которого также переопределен «()». Будет получена ошибка – объект класса «torch.tensor»;
- У полученной ошибки вызывают метод «baskward», который представляет собой ничто иное как реализацию обратного распространения ошибки, алгоритма о котором мы говорили в подразделе 1.4. В результате оптимизатор получает информацию о градиенте;
- В последнюю очередь вызывают метод «step» оптимизатора, тут то и происходит каждое смещение весов.

Описанные шаги повторяются много раз, так добиваются оптимизации весов. Данный подход самый базовый и, в большинстве случаев, его не достаточно. Оказалось недостаточно и в нашем, потому для обучения модели был создан класс «model_traner», представленный на листинге И.2. В названном классе использованы различные техники, позволяющие проводить обучение более эффективно.

Конструктор класса обязательно ожидает модель, оптимизатор и целевую функцию, кроме того имеется именованный аргумент «lr_scheduler» сущность которого будет раскрыта ниже, когда речь коснется соответствующей техники.

Обыденной практикой машинного обучения считается разбиение выборки на тренировочную и тестовую. Дело в том, что нейронная сеть – очень мощный метод аппроксимации, нередко получается так, что модель не улавливает закономерность, а просто запоминает выборку на которой училась, это ведет к тому, что она необоснованно ошибается на данных которых не видела, такую ситуацию в машинном обучении называют переобучением модели [18 с. 127]. Разбиение на тренировочную и тестовую выборки решает эту проблему – выгодно отслеживать целевую функцию и проводить валидацию модели на выборке которая не участвовала в обучении, несмотря на то, что для оценки коэффициентов будет задействован не весь массив данных. Мы рассказываем об этом именно тут потому как дальнейшее описание класса обучающего модель неразрывно связано с этой практикой.

Особого внимания заслуживает метод «fit» – центральный этого класса. В названном методе, по сути, и работают шаги обучения модели. Этот метод обязательно принимает аргументы «train_loader» и «test_loader». В терминах «tytorch» это загрузчики данных, соответственно получаем загрузчик обучающих и тестовых данных.

Загрузчики позволяют «элегантно» реализовать еще одну важную для данной работы технику – обучения модели по эпохам. При обучении модели по эпохам набор данных, выделенный для обучения, разбивают на равные части – так называемые батчи. Затем батчи перебирают и для каждого из них вычисляется градиент целевой функции и производится соответствующее смещение весов. Эпохой, применительно к алгоритмам обучения нейронных сетей, называют одну итерацию прохода по всем таким частям [19 с. 202]. Делают это для того, чтобы при вычислении градиента упрощать целевую функцию. А упрощение целевой функции, в свою очередь, приводит к тому, что

в задаче оптимизации становиться меньше локальных минимумов и алгоритм вероятнее сходится к глобальному минимуму или хотя-бы выбирает лучший локальный минимум.

Касательно батчей, мы провели отдельное небольшое исследование этого механизма. Его можно найти в ноутбуке «proc/batch_size_research/barch_size_research.ipynb» основного репозитория данной работы. Тут представим краткие результаты – мы создали случайный двумерный и набор данных с двумя классами, он представлен точками на рисунке К.1. Специально пытались создать набор данных так, чтобы была группа точек «ломающая» основную статистическую закономерность (она в левом верхнем углу). Таким образом мы создали два здравых решения: левый рисунок представляет первое, правый – второе. Решения, для просторы, линейные и представляют собой двухфакторные логистические регрессии. Тепловой картой обозначены предсказания соответствующих моделей. А в заголовках к отдельным графикам представлены значение целевой функции бинарной кросс энтропии для данных предсказаний. Интересно, что мы рассчитывали, что закономерность заложенная в правый график будет побочной, но получилось так, что она стала основной (у нее ошибка ниже). Но так даже лучше, пример еще показателен в том плане, что нередко наиболее оптимальное решение оказывается неожиданным и, даже, нелогичным.

Как известно, двухфакторная логистическая регрессия, в классическом виде, требует три параметра: два при переменных и свободный член. Это не очень то удобно для визуализации целевой функции по параметрам, потому мы два параметра при переменных объединили в один, со смыслом поворота дискриминирующей прямой. Формула выражения под логитом приняла вид:

$$\cos(\alpha)\, x_1 + \sin(\alpha)\, x_2 + \beta,$$

где α – поворот дискриминирующей кривой в радианной мере;

β – смещение;

x_i – i-я переменная $i = \{1,2\}$.

Такое изменение немного отрывает нас от реального положения дел (модели так никогда не параметризуют), но, как увидите далее, для целей нашего примера такой поход хорош. Это нужно было для того чтобы нанести рисунок 3.1.

На рисунке 3.1 линии уровня целевой функции в зависимости от введенных параметров. Слева рисунок на полных данных – отчетливо выделяются два локальных минимума, но глобальный только правый. И если мы начнем оптимизацию с неудачных параметров, то мы рискуем прийти в левый. И тут это решается просто визуализацией, но, напомним, что в нашей задаче будет не менее 113 параметров. Не возникает сомнений, что подобная визуализация невозможна. Потому и пользуются всевозможными уловками для того, чтобы обходить локальные минимумы.

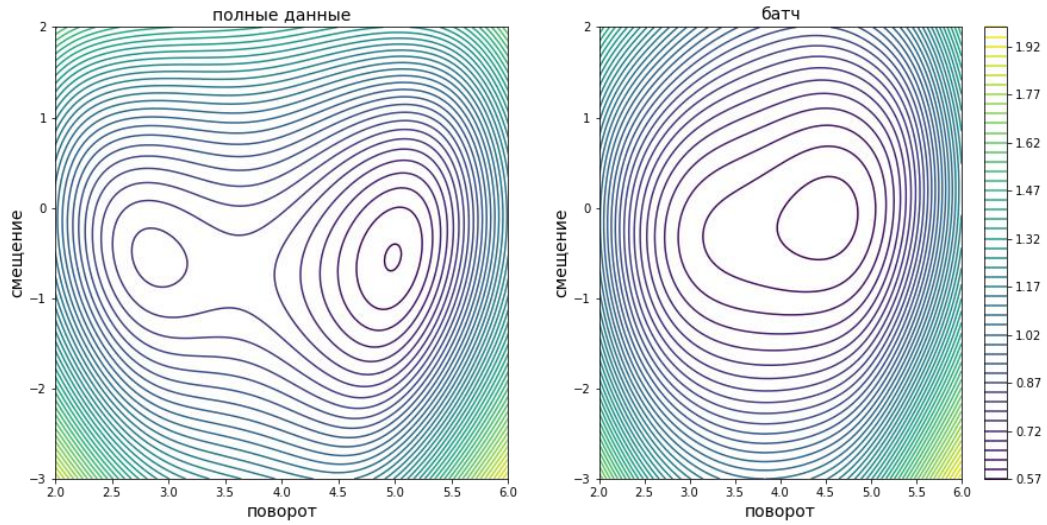

Рисунок 3.1 – Целевая функция по параметрам
Примечание – Источник: собственная разработка.

Теперь, случайным образом, выделим из общего набора данных батч. Он представлен на рисунке К.2 или, еще ценнее, для нас рисунок 3.1 справа. По этим рисункам можно увидеть, что второе решение попросту выродилось при использовании батча. Можно, конечно, заметить, что могли просто выпасть так наблюдения в батч, но в реальных условиях в алгоритме за одну эпоху отрабатывает множество батчей и, похоже, можно доказать, что чаще всего в них будет сохраняться наиболее заметная статистическая закономерность, что приведет к тому что алгоритм будет сходиться к более заметным минимумам.

Вернемся к загрузчику – реализации обучения по эпохам в pytorch. Загрузчик данных должен представлять собой экземпляр класса «torch.utils.data.DataLoader». Для его создания, в конструктор надо передать экземпляр-наследник класса «torch.utils.data.DataSet», с переопределенными методами «__len__» и «__getitem__». Первый, из названных методов, должен возвращать длину используемого набора данных, второй – пердоставлять доступ к элементам по переданному индексу. Таким образом в «pytorch» достигают независимости от формата входных данных, разработчику модели предоставлена возможность описать как ведет себя его набор данных. Наша реализация наследника «torch.utils.data.DataSet» представлена на листинге И.3.

Не синтаксически, но фактически обязательно, знать про именованный аргумент «batch_size» загрузчика данных. Названный аргумент позволяет регулировать размер батча. Размер батча, пожалуй, самый важный параметр касающийся данных на этапе обучения сети – малый батч позволит обойти локальные минимумы, вместе с тем, целевая функция теряет точность, потому и оценки параметров на больших батчах будут точнее.

Возвращаясь к алгоритму обучения сети, в методе «fit» мы запускаем цикл по загрузчику тренировочных данных. Этот цикл обеспечивает обход батчей, а внутри уже самый обычный алгоритм обратного распространения ошибки.

Заметим, что для тестовых данных можно было обойтись и без использования загрузчика, но, для общности, они в алгоритм обучения попадают также через него.

Без внимания остались еще поля «train_loss» и «test_loss» класса «model_trainer». В них, по окончании каждой эпохи, сохраняются значения целевой функции на полных тренировочных и тестовых данных соответственно. Эта информация потом используется для построения кривых обучения – графиков на которых можно отследить как изменяется значение целевой функции с новыми эпохами. Эта визуализация является популярным способом диагностики состояния модели.

В этом подразделе работы были описаны основы работы с пакетом pytorch и представлены созданные программные методы для решения поставленной задачи. Еще остаются нераскрытые детали, но приведенных сведений достаточно для того чтобы начать рассматривать конкретные примеры, а прочие механизмы мы подключим по мере необходимости.

3.2 Обучение и валидация модели

3.1.1 Подбор параметров обучения

Перед тем, как использовать созданные вычислительные процедуры, придется провести еще три относительно простых преобразования данных. Они вынесены за рамки второй главы работы, так как действуют уже без использования какого-либо смысла вложенного в показатели, а рассматриваются лишь как наборы чисел.

Во первых для номинативных показателей следует провести one hot encoding (OHE) – процедуру при которой каждый номинативный показатель распадается на столько показателей, сколько он имеет уровней, каждый произведенный таким образом показатель принимает значение «1» в том случае если базовый для него показатель принимал соответствующий уровень и «0» в любом другом случае. Смысл легко понять взглянув на рисунок 3.2.

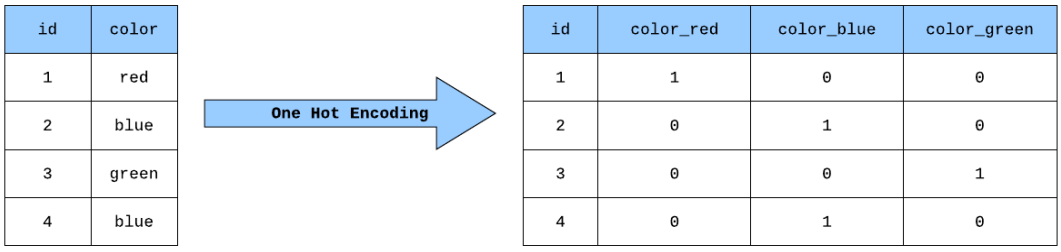

Рисунок 3.2 – Пример процедуры OHE
Примечание – Источник: [20].

Имеется таблица содержащая колонки «id» и «color». Столбец «color» принимает три уровня: «red», «blue» и «green». После проведения названного

преобразования к колонке «color» он образует три столбца, по одному для каждого из уровней.

Для проведения такой процедуры можно использовать класс «OneHotEncoder» модуля «sklearn.preprocessing».

Вторя процедура, которую мы проводили над данными – нормализация данных. Это не обязательная процедура для моделирования, но ее нередко рекомендуют проводить для увеличения эффективности процесса обучения[21 с. 112]. Опуская подробности, эта процедура оказала заметное позитивное влияние на процесс обучения. Нормализация проводилась с использованием функции «normalize» модуля «sklearn.preprocessing».

Последнее преобразование данных касается разделения данных на тренировочную и обучающую выборки. Для этого используется метод «train_test_split» модуля «sklearn.preprocessing». По умолчанию, этот метод в тестовую выборку помещает 25% процентов исходной. В именованном аргументе «stratify» мы указали столбец отклик, для того, чтобы исходное соотношение наблюдений с дефолтом и без сохранилось и внутри производных выборок. В аргументе «random_state» можно указать произвольное число, это обеспечит одинаковое разбиение при различных запусках программы.

Теперь, наконец, перейдем к описанию процесса обучения модели. Данные после ОНЕ содержат 113 столбцов, потому входной слой нейронной сети должен содержать столько же нейронов.

Изначально мы пытались использовать подход, который позволял нам рассматривать сразу некоторое множество идентификационных форм модели, но он не дал особого результата – каждая конкретная модель вела себя по разному и для их улучшения требовались уникальные настройки алгоритма обучения. Потому мы сконцентрировались на конкретной модели – неплохие результаты получались при использовании 113 же нейронов в скрытом слое.

После ряда экспериментов мы пришли к выводу, что лучше всего, во всяком случае вначале, использовать размер батча в 500 наблюдений. Примерно также мы определились с начальным learning rate в одну сотую.

На листинге И.4 представлен полный код как просто запустить модель на обучения, используя все те техники и параметры, которые мы описали выше. Заметим лишь, что под именами «X_train», «y_train», «X_test», «y_test» определены разбитые на тренировочную и тестовую выборки предикторы и столбец отклик, а запись «torch.manual_seed(0)» непосредственно перед созданием экземпляра модели обеспечит постоянство весов вне зависимости от запуска программного кода.

Кривые обучения, за первые 20 эпох обучения модели, представлены на рисунке 3.3.

Слева представлены результаты оптимизации за все 20 эпох. Первый шаг оптимизатора дает очень значимое снижение ошибки, в результате вся последующая оптимизация совсем не видна на графике, для того мы справа показали меньше эпох. Впредь на обучающих кривых будут опущены первые эпохи потому читателю стоит обращать внимание на ось ординат

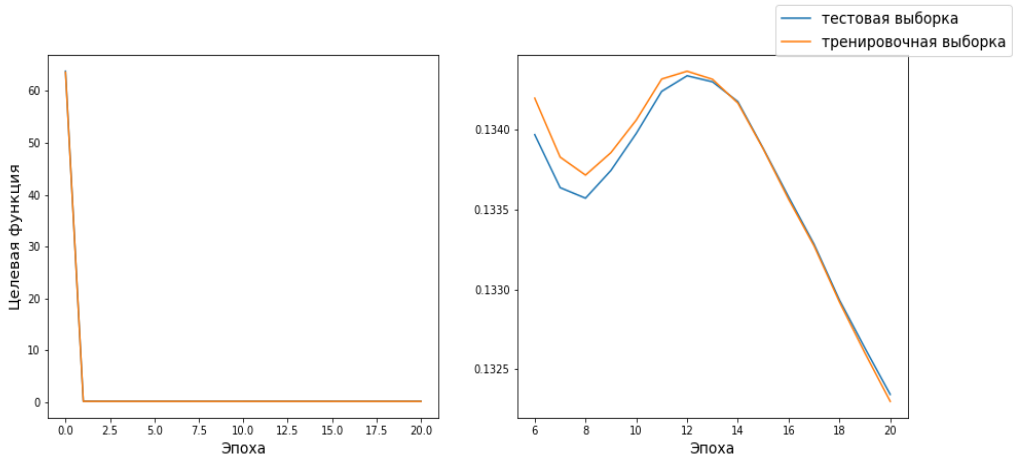

Рисунок 3.3 – Обучающая кривая за 20 эпох
Примечание – Источник: собственная разработка.

Тут может возникнуть резонный вопрос: почему с 8 по 10ю эпохи модель ухудшает свои результаты, хотя алгоритм обратного распространения ошибки однозначно всегда должен быть направлен в сторону наискорейшего спада целевой функции? Это сыграла свою роль техника обучения модели с использованием разбиения на батчи – целевую функцию мы вычисляем на полных данных, но при использовании некоторых батчей для определения градиента алгоритм «не видел» локального минимума возле которого находился в момент 9ой эпохи, в результате успешно пропустил этот минимум продолжил оптимизацию. По завершению оптимизации были снят AUC модели, он представлен в таблице Л.1 в строке соответствующей 20 эпохам.

Как видно, по рисунку оптимизация могла быть успешно продолжена – последние эпохи наблюдалось стремительное улучшение целевой функции. Потому, попробуем доучить модель еще на 30 эпох. Для воплощения этого в жизнь достаточно лишь еще раз вызвать метод «fit» ранее созданного «lay1_trainer», указав в аргументе «epochs» требуемое число эпох

Кривые обучения за 50 полученных таким образом эпох представлены на рисунке 3.4.

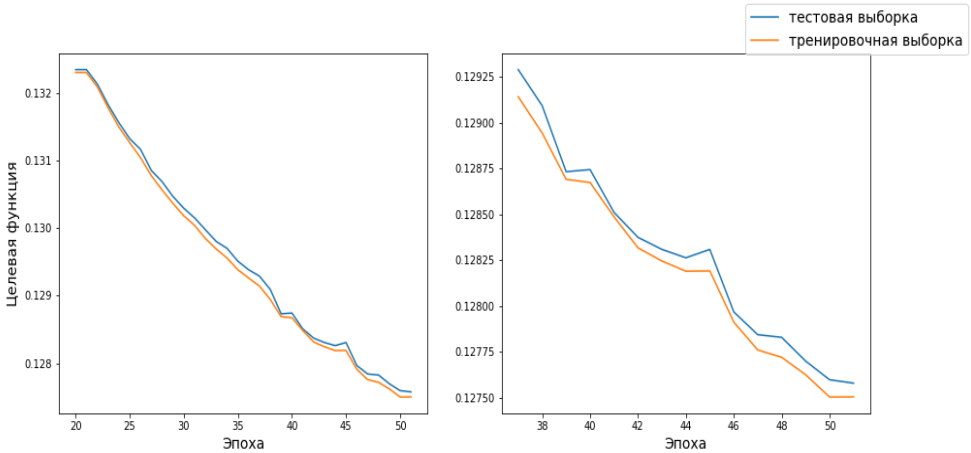

Рисунок 3.4 – Обучающая кривая за 50 эпох
Примечание – Источник: собственная разработка.

Метрика AUC модели после этого шага представлена в таблице Л.1 в строке соответствующей 50-ти эпохам.

Как и ожидалось, оптимизация была продолжена, но обратим внимание, что на более поздних этапах целевая функция не слишком плавно начинает убывать, появляются флуктуации. Это следствие того, что иногда делаются слишком большие шаги не гарантирующие уменьшение ошибки. Для решения этой проблемы можно понизить параметр learning rate. Переопределив оптимизатор с параметром «lr» равным 0,0001, проведем еще 50 эпох обучения модели. Данный шаг находит отражение в рисунке 3.4. А характеристика AUC указана с строке соответствующей сотне эпох таблицы Л.1.

Как видно, это помогло добиться более плавной оптимизации, но вместе с тем после 50-й эпохи скорость оптимизации заметно падает. Вне описания этой работы еще лежит необходимость экономии вычислений – важная часть работы инженера по машинному обучению связанная с экономией ресурсов затрачиваемых на вычисления. К сожалению, современная вычислительная техника не всегда позволяет просчитать все варианты, потребуется недопустимо много времени, электроэнергии или денег на аренду вычислительных мощностей на стороне. Сейчас мы столкнулись с одним из примеров, дейсвительно используя меньший learning rate у нас больше шансов подобраться к минимуму более точно, но нельзя забывать, что в каждый шаг вложена достаточно сложная вычислительная процедура.

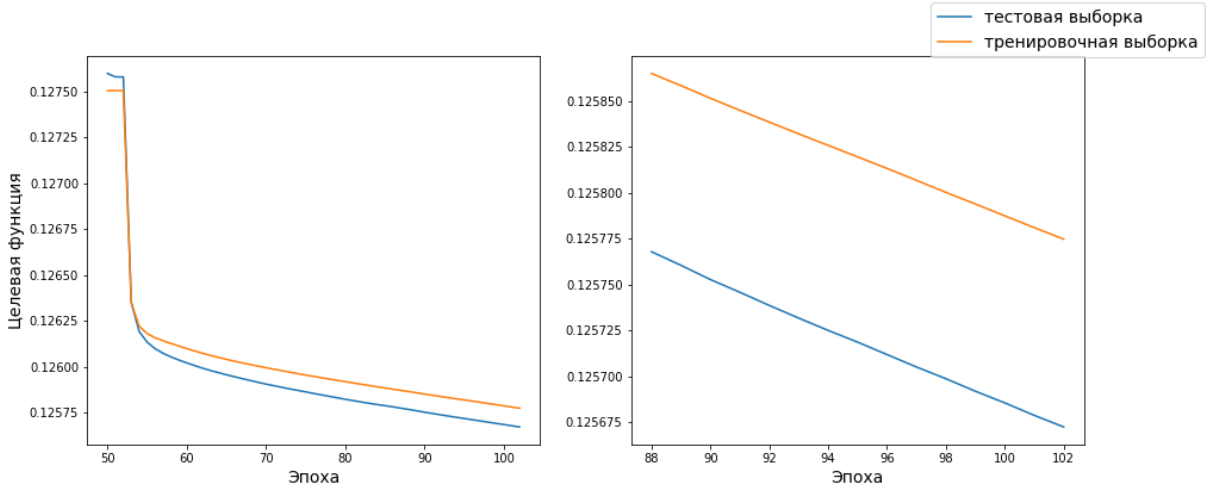

Рисунок 3.5 – Понижение learning rate

Примечание – Источник: собственная разработка.

Используя только описанные выше техники было очень сложно добиться пущего улучшения модели, потому следующий пункт мы начнем с описания тех улучшений которые нам потребовались дополнительно.

Этот пункт дает представление о том как начать обучение модели от технической подготовки данных и до первых заметных результатов, но тут не указано множество экспериментов, которые пришлось провести прежде чем мы пришли к такому состоянию, а указан лишь самый успешный случай. Тем не

менее, по результатам этого раздела мы уже пришли к заметным результатам так на момент завершения раздела имеется модель с показателем AUC в 0,7792 на тренировочной выборке, что уже более чем допустимо. Но, как будет доказано далее, это не предел.

3.2.2 Более тонкая подгонка и валидация финальной модели

В предыдущем пункте мы остановились на том, что после определенной итерации процесс оптимизации теряет стабильность в результате слишком большого шага оптимизации. Позже это вовсе может привести к тому, что целевая функция вообще не может оптимизироваться и даже может показаться системный рост. Но вместе с этим решения связанные с радикальным понижением learning rate нам так же подходили в виду слишком медленного процесса оптимизации, кроме того, алгоритм становиться более чувствительным к локальным минимумам. Проблемы связанные с большей вероятностью попадая в локальный минимум можно конечно обойти с помощью меньшего размера батча, но это также приводит к повышению вычислительных затрат – в каждую эпоху будет включено больше шагов.

Описанные проблемы можно решить динамическим изменением learning rate в зависимости от стадии обучения. Конечно, можно было бы самостоятельно описать логику, но в pytorch реализован специальный набор инструментов – планировщики learning rate. Более подробное описание можно найти в [15] в разделе «How to adjust learning rate».

Для его использования нужно создать экземпляр одного из классов-оптимизаторов или, даже, создать собственный наследуя от одного из уже созданных. При создании, первым обязательным аргументом указывается оптимизатор вместе с которым участвует планировщик. Для задействования оптимизатора в алгоритме рекомендуется сразу после вызова метода «step» оптимизатора вызвать тот-же метод у планировщика.

При проектировании класса-учителя модели мы сразу заложили возможность использования планировщика, достаточно лишь в конструктор именованным аргументом передать созданный планировщик.

Так же достаточно трудоемко вести оптимизацию в полу ручном режиме, потому был предусмотрен алгоритм остановки оптимизации на случай остановки уменьшения целевой функции. Тут напомним случай рисунка 3.2 справа, несмотря на то, что функция возрастает начиная с 8-ой эпохи, к 12-й оптимизация продолжается и к 20-й достигает куда более хороших результатов. Представленный пример описывает, что просто контролировать чтобы значение убывало по сравнению с предыдущей итерацией недостаточно.

Мы предлагаем следующий подход – будем контролировать не по предыдущей эпохе, но через произвольно выбираемое число эпох. Для указания числа эпох через которое начинается осуществляться проверка у метода «model_trainer.fit» предусмотрен именованный аргумент «check_epochs». И так теперь более подробно, если на каждой конкретной эпохе значение целевой

функции на тестовых данных оказывается больше нежели «check_epochs» эпох назад то алгоритм прерывается, не взирая на то, сколько эпох указано для обучения модели.

Кроме того предусмотрено сохранение лучшей, в смысле целевой функции на тестовых данных, модели и советующей ей эпохи в полях «best_model» и «best_epoch». Нужно это потому, что приведенный критерий остановки в общем случае допускает повышение целевой функции и таким образом мы отловим лучшую модель.

Начнем описание конкретной реализации всех описанных идей. Мы будем использовать планировщик «optim.lr_scheduler.ExponentialLR» этот планировщик в конструктор принимает именованный аргумент «gamma» который показывает во сколько раз после каждой эпохи уменьшается learning rate, понятно, что нужно указывать число меньше единицы.

Вот пример, как может быть создан планировщик:

lr_scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma = 0.99).

Расскажем о неудачных попытках применения этой технологии. Изначально выставили «gamma» 0,95 и за 300 эпох было модель не достигла значимого результата. В таблице Л.1 последняя ситуация представлена в строке соответствующей тремя сотням итераций и 0,95 «gamma» – результаты даже хуже, чем без использования нововведений. Объяснение такого поведения объясняет рисунок Л.1. Там алгоритм останавливается досрочно после выхода на плато, мы сделали вывод, что такое поведение связано со слишком быстрым уменьшением learning rate.

В результате параметр «gamma» в планировщике было решено увеличить до 0,99. Результаты модели обученной при таких параметрах представлены в строке с 300 эпох и 0,99 «gamma». Уже достаточно хорошо и то, что функция обучающей выборки постепенно отрывается от тестовой, свидетельствует о постепенно появляющемся переобучении модели. Но, после изучения поля «best_epoch» облучателя модели, по значению 300 стало понятно, что не выполнилось выставленное требование к остановке модели, потому, скажем доучивать модель до тысячи эпох в надежде что алгоритм остановиться.

После около десяти минут вычислений на персональном компьютере базовой комплектации, алгоритм остановился. Характеристика качества полученного классификатора представлена в таблице Л.1 в строке соответствующей тысяче эпох и параметру «gamma» 0,99.

Для того, чтобы удостовериться, что мы действительно вышли на плато на рисунке 3.6 рассмотрим полученную кривую обучения. По рисунку, во первых, можно понять, что мы действительно вышли из алгоритма по критерию остановки на 516-й эпохе, во вторых, что скорее всего мы выжали все возможное из этой модели – последние три эпохи нет совершенно никакого улучшения значения целевой функции ни по тестовым ни по тренировочным данным.

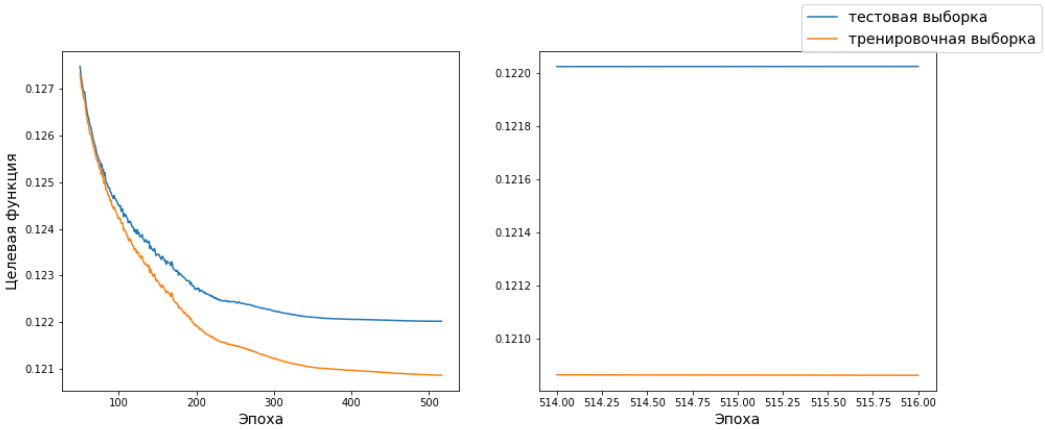

Рисунок 3.6 – Кривая обучения последней модели
Примечание – Источник: собственная разработка.

Еще у нас была идея путем увеличения батча увеличить точность той «ямы» на которой сошелся алгоритм. Мы пересоздали загрузчик тренировочных данных и установили его «batch_size» в размер тренировочных данных. При попытках так доучить модель дейсвительно получается добиться некоторого улучшения значения целевых функции, но они были настолько не значимые, что AUC полученных моделей становился даже хуже, чем у лучшей зафиксированной модели. Это, кстати, еще одна важная идея – у модели с минимальной целевой функции из рассмотренных не обязательно наименьший AUC хотя некоторая связь конечно наблюдается. В связи с этим можно было бы редактировать критерий выбора лучшей модели по наивысшему AUC но такая модернизация уже выходит за рамки этой работы.

В целом, модель не плохая, потому остановимся на ней.

На 3.6 слева можно заметить некоторые флуктуации в начале обучающих кривых. Модель все равно сошлась но настроив обучение так, чтобы убрать эти флуктуации можно было бы увеличить скорость обучения. Но модель уже обучена, потому в этом нет необходимости лишь с оговоркой описанной ниже.

Для того, чтобы не было необходимости каждый раз ждать пока модель обучится при каждой новой рабочей сессии (а то и чаще), pytorch предлагает возможности сохранения модели. Для того может быть использована функция «torch.save», первым аргументом следует передавать значение, которое возвращает метод «state_dict» объекта модели, вторым путь для сохранения. Например, мы делали так:

```
torch.save(lay1_trainer.best_model.state_dict(), 'model 1lay 113
neurons_after_bigfit').
```

Для загрузки используется метод «torch.load», притом загрузка должна проводиться в уже созданный объект модели соответствующей идентификационной формы, через его метод «load_state_dict», мы, например, делали так:

```
model = ResultNet([113, 113])
model.load_state_dict(torch.load('model 1lay 113 neurons_after_bigfit')).
```

Это особенно важный этап при валидации модели, потому как он никак не может проводится без модели, а в каждой новой рабочей сессии проводить оптимизацию очень затратно.

На рисунке Л.2 предложена ROC кривая финальной модели. При ее вычислении на тестовых данных. Подробно ROC анализ мы обсуждали в во второй главе. Тут разница лишь в том что *FPR* и *TPR* вычисляются на предсказаниях вероятностей полученных из модели. Площадь под этой кривой является хорошим способом оценить качество модели, такой показатель называется *AUC*. В данном случае *AUC* округляется до 0,800. От начальства была получена установка добиться *AUC* не менее 0,7, это требование успешно выполнено. Хотя, скорее всего, возможно добиться и лучшего результата.

Для выбора той вероятности при которой потенциальный кредитополучатель считается склонным к невозврату кредита используем ту же идею, которая была использована при разбиении числовых переменных на бинарные. Найдем *KS* статистику для эмпирических распределений наблюдений реально в статусе дефолта и без него по предсказанной моделью вероятности. Тут нам сама по себе эта статистика не важна, важна та оценка вероятности при которой она достигается. Именно при этой точке достигается максимальное расстояние между долей правильных предсказаний модели и долей ошибок, что можно, ближе к практике, описать так – доля правильных предсказаний достаточно велика при недопущении заметного возрастания ошибки. Заметим также, что в отличие от вычисления характеристик классифицирующей способности, точку отсечения лучше рассчитывать на полных данных, в данном случае нет проблем с переобучением а нужно выбрать точку максимально советующую генеральной совокупности. Исчерпывающая информация о таком анализе помещена на рисунок 3.7.

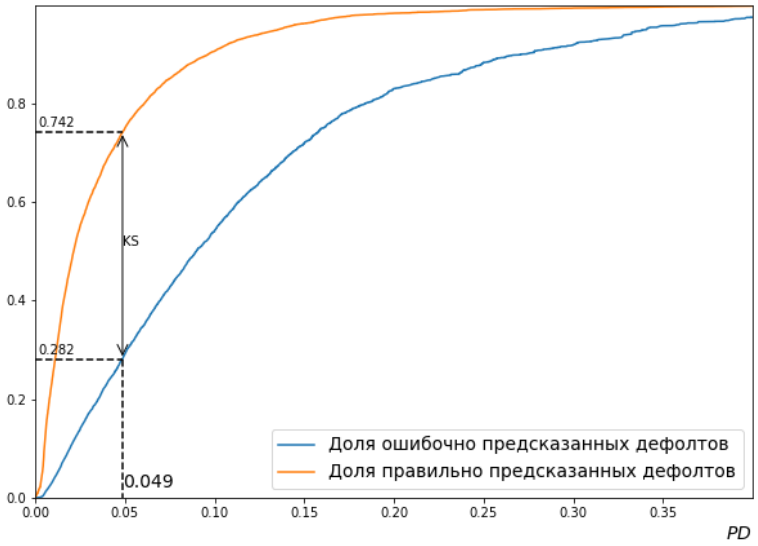

Рисунок 3.7 – Подбор точки отсечения

Примечание – Источник: собственная разработка.

В соответствии с представленным критерием точку отсечения лучше всего выбрать на при вероятности 0,049. При таком выборе точки отсечения удается 74,2% правильных предсказаний для всех дефолтов и 100%-28,2%=71,8% правильных предсказаний для всех кредитополучателей погасивших задолженность.

В таких вопросах всегда лучше иметь наиболее пессимистичную оценку, потому посмотрим как будет вести себя полученная модель при выбранной точке отсечения для тестовой выборки более подробно. Таблица 3.1, по определению, аналогична таблице 2.3, но заполненная числами характеризующими полученную модель для тестовых данных.

Таблица 3.1 – TP, FP, TN, FN для полученной модели

Истинный класс	Предсказанный класс		Всего
	0	*1*	
0	42794	9556	52350
1	674	1067	1741
Всего	43468	10623	54091

Примечание – Источник: собственная разработка.

А ее аналог в относительных величинах – таблица 3.2.

Таблица 3.2 – TPR, FPR, TNR, FNR для полученной модели

Истинный класс	Предсказанный класс	
	0	*1*
0	81,74%	18,26%
1	38,71%	61,29%
Всего	80,36%	19,64%

Примечание – Источник: собственная разработка.

В этих таблицах нулем кодируется клиент без дефолта с единицей с дефолтом.

Как видно, на тестовых данных получается отличный классификатор клиентов без дефолта и удовлетворительный классификатор клиентов с дефолтом.

Заметим, что при выборе точки отсечения почти никогда не применяют подобные вычислительные методы, зачастую все зависит от политики банка и решение принимается ответственным управленцем. Тем не менее, разработчик модели может ориентироваться на такой метод, когда просят представить его мнение.

Данный раздел работы наполнен техническими деталями связанными с использованием библиотеки pytorch языка программирования python для реализации поставленной задачи. Первым делом были представлены базовый минимум для начала работы с библиотекой – описано как создавать и использовать классы моделей, оптимизаторы, и целевые фукнции. Через созданный класс «ResultModel» полноценную реализацию, во всех возможных

уточнениях идентификационной формы получила модель рассматриваемая в первой главе. В качестве оптимизатора было решено использовать алгоритм Adam. Получила полноценное описание целевая функция бинарной кросс энтропии.

После того, читатель начинает погружаться в реализацию алгоритма обратного распространения ошибки и то, какую роль в ней играют описанные выше элементы. Обоснована необходимость и описывается идея обучения по эпохам – способа обучения модели при котором каждый шаг алгоритма обратного распространения ошибки вычисление градиента производиться не на полном наборе данных, но на некоторой его части. В pytorch обучение по эпохам реализовано с использованием загрузчиков данных, процесс объявления и создания которых подробно описан в пункте 3.1.2.

Для обучения модели создан отдельный класс языка программирования python – «Model_trainer» (листинг 3.2). Использовать для реализации операции объектно-ориентированный подход сначала казалось не здравым решением, но при обучении приходиться оперировать многими переменными, потому в конце концов класс оказался более эргономичным нежели функция. Удобно в конструкторе указать относительно постоянные элементы: модель, оптимизатор и целевую функцию; а обучение вывести в отдельный метод который можно вызывать по любой потребности в обучении или «доучивании» модели со специфичными настройками.

Перед тем как приступить, наконец, к обучению модели мы привели еще три преобразований над данными: OHE, нормализация и разбиение на тестовую и тренировочную выборки.

Мы решили вырезать предварительные неудачные версии моделей, чаще всего просто не удавалось добиться установленного минимального значения в 0,7 центральной метрики качества модели – показателя AUC. Достаточно неплохие результаты показала модель с одним скрытым слоем и 113 нейронами в нем. На исследовании ее процесса обучения и попытки улучшить ее показатели были направлены все наши дальнейшие усилия.

При диагностике модели мы в основном обращали внимание на кривые обучения. В работе нет даже половины всех базовых конфигураций кривых обучения, тем не менее, те ситуации в которые попадали мы, тщательно описаны и предложены возможные пути выхода из них. Так если кривая заканчивается на стремительном спаде, следует увеличить число эпох или изменить критерий остановки алгоритма, если кривая обучения на некотором шаге теряет стабильность то следует искать возможность уменьшить шаг обучения, например, через уменьшение параметра learning rate и действовать наоборот в случае если кривая начала заметно замедляться в спуске.

Действуя таким образом нам удалось развить модель от 0,75 до 0,78 показателя AUC. Далее было уже сложно в ручную конфигурировать настройки learning rate потому мы решили включить в алгоритм обучения некоторый компонент, который мог бы полуавтоматически проводить настройку. В pytoch это можно сделать с использованием планировщика learning rate, возможность

его использования, кстати, также заложена в класс «model_trainer». Мы применяли достаточно простой планировщик понижающий learning rate каждую эпоху в установленное число раз. Но даже с таким простым правилом обучения следуют обращаться аккуратно так, при первой попытке использования планировщика, установив слишком высокую скорость понижения learning rate, мы даже ухудшили результаты по сравнению с моделями полученными без его использования.

В конце концов, нам удалось подобрать некоторые настройки, которые привели к появлению модели у которой показатель AUC составил 0,8, что считается более чем удовлетворительным результатом. Все дальнейшие попытки улучшить модель не увенчались успехом потому эту модель мы оставили как финальную.

Кроме *AUC* для классификационной модели разумно подсчитать долю правильных предсказаний для каждого из класса. При точке отсечения выбранной, опираясь на *KS* статистику, названные показатели модели можно увидеть в таблицах 3.1 и 3.2. Модель отлично справляется с выделением хорошего клиента определяя более 80% правильно, но несколько проседает с определением клиента с зафиксированным дефолтом, предсказывая 61%. Это обыденная ситуация в кредитном скоринге, опытные аналитики связывают это с тем, что в выборке этой предметной области всегда доминируют наблюдения без дефолта, а те, что вышли в дефолт нередко выходят по уникальным причинам без прослеживаемой статистической закономерности. В целом, показатели модели удовлетворительные. На этом считаем целесообразным закончить данную работу.

ЗАКЛЮЧЕНИЕ

По завершению исследования, приведенного в данной работе, мы пришли к моделям искусственных нейронных сетей, подготовили набор данных и построили модель соответствующей идентификационной формы. Далее выводы по каждому из разделов.

Первая глава начиналась с постановки задачи классификации, кратко: имея набор данных вида как в таблице 1.1 нужно сформировать правило позволяющее проводить классификацию наиболее точно в некотором, пока точно не определенном, смысле, то есть предсказывать элемент столбца Y (отклик) располагая только элементами столбцов X (предикторами, показателями, переменными).

Модель линейной регрессии оказалась плохим решением, так как она предполагает численный тип отклика а у нас он номинативный, из того вытекали два основные последствия: предсказания выходят за границы возможных вероятностей и переход от одного класса к другому происходит линейно, что осложняет их разделение по предсказанным вероятностям.

Для решения этой проблемы в логистической регрессии используется особая функция – логит, свойства которой позволяют избавиться от названных проблем. Однако логистическая регрессия остается линейным классификатором. Это происходит потому, что решение о отнесении к тому или иному классу принимается опираясь на некоторую точку отсечения – если предсказана большая вероятность считается, что исследуемый признак проявился, получается, мы выбираем некоторую линию уровня после которой считаем наблюдения принадлежащие к тому или иному классу. Несложно доказать, что линии уровня логит функции линейные, потому и правило классификации получается линейным. В работе предложены достаточна понятная визуализация на рисунках 1.3 и А.1.

Линейное правило бывает слишком простым, очень во многих случаях его недостаточно. На рисунке 1.4 показано как может справиться логистическая регрессия с нелинейным случаем. В самом деле, результат не слишком плохой но добавив в модель нелинейность можно добиться стопроцентной точности классификации.

Нелинейность может быть добавлена путем усложнения идентификационной формы модели. Мы предлагаем использовать кусочно-линейную функцию под логит функцией, таким образом «ломать» дискриминирующую прямую в нужных местах. Абсолютно аналогичные модели получатся при использовании идей искусственных нейронных сетей при включении логит нейрона в выходном слое и ReLU нейронов в скрытом слое. Как решает задачу с которой не справился алгоритм логистической регрессии подобная модель искусственной нейронной сети представлено на рисунке А.2.

По завершению первой главы мы коснулись вопроса оценки коэффициентов в нейронной сети. Популярным методом оценки коэффициентов является алгоритм обратного распространения ошибки. Формально, это тоже

градиентный спуск (метод нелинейной оптимизации), в удобной записи для работы с моделями типа нейронной сети.

Для построения модели был получен набор данных содержащий 247062 записи и 45 показателей. Сразу было решено исследовать типы входящих переменных, для того использовался созданный самостоятельно инструмент, позволяющий формировать таблицы, подобные тем, что представлены в приложении Г. Опираясь на информацию из этих таблиц, мы сразу сделали выводы относительно целесообразности использования некоторых столбцов. Так сразу опустили столбцы которые содержат слишком много уровней и проводить укрупнение не целесообразно. Другие столбцы пытались укрупнить. Создавали новые столбцы с, предположительно, лучшей классифицирующей способностью и т.д.

Особый интерес представляет процедура, которую мы использовали при отборе показателей в модель. Мы оперлись на показатель *AUC* – популярный способ оценить качество готовой модели. Он тем больше, чем сильнее различаются распределения по предсказанным вероятностям для конкурирующих классов. Для того, чтобы использовать его для оценки классифицирующей способности отдельных предикторов, мы рассмотрели однофакторные классификаторы вида (2.5) и для них вычисляли *AUC*. В данном случае даже важнее *KS* статистика, а точнее, соответствующий тест Колмогорова-Смирнова для двух выборок. В тесте мы сравнивали распределения хороших и плохих клиентов вдоль изучаемого предиктора и не допускали показатель в модель в том случае, если удавалось подтвердить нулевую гипотезу об однородности выборок. Это делалось автоматически благодаря созданному инструменту, результатом его работы стали таблицы, подобные таблицам приложения Ж.

Заканчивается второй раздел описанием способов избавления от пропусков. Для номинативной переменной пропуск воспринимался как отдельный уровень. Для численной применялось при основных подхода: перекодировка в по принципу «есть данные/нет данных», перекодировка с использованием точки соответствующей *KS* статистике в качестве разделителя классов и заполнение пропусков нулем. Полученные таким образом переменные сравнивались (в том числе и с исходной переменной) и из них выбиралась наилучшая соответствующая здравому смыслу.

Для построения модели в работе используется библиотека для создания нейронных сетей pytorch. Первым делом описываются самые базовые принципы: оптимизатор, целевая функция и класс модели. Оптимизатор используется Adam. В качестве целевой функции задействована бинарная кросс-энтропия. Класс для описания модели представлен на листинге И.1. Для реализации базового алгоритма обратного распространения ошибки этого, кончено достаточно, но на практике сложно получить качественные оценки параметров используя только их.

Далее мы описываем более продвинутые (но по прежнему базовые) техники используемые для обучения моделей. Это разбиение выборки на тренировочные

и тестовые данные и использование обучения по эпохам. Первая техника позволяет избежать феномена переобучения, вторая обойти больше локальных минимумов в задаче оптимизации.

Используя только эти подходы нам удалось обучить модель с одним скрытым слоем и 113 нейронами в нем с удовлетворительной характеристикой в 0,78 *AUC*. Но исследуя кривые обучения, удалось сделать выводы о возможности улучшения модели путем динамического изменения learning rate. Для того использовались особые объекты – планировщики learning rate. С их использованием удалось улучшить модель.

И так, финальная модель достигла 0,80 показателя *AUC* на тестовых данных и, при выборе точки отсечения опираясь на значение соответствующее *KS* статистике, было достигнуто 61% правильной классификации наблюдений с дефолтом и 81% правильной классификации наблюдений без дефолта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Флах, П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / П. Флах; пер. с англ. А.А. Слинкина. – М.: ДМК Пресс, 2015. – 400 с.
2. Введение в статистическое обучение с примерами на языке R / Г. Джеймс [и др.]; пер с англ. С.Э. Мастицкого. – М.: МДК Пресс, 2017. – 456 с.
3. Hosmer, D.W. Applied Logistic Regression / D.W. Hosmer, J.S. Lemeshow // University of Massachusetts. – 2-nd ed. – 2000. – 376 p.
4. Микелуччи У. Прикладное глубокое обучение. Подход к пониманию глубоких нейронных сетей на основе метода кейсов / У. Микелуччи; пер. с англ. – СПб: БХВ-Петербург, 2020. – 368 с.
5. Грас Дж. Data Science. Наука о данных с нуля / Дж. Грас; Пер. с англ. – СПб.: БХВ-Петербург, 2017. – 336с.
6. Pandas, main page [Electronic resource]. – 02.04.2022. – Mode of access: https://pandas.pydata.org/. – Date of access: 10.04.2022.
7. Numpy, main page [Electronic resource]. – 2022. – Mode of access: https://numpy.org/. – Date of access: 10.04.2022.
8. Scipy, main page [Electronic resource]. – 05.02.2022. – Mode of access: https://scipy.org/. – Date of access: 10.04.2022.
9. Scikit-learn, machine learning in python [Electronic resuorce]. – 2021. – Mode of access: https://scikit-learn.org/stable/. – Date of access: 10.04.2022.
10. PyTorch, main page [Electronic resource]. – Mode of access: https://pytorch.org/. – Date of access: 10.04.2022.
11. Виды экономической деятельности, общегосударственный классификатор Республики Беларусь. – Минск, 2011. – 364 с.
12. Выброс (статистика) / Википедия, свободная энциклопедия [Электронный ресурс]. – 2021. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D0%B1%D1%80%D0%BE%D1%81_(%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0). – Дата доступа: 10.04.2022.
13. Lopes, R.H.C. The two-dimentional Kolmogorov-Smirnov test / R.H.C. Lopes, I. Reid, P.R. Hobson // XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research. – 2007. – Amsterdam, Netherlands.
14. Loss functions/ PYTORCH DOCUMENTATION [Electronic resource]. – Mode of access: https://pytorch.org/docs/stable/nn.html#loss-functions. – Date of access: 16.04.2022.
15. TORCH.OPTIM/ PYTORCH DOCUMENTATION [Electronic resource]. – Mode of access: https://pytorch.org/docs/stable/optim.html. – Date of access: 16.04.2022.
16. Пойнтер, Я. Программируем с PyTorch: Создание приложений глубокого обучения / Я. Пойнтер. – СПб.: Питер, 2020. – 256 с.
17. Goboy, D. Understanding binary cross-entropy/ log loss: a visual explanation [Electronic resource]. – Mode of access:

https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. – Date of access: 23.04.2022.

18. Хендрик, Б. Машинное обучение / Б. Хендрик, Дж. Ричардс, М. Феверолф. – СПб.: Питер, 2017. – 336 с.

19. Тарик, Р. Создаем нейронную сеть.: Пер. с англ / Р. Тарик. – СПб.: ООО «Альфа-книга», 2017. – 272 с.

20. Novak, G. Building a One Hot Ecoding Layer with TensorFrow[Electronic resource] / Towards Data Science. – 07.06.2020. – Mode of access: https://towardsdatascience.com/building-a-one-hot-encoding-layer-with-tensorflow-f907d686bf39. – Date of access: 20.04.2022.

21. Шолле, Ф. Глубокое обучение на python / Ф. Шолее. – СПб.: Питер, 2018. – 400 с.

ПРИЛОЖЕНИЕ А
Визуализация сигмоиды двух переменных

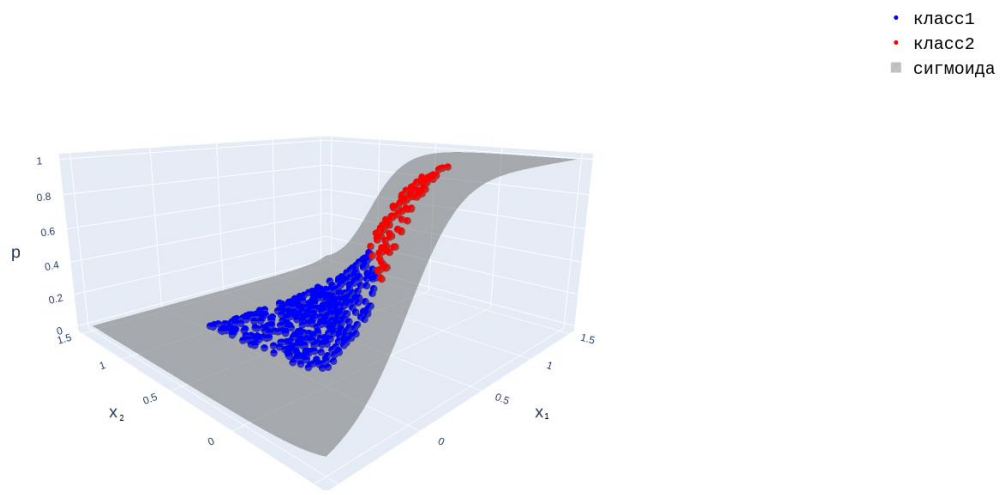

Рисунок А.1 – Сигмоида линейной модели двух переменных
Примечание – Источник: собственная разработка.

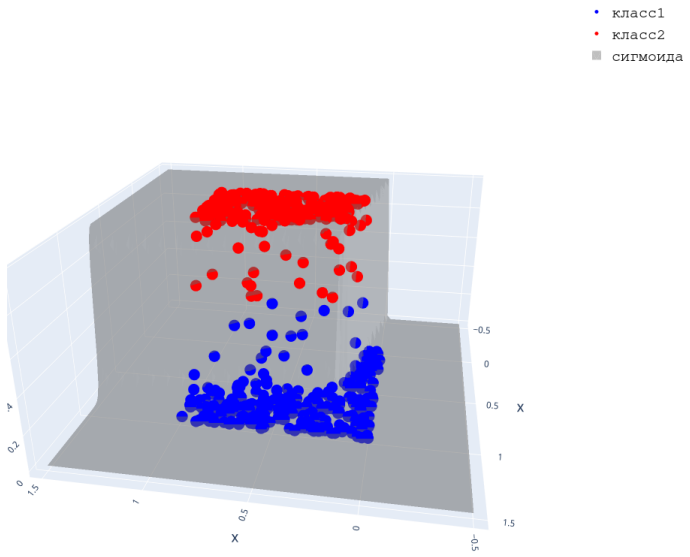

Рисунок А.2 – Сигмоида кусочно-линейной модели двух переменных
Примечание – Источник: собственная разработка.

ПРИЛОЖЕНИЕ Б
Архитектуры с сигмоидой на выходном слое

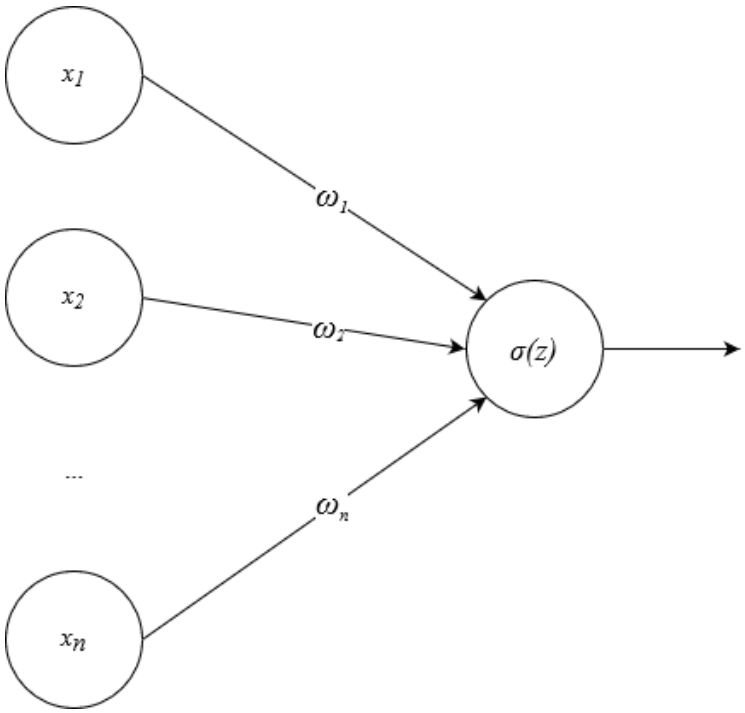

Рисунок Б.1 – Логистическая регрессия как нейронная сеть
Примечание – Источник: собственная разработка.

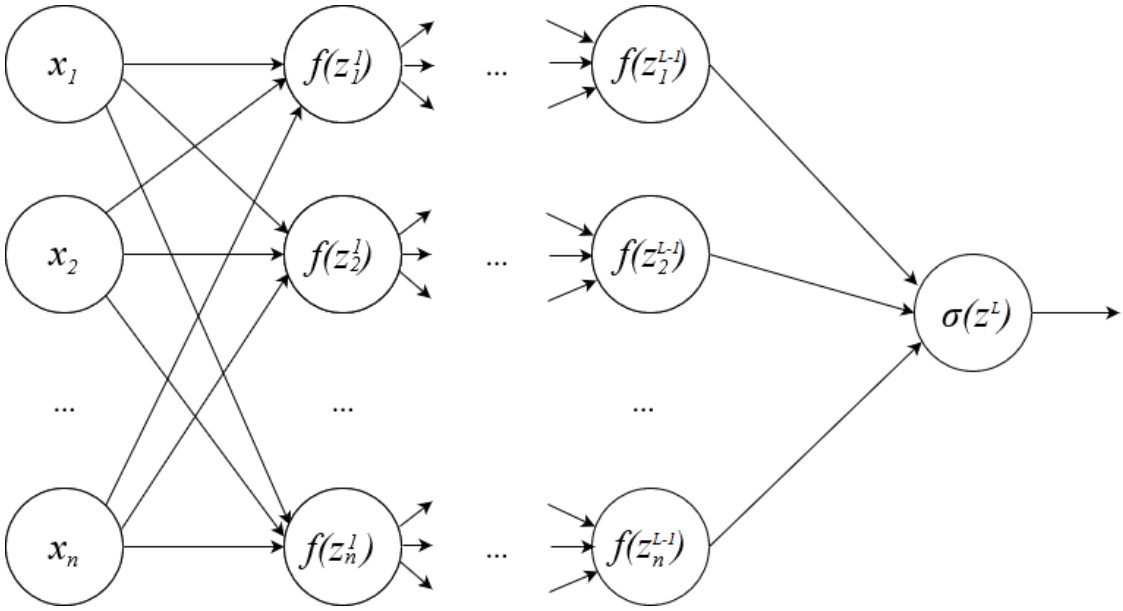

Рисунок Б.2 – Центральная архитектура сетей используемая в работе
Примечание – Источник: собственная разработка.

ПРИЛОЖЕНИЕ В
Обобщенные программные функции процессинга данных

```
import pandas as pd
import numpy as np

types_natural_names = {
   "datetime64[ns]" : 'Дата',
   "category" : "Номинативная",
   "bool" : "Номинативная",
   "int64" : "Целое число",
   "int32" : "Целое число",
   "float64": "Действительное число"
}



def get_col_av_values(col):
   '"Получить область допустимых значений колонки"'
   if pd.api.types.is_numeric_dtype(col):
      return "[" + str(np.min(col)) + ";" + str(np.max(col)) + ']'
   if pd.api.types.is_datetime64_any_dtype(col):
      return "[" + np.min(col).strftime("%d.%m.%Y") + ";" +\
            np.max(col).strftime("%d.%m.%Y") + ']'
   else:
      return str(col.cat.categories.tolist())[1:-1].replace("'", "")

def get_col_cat_count(col):
   '"получить для категориальной переменной число
   уровней, а для численной "-"'
   if pd.api.types.is_numeric_dtype(col) or\
      pd.api.types.is_datetime64_any_dtype(col):
      return "-"
   else:
      return len(col.cat.categories)



def get_data_frame_settings(df):

   result = pd.DataFrame(columns = [
      'Тип данных',
      'Область допустимых значений',
      'Число допустимых значений',
      'Число пропусков'
   ])

   for col in df.columns:
      result.loc[col, :] = [
         df[col].dtype,
         get_col_av_values(df[col]),
```

```
            get_col_cat_count(df[col]),
            sum(df[col].isna())
        ]

    result["Тип данных"] = result["Тип данных"].replace(types_natural_names)
    return result
```

Листинг В.1 – Функция python3 для получения описания типов данных столбцов произвольного pandas.DataFrame

Примечания

1. Источник: собственная разработка,
2. Для корректной работы функций предварительно необходимо установить библиотеки pandas, numpy.

```
def get_describes(df):
    result = pd.DataFrame()
    for col in df:
        result[col] = df[col].describe()
    return result
```

Листинг В.2 – Функция python3 для получения описательных статистик произвольного pandas.DataFrame

Примечания:

1. Источник: собственная разработка.
2. Для корректной работы функций предварительно необходимо установить библиотеку pandas.

```
import numpy as np
# this file represents several of fucntions wich
# can be used for emissions management

def get_frame_quantiles_25_75(col):
    '''The function returns 25 and 75 persentiles of getted column'''
    # inputs:
    # col - column which persentiles to be computed
    # output:
    # {'25%': <25th persentile>, "75%":<75th persentile>}
    descr = col.describe()
    return {'25%' : descr['25%'], '75%' : descr['75%']}


def get_selcond_emiss_25_75(col, constant = 1.5, cut_type = "both" ):
    '''Funciton for getting selection condition from
    column, which that will get rid of the emissions.
    According with rule:
https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D0%B1%D1%80%D0%BE%D1%81_(%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0),
    '''
    #inputs:
```

```
    # col - pandas.Series column which emissions needs to be cleared
    # constan - int which adjusts the width of the deleted interval
    # but_type -    "left": deletes only left tale
    #               "right": deletes only right tale
    #               "both": deletes both tales
    # outputs:
    # pd.Seires with boolean type - False if the observation interpreted as an emission
    #                               else True

    result = np.ones(col.shape).astype('bool')
    quant = get_frame_quantiles_25_75(col)

    range_25_75 = quant['75%'] - quant['25%']

    if (cut_type == "right") | (cut_type == "both"):
        result = result & (col <= (quant['75%'] + range_25_75 * constant))
    if (cut_type == "left") | (cut_type == "both"):
        result = result & (col >= (quant['25%'] - range_25_75 * constant))

    result[col.isna()] = True

    return result

def cut_emissioins(col):
    return col[em.get_selcond_emiss_25_75(col, cut_type = 'right')]
```

Листинг В.3 – Функция python3 для очистки выбросов по правилу (2.1)

Примечания

1. Источник: собственная разработка,
2. Для корректной работы функций предварительно необходимо установить библиотеку numpy.

ПРИЛОЖЕНИЕ Г
Данные на разных этапах обработки

Таблица Г.1 – Начальный формат данных.

Название показателя	Тип данных	Область допустимых значений	Число допустимых значений	Число пропусков
1	2	3	4	5
Автомобиль год выпуска1	Дата	[18.01.1986;22.12.2019]	-	202744
Автомобиль год выпуска2	Дата	[22.01.1986;10.11.2019]	-	245403
Автомобиль год выпуска3	Дата	[11.03.1987;30.05.2019]	-	246960
Воинская служба	Номинативная	военнослужащий, другое, не отслужил, невоеннообязанный, отсрочка, призывник, уволен в запас	7	108914
Количество детей	Действительное число	[1.0;10.0]	-	161319
Количество иждивенцев	Действительное число	[1.0;10.0]	-	237508
Недвижимость	Номинативная	есть, нет	2	40
Образование	Номинативная	высшее, незаконченное высшее, неполное среднее, среднее, среднее специальное	5	70933
Отношение к банку	Номинативная	акционер, другое, не имеет отношения, работник	4	1060
Работа занимаемая должность	Номинативная	безработный (временно не работающий), государственный служащий, заместитель руководителя, индивидуальный предприниматель, пенсионер, рабочий, руководитель, специалист, студент	9	0

Продолжение таблицы Г.1

1	2	3	4	5
Работа последнее место стаж лет	Действительное число	[-998.0;50.0]	-	1035
Работа уровень дохода BYR	Действительное число	[-1500.0;92207.0]	-	923
Семейное положение	Номинативная	вдовец/вдова, женат/замужем, повторный брак, разведен/разведена, холост/не замужем	5	63178
Собственная квартира	Номинативная	есть, нет	2	40
Собственный дом	Номинативная	есть, нет	2	40
Уголовная ответственность	Номинативная	есть, нет	2	40
Адрес проживания - Населенный пункт	Номинативная	Название населенного пункта	6122	0
Адрес проживания - Тип населенного пункта	Номинативная	Агрогородок, Город, Городской поселок, Деревня, Курортный поселок, Поселок сельского типа, Рабочий поселок, Село, Сельский населенный пункт, Хутор	10	80333
Вид деятельности по ОКЭД	Номинативная	Вид деятельности по ОКЭД	44	15253
Гражданин РБ	Номинативная	Без гражданства, Другое, РБ	3	68942
Дата регистрации договора	Дата	[04.01.2016;21.12.2020]	-	0
Дата рождения	Дата	[03.01.1878;30.09.2002]	-	0
Был ли хоть один договор прекращен досрочно	Номинативная	есть, нет	2	137919
Количество действующих договоров обеспечения	Действительное число	[0.0;44.0]	-	179682
Количество действующих кредитных договоров	Действительное число	[0.0;26.0]	-	149948
Количество запросов в КБ за последние 30 дней	Действительное число	[0.0;15.0]	-	135276

Продолжение таблицы Г.1

1	2	3	4	5
Количество фактов просрочки по основному долгу	Действительное число	[0.0;284.0]	-	165097
Максимальное количество дней просрочки	Действительное число	[0.0;4471.0]	-	172113
Максимальный срок, на который заключался договор, в годах	Действительное число	[0.0;122.08]	-	139103
Наличие кредитной истории	Номинативная	есть, нет	2	134471
Общее количество запросов в КБ	Действительное число	[0.0;222.0]	-	135276
Отношение факт срока к плановому при прекращении КД	Действительное число	[0.0;10000.0]	-	145229
Причина прекращения договора	Номинативная	Прекращение договора исполнением, Прекращение договора по иным основаниям предусмотренным законодательством Республики Беларусь, Прекращение договора по решению суда, Прощение долга	4	151260
Сумма кредитных лимитов	Действительное число	[0.0;678562.6]	-	149948
Дата планируемого закрытия	Дата	[20.05.2016;20.07.2058]	-	10566
Дата фактического закрытия	Дата	[15.01.2016;08.07.2021]	-	107885
Кредитный продукт	Номинативная	Название кредитного продукта	262	748
Овердрафт	Номинативная	есть, нет	2	0
Сумма договора	Действительное число	[-549.81;600000.0]	-	0
Количество потребляемых банковских продуктов	Номинативная	1, 2, более 2-х	3	118779

Окончание таблицы Г.1

1	2	3	4	5
Место работы	Номинативная	Наименование предприятия	123712	1445
Пол	Номинативная	Ж, М	2	0
Социальная группа	Номинативная	безработный, индивидуальный предприниматель, пенсионер, работающий по найму, служащий, учащийся	6	89495
Дефолт	Целое число	[0;1693]	-	0
Код подразделения	Номинативная	739-100, 739-200, 739-200-202, 739-200-203, 739-200-228, 739-300, 739-400, 739-600, 739-800, 739-800-831, 739-900, 739-900-500, 739-900-527, 739-900-535, 739-900-536, 739-900-537, 739-900-538, 739-900-905, 739-900-906, 739-900-907, 739-900-932, 739-900-933	22	0

Примечание – Источник: собственная разработка.

Таблица Г.2 – Формат данных после предварительной обработки

Название показателя	Тип данных	Область допустимых значений	Число допустимых значений	Число пропусков
1	2	3	4	5
Автомобиль год выпуска1	Действительное число	[-2150.0;10241.0]	-	195030
Автомобиль год выпуска2	Действительное число	[-2146.0;10199.0]	-	234900
Автомобиль год выпуска3	Действительное число	[-1733.0;10035.0]	-	236396
Воинская служба	Номинативная	военнослужащий, невоеннообязанный, уволен в запас, не отслужил, отсрочка, другое, призывник, nan	8	108124
Количество детей	Действительное число	[1.0;9.0]	-	155538

Продолжение таблицы Г.2

1	2	3	4	5
Количество иждивенцев	Действительное число	[1.0;10.0]	-	227405
Недвижимость	Номинативная	есть, нет	2	0
Образование	Номинативная	среднее специальное, высшее, среднее, незаконченное высшее, неполное среднее, nan	6	70250
Отношение к банку	Номинативная	не имеет отношения, работник, акционер, другое	4	0
Работа занимаемая должность	Номинативная	специалист, рабочий, руководитель, заместитель руководителя, государственный служащий, пенсионер, студент, индивидуальный предприниматель	8	0
Работа последнее место стаж лет	Действительное число	[-998.0;50.0]	-	369
Работа уровень дохода BYR	Действительное число	[-1500.0;92207.0]	-	769
Семейное положение	Номинативная	женат/замужем, холост/не замужем, разведен/разведена, вдовец/вдова, повторный брак, nan	6	63143
Собственная квартира	Номинативная	есть, нет	2	0
Собственный дом	Номинативная	нет, есть	2	0
Уголовная ответственность	Номинативная	нет, есть	2	0
Адрес проживания - Населенный пункт	Номинативная	витебск, минск, другой, брест, гомель, могилев, гродно	7	0

Продолжение таблицы Г.2

1	2	3	4	5
Адрес проживания - Тип населенного пункта	Номинативная	Город, Деревня, Городской поселок, Агрогородок, Рабочий поселок, Поселок сельского типа, Сельский населенный пункт, nan, Курортный поселок, Село, Хутор	11	80329
Гражданин РБ	Номинативная	РБ, Другое, Без гражданства, nan	4	68942
Дата рождения	Действительное число	[-41611.0;3949.0]	-	0
Был ли хоть один договор прекращен досрочно	Номинативная	nan, есть, нет	3	134681
Количество действующих договоров обеспечения	Действительное число	[0.0;44.0]	-	173674
Количество действующих кредитных договоров	Действительное число	[0.0;26.0]	-	145639
Количество запросов в КБ за последние 30 дней	Действительное число	[0.0;15.0]	-	132363
Количество фактов просрочки по основному долгу	Действительное число	[0.0;284.0]	-	159481
Максимальное количество дней просрочки	Действительное число	[0.0;4471.0]	-	166015
Максимальный срок, на который заключался договор, в годах	Действительное число	[0.0;122.08]	-	135529
Наличие кредитной истории	Номинативная	nan, есть, нет	3	131648
Общее количество запросов в КБ	Действительное число	[0.0;222.0]	-	132363
Сумма кредитных лимитов	Действительное число	[0.0;678562.6]	-	145639

Окончание таблицы Г.2

1	2	3	4	5
Сумма договора	Действительное число	[-549.81;600000.0]	-	0
Количество потребляемых банковских продуктов	Номинативная	более 2-х, 1, 2, nan	4	117790
Пол	Номинативная	М, Ж	2	0
Социальная группа	Номинативная	служащий, работающий по найму, nan, учащийся, пенсионер, безработный, индивидуальный предприниматель	7	89220
Код подразделения	Номинативная	739-600, 739-900, 739-900-500, 739-400, 739-900-536, 739-300, 739-900-932, 739-200-202, 739-800, 739-200, 739-800-831, 739-900-538, 739-900-905, 739-900-906, 739-900-535, 739-900-537, 739-200-228, 739-900-527, 739-200-203, 739-900-933, 739-900-907, 739-100	22	0
Автомобиль год выпуска	Действительное число	[-2150.0;10241.0]	-	194937
Число авто	Целое число	[0;3]	-	0
Есть авто	Номинативная	есть, нет	2	0
Срок кредита в днях	Действительное число	[65.0;14268.0]	-	1
Ежедневный платеж	Действительное число	[-2.924;196.241]	-	1
Столица	Номинативная	Нет, Да	2	0
Областной центр	Номинативная	Да, Нет	2	0
Y	Целое число	[0;1]	-	0

Примечание – Источник: собственная разработка.

Таблица Г.3 – Формат данных после очистки от выбросов

Название показателя	Тип данных	Область допустимых значений	Число допустимых значений	Число пропусков
1	2	3	4	5
Автомобиль год выпуска1	Действительное число	[-2150.0;10241.0]	-	195030
Автомобиль год выпуска2	Действительное число	[-2146.0;10199.0]	-	234900
Автомобиль год выпуска3	Действительное число	[-1733.0;10035.0]	-	236396
Воинская служба	Номинативная	военнослужащий, невоеннообязанный, уволен в запас, не отслужил, отсрочка, другое, призывник, nan	8	108124
Количество детей	Действительное число	[1.0;9.0]	-	155538
Количество иждивенцев	Действительное число	[1.0;10.0]	-	227405
Недвижимость	Номинативная	есть, нет	2	0
Образование	Номинативная	среднее специальное, высшее, среднее, незаконченное высшее, неполное среднее, nan	6	70250
Отношение к банку	Номинативная	не имеет отношения, работник, акционер, другое	4	0
Работа занимаемая должность	Номинативная	специалист, рабочий, руководитель, заместитель руководителя, государственный служащий, пенсионер, студент, индивидуальный предприниматель	8	0
Работа последнее место стаж лет	Действительное число	[-998.0;0.0]	-	217748
Работа уровень дохода BYR	Действительное число	[-1500.0;100.0]	-	221099

Продолжение таблицы Г.3

1	2	3	4	5
Семейное положение	Номинативная	женат/замужем, холост/не замужем, разведен/разведена, вдовец/вдова, повторный брак, nan	6	63143
Собственная квартира	Номинативная	есть, нет	2	0
Собственный дом	Номинативная	нет, есть	2	0
Уголовная ответственность	Номинативная	нет, есть	2	0
Адрес проживания - Населенный пункт	Номинативная	витебск, минск, другой, брест, гомель, могилев, гродно	7	0
Адрес проживания - Тип населенного пункта	Номинативная	Город, Деревня, Городской поселок, Агрогородок, Рабочий поселок, Поселок сельского типа, Сельский населенный пункт, nan, Курортный поселок, Село, Хутор	11	80329
Гражданин РБ	Номинативная	РБ, Другое, Без гражданства, nan	4	68942
Дата рождения	Действительное число	[-41611.0;3949.0]	-	0
Был ли хоть один договор прекращен досрочно	Номинативная	nan, есть, нет	3	134681
Количество действующих договоров обеспечения	Действительное число	[0.0;44.0]	-	173674
Количество действующих кредитных договоров	Действительное число	[0.0;26.0]	-	145639
Количество запросов в КБ за последние 30 дней	Действительное число	[0.0;15.0]	-	132363
Количество фактов просрочки по основному долгу	Действительное число	[0.0;12.0]	-	168714

Продолжение таблицы Г.3

1	2	3	4	5
Максимальное количество дней просрочки	Действительное число	[0.0;20.0]	-	173510
Максимальный срок, на который заключался договор, в годах	Действительное число	[0.0;122.08]	-	135529
Наличие кредитной истории	Номинативная	nan, есть, нет	3	131648
Общее количество запросов в КБ	Действительное число	[0.0;23.0]	-	136244
Сумма кредитных лимитов	Действительное число	[0.0;20464.32]	-	153215
Сумма договора	Действительное число	[0.0;11600.0]	-	20133
Количество потребляемых банковских продуктов	Номинативная	более 2-х, 1, 2, nan	4	117790
Пол	Номинативная	М, Ж	2	0
Социальная группа	Номинативная	служащий, работающий по найму, nan, учащийся, пенсионер, безработный, индивидуальный предприниматель	7	89220
Код подразделения	Номинативная	739-600, 739-900, 739-900-500, 739-400, 739-900-536, 739-300, 739-900-932, 739-200-202, 739-800, 739-200, 739-800-831, 739-900-538, 739-900-905, 739-900-906, 739-900-535, 739-900-537, 739-200-228, 739-900-527, 739-200-203, 739-900-933, 739-900-907, 739-100	22	0
Автомобиль год выпуска	Действительное число	[-2150.0;10241.0]	-	194937
Число авто	Целое число	[0;3]	-	0

Окончание таблицы Г.3

1	2	3	4	5
Есть авто	Номинативная	есть, нет	2	0
Срок кредита в днях	Действительное число	[65.0;14268.0]	-	1
Ежедневный платеж	Действительное число	[0;6.668]	-	12330
Столица	Номинативная	Нет, Да	2	0
Областной центр	Номинативная	Да, Нет	2	0
Y	Целое число	[0;1]	-	0

Примечание – Источник: собственная разработка.

Таблица Г.4 – Формат данных подготовленных для построения модели

Название показателя	Тип данных	Область допустимых значений	Число допустимых значений	Число пропусков
1	2	3	4	5
Воинская служба	Номинативная	военнослужащий, невоеннообязанный, уволен в запас, не отслужил, отсрочка, призывник, другое, нет данных	8	0
Недвижимость	Номинативная	есть, нет	2	0
Образование	Номинативная	среднее специальное, высшее, среднее, незаконченное высшее, неполное среднее, нет данных	6	0
Работа занимаемая должность	Номинативная	специалист, рабочий, руководитель, заместитель руководителя, государственный служащий, пенсионер, студент, индивидуальный предприниматель	8	0
Семейное положение	Номинативная	женат/замужем, холост/не замужем, разведен/разведена, вдовец/вдова, повторный брак, нет данных	6	0
Собственная квартира	Номинативная	есть, нет	2	0
Собственный дом	Номинативная	нет, есть	2	0

Продолжение таблицы Г.4

1	2	3	4	5
Уголовная ответственность	Номинативная	нет, есть	2	0
Адрес проживания - Тип населенного пункта	Номинативная	Город, Деревня, Городской поселок, Агрогородок, Рабочий поселок, Поселок сельского типа, Сельский населенный пункт, нет данных, Курортный поселок, Село, Хутор	11	0
Гражданин РБ	Номинативная	РБ, Другое, Без гражданства, нет данных	4	0
Дата рождения	Действительное число	[-41611.0;3949.0]	-	0
Был ли хоть один договор прекращен досрочно	Номинативная	нет данных, есть, нет	3	0
Наличие кредитной истории	Номинативная	нет данных, есть, нет	3	0
Сумма договора	Действительное число	[0.0;11600.0]	-	0
Количество потребляемых банковских продуктов	Номинативная	более 2-х, 1, 2, нет данных	4	0
Пол	Номинативная	М, Ж	2	0
Социальная группа	Номинативная	служащий, работающий по найму, нет данных, учащийся, пенсионер, безработный, индивидуальный предприниматель	7	0

Окончание таблицы Г.4

1	2	3	4	5
Код подразделения	Номинативная	739-600, 739-900, 739-900-500, 739-400, 739-900-536, 739-300, 739-900-932, 739-200-202, 739-800, 739-200, 739-800-831, 739-900-538, 739-900-905, 739-900-906, 739-900-535, 739-900-537, 739-200-228, 739-900-527, 739-200-203, 739-900-933, 739-900-907, 739-100	22	0
Y	Целое число	[0;1]	-	0
Количество действующих кредитных договоров	Действительное число	[0.0;26.0]	-	0
Количество фактов просрочки по основному долгу	Действительное число	[0.0;12.0]	-	0
Максимальное количество дней просрочки	Действительное число	[0.0;20.0]	-	0
Сумма кредитных лимитов	Действительное число	[0.0;20464.32]	-	0
Автомобиль год выпуска1	Номинативная	<=3173.0, нет данных, >3173.0	3	0
Количество запросов в КБ за последние 30 дней	Номинативная	нет данных, <=2.0, >2.0	3	0
Максимальный срок, на который заключался договор, в годах	Номинативная	нет данных, <=7.94, >7.94	3	0
Общее количество запросов в КБ	Номинативная	нет данных, <=12.0, >12.0	3	0
Срок кредита в днях	Номинативная	<=934.0, >934.0, нет данных	3	0

Примечание – Источник: собственная разработка.

ПРИЛОЖЕНИЕ Д
Графическая интерпретация TP, FP

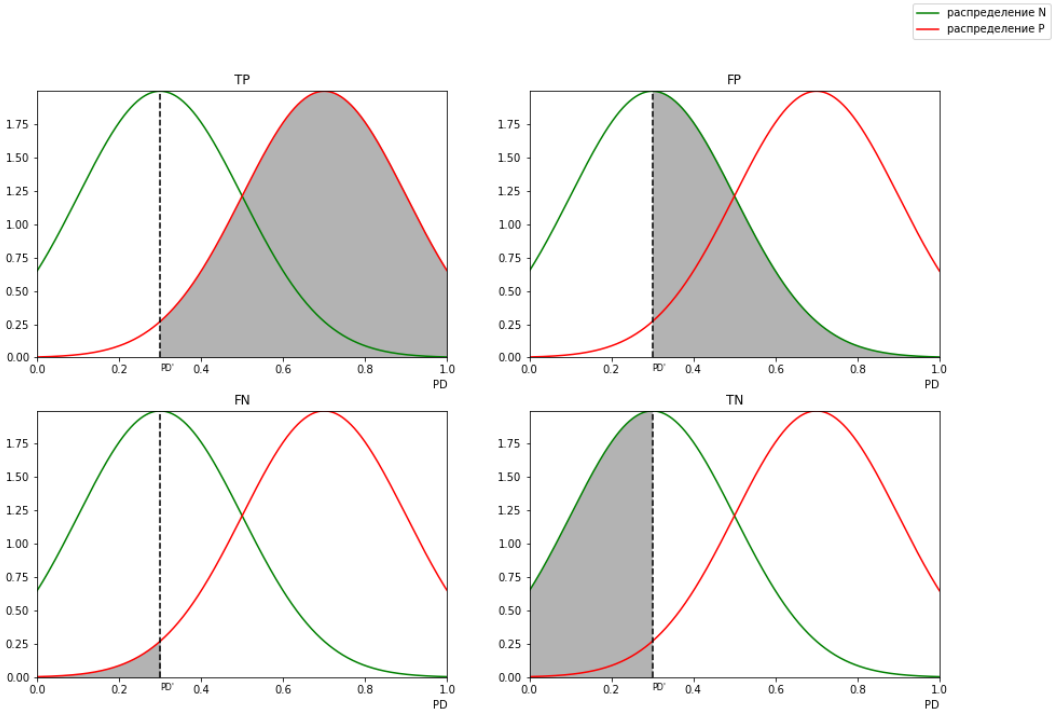

Рисунок Д.1 – Показатели классификации в строгой модели
Примечание – Источник: собственная разработка.

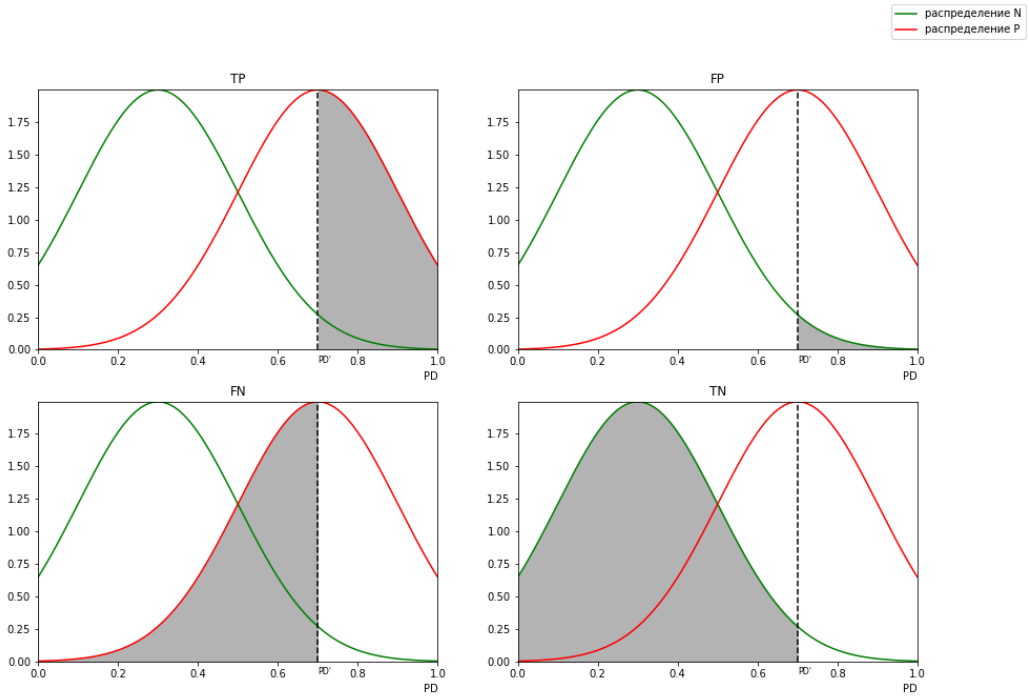

Рисунок Д.2 – Показатели классификации в лояльной модели
Примечание – Источник: собственная разработка.

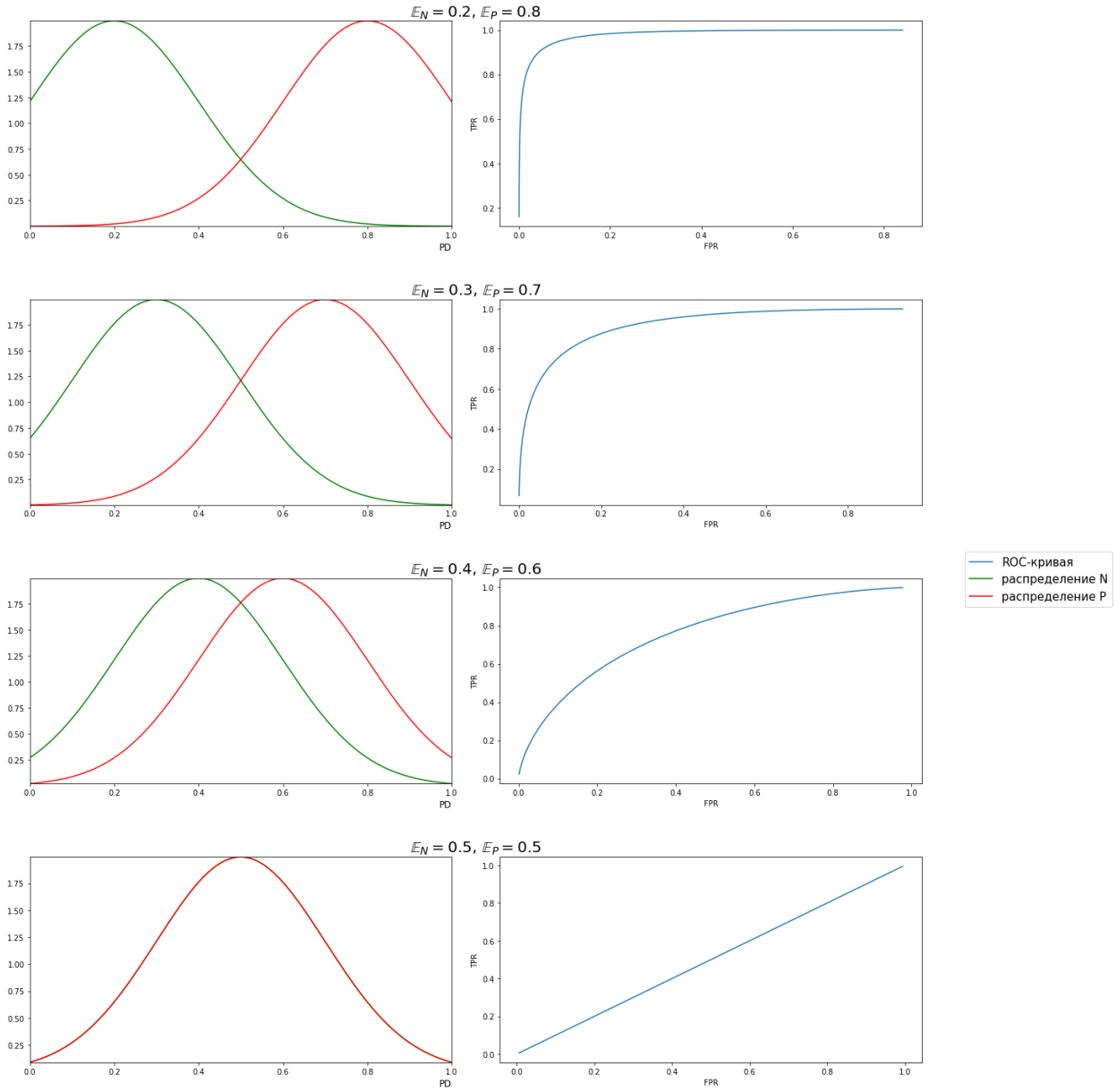

Рисунок Д.3 – Взаимосвязь различий распределения и ROC кривой

Примечание – Источник: собственная разработка.

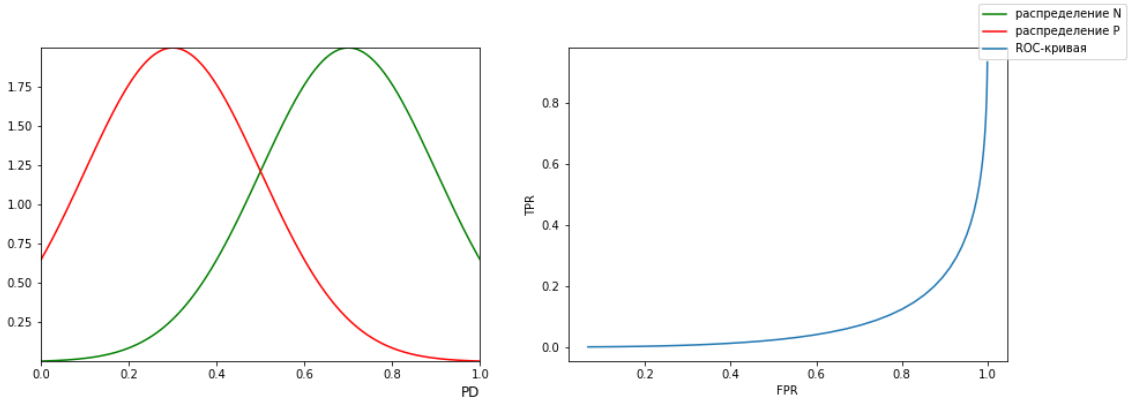

Рисунок Д.4 – ROC кривая в случае обмена распределений

Примечание – Источник: собственная разработка.

ПРИЛОЖЕНИЕ Е
ROC анализ для номинативной переменной

Таблица Е.1 – Процесс изменения TPR и FPR номинативного предиктора

l	Решение	FP_l	TP_l	FPR_l	TPR_l
m	$A_N = \{t_1, t_2, \dots, t_m\}$, $A_P = \emptyset$	0	0	0	0
m-1	$A_N = \{t_1, \dots, t_{m-1}\}$, $A_P = \{t_m\}$	n_m^N	n_m^P	$\frac{n_m^N}{N'}$	$\frac{n_m^P}{P'}$
m-2	$A_N = \{t_1, \dots, t_{m-2}\}$, $A_P = \{t_{m-1}, t_m\}$	$n_m^N + n_{m-1}^N$	$n_m^P + n_{m-1}^P$	$\frac{n_m^N + n_{m-1}^N}{N'}$	$\frac{n_m^P + n_{m-1}^P}{P'}$
…	…	…	…	…	…
u	$A_N = \{t_1, \dots, t_u\}$, $A_P = \{t_{u+1}, \dots, t_m\}$	$\sum_{k=u+1}^{m} n_k^N$	$\sum_{k=u+1}^{m} n_k^P$	$\frac{\sum_{k=u+1}^{m} n_k^N}{N'}$	$\frac{\sum_{k=u+1}^{m} n_k^P}{P'}$
u-1	$A_N = \{t_1, \dots, t_{u-1}\}$, $A_P = \{t_u, \dots, t_m\}$	$\sum_{k=u}^{m} n_k^N$	$\sum_{k=u}^{m} n_k^P$	$\frac{\sum_{k=u}^{m} n_k^N}{N'}$	$\frac{\sum_{k=u}^{m} n_k^P}{P'}$
…	…	…	…	…	…
j+1	$A_N = \{t_1, \dots, t_{j+1}\}$, $A_P = \{t_{j+2}, \dots, t_m\}$	$\sum_{k=j+2}^{m} n_k^N$	$\sum_{k=j+2}^{m} n_k^P$	$\frac{\sum_{k=j+2}^{m} n_k^N}{N'}$	$\frac{\sum_{k=j+2}^{m} n_k^P}{P'}$
j	$A_N = \{t_1, \dots, t_j\}$, $A_P = \{t_{j+1}, \dots, t_m\}$	$\sum_{k=j+1}^{m} n_k^N$	$\sum_{k=j+1}^{m} n_k^P$	$\frac{\sum_{k=j+1}^{m} n_k^N}{N'}$	$\frac{\sum_{k=j+1}^{m} n_k^P}{P'}$
j-1	$A_N = \{t_1, \dots, t_{j-1}\}$, $A_P = \{t_j, \dots, t_m\}$	$\sum_{k=j}^{m} n_k^N$	$\sum_{k=j}^{m} n_k^P$	$\frac{\sum_{k=j}^{m} n_k^N}{N'}$	$\frac{\sum_{k=j}^{m} n_k^P}{P'}$
…	…	…	…	…	…
s	$A_N = \{t_1, \dots, t_s\}$, $A_P = \{t_{s+1}, \dots, t_m\}$	$\sum_{k=s+1}^{m} n_k^N$	$\sum_{k=s+1}^{m} n_k^P$	$\frac{\sum_{k=s+1}^{m} n_k^N}{N'}$	$\frac{\sum_{k=s+1}^{m} n_k^P}{P'}$
s-1	$A_N = \{t_1, \dots, t_{s-1}\}$, $A_P = \{t_s, \dots, t_m\}$	$\sum_{k=s}^{m} n_k^N$	$\sum_{k=s}^{m} n_k^P$	$\frac{\sum_{k=s}^{m} n_k^N}{N'}$	$\frac{\sum_{k=s}^{m} n_k^P}{P'}$
…	…	…	…	…	…
i+1	$A_N = \{t_1, \dots, t_{i+1}\}$, $A_P = \{t_{i+2}, \dots, t_m\}$	$\sum_{k=i+2}^{m} n_k^N$	$\sum_{k=i+2}^{m} n_k^P$	$\frac{\sum_{k=i+2}^{m} n_k^N}{N'}$	$\frac{\sum_{k=i+2}^{m} n_k^P}{P'}$
i	$A_N = \{t_1, \dots, t_i\}$, $A_P = \{t_{i+1}, \dots, t_m\}$	$\sum_{k=i+1}^{m} n_k^N$	$\sum_{k=i+1}^{m} n_k^P$	$\frac{\sum_{k=i+1}^{m} n_k^N}{N'}$	$\frac{\sum_{k=i+1}^{m} n_k^P}{P'}$
i-1	$A_N = \{t_1, \dots, t_{i-1}\}$, $A_P = \{t_i, \dots, t_m\}$	$\sum_{k=i}^{m} n_k^N$	$\sum_{k=i}^{m} n_k^P$	$\frac{\sum_{k=i}^{m} n_k^N}{N'}$	$\frac{\sum_{k=i}^{m} n_k^P}{P'}$
…	…	…	…	…	…
r	$A_N = \{t_1, \dots, t_r\}$, $A_P = \{t_{r+1}, \dots, t_m\}$	$\sum_{k=r+1}^{m} n_k^N$	$\sum_{k=r+1}^{m} n_k^P$	$\frac{\sum_{k=r+1}^{m} n_k^N}{N'}$	$\frac{\sum_{k=r+1}^{m} n_k^P}{P'}$

Продолжение таблицы Е.1

r-1	$A_N = \{t_1, \dots, t_{r-1}\}$, $A_P = \{t_r, \dots, t_m\}$	$\sum_{k=r}^{m} n_j^N$	$\sum_{k=r}^{m} n_k^P$	$\frac{\sum_{k=r}^{m} n_j^N}{N'}$	$\frac{\sum_{k=r}^{m} n_k^P}{P'}$
…	…	…	…	…	…
1	$A_N = \{t_1\}$, $A_P = \{t_2, \dots, t_m\}$	$\sum_{k=2}^{m} n_k^N$	$\sum_{k=2}^{m} n_k^P$	$\frac{\sum_{k=2}^{m} n_k^N}{N'}$	$\frac{\sum_{k=2}^{m} n_k^P}{P'}$
0	$A_N = \emptyset$ $A_P = \{t_1, \dots, t_m\}$	1	1	1	1

Примечания:

1. Источник: собственная разработка,
2. n_k^N – число наблюдений с отсутствующим в действительности признаком оклика,
3. n_k^P – число наблюдений с наличием в действительности признаком оклика;

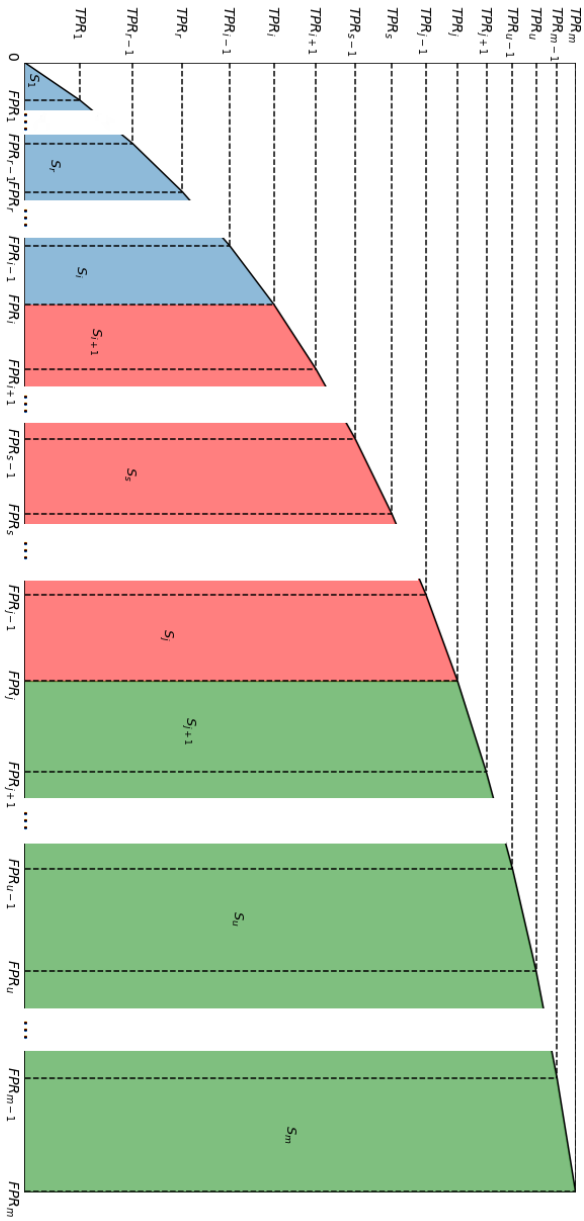

Рисунок Е.1 – Общий вид ROC кривой

Примечание – Источник: собственная разработка.

ПРИЛОЖЕНИЕ Ж
ROC анализ данных для модели

Таблица Ж.1 – Оценки классифицирующей способности отдельных предикторов

Показатель	AUC	KS	p-значение	Пропуски	Пропуски Нет дефолта%	Пропуски Дефолт%
1	2	3	4	5	6	7
Автомобиль год выпуска1	0,572	0,121	0,000	195030	96,972	3,028
Автомобиль год выпуска2	0,613	0,239	0,000	234900	96,935	3,065
Автомобиль год выпуска3	0,768	0,530	0,013	236396	96,924	3,076
Воинская служба	0,669	0,301	0,000	108124	98,884	1,116
Количество детей	0,511	0,023	0,092	155538	97,262	2,738
Количество иждивенцев	0,552	0,106	0,003	227405	96,927	3,073
Недвижимость	0,526	0,052	0,000	0	0,000	0,000
Образование	0,666	0,254	0,000	70250	99,162	0,838
Отношение к банку	0,505	0,010	0,480	0	0,000	0,000
Работа занимаемая должность	0,649	0,284	0,000	0	0,000	0,000
Работа последнее место стаж лет	0,500	0,001	1,000	217748	97,149	2,851
Работа уровень дохода BYR	0,507	0,018	1,000	221099	96,816	3,184
Семейное положение	0,653	0,216	0,000	63143	99,012	0,988
Собственная квартира	0,513	0,025	0,000	0	0,000	0,000
Собственный дом	0,511	0,021	0,003	0	0,000	0,000
Уголовная ответственность	0,516	0,031	0,000	0	0,000	0,000
Адрес проживания - Населенный пункт	0,538	0,072	0,000	0	0,000	0,000
Адрес проживания - Тип населенного пункта	0,621	0,222	0,000	80329	98,873	1,127
Гражданин РБ	0,593	0,184	0,000	68942	98,808	1,192
Дата рождения	0,560	0,095	0,000	0	0,000	0,000
Был ли хоть один договор прекращен досрочно	0,606	0,187	0,000	134681	97,903	2,097

Окончание таблицы Ж.1

1	2	3	4	5	6	7
Количество действующих договоров обеспечения	0,512	0,028	0,073	173674	97,070	2,930
Количество действующих кредитных договоров	0,559	0,072	0,000	145639	97,743	2,257
Количество запросов в КБ за последние 30 дней	0,570	0,109	0,000	132363	97,989	2,011
Количество фактов просрочки по основному долгу	0,650	0,242	0,000	168714	97,274	2,726
Максимальное количество дней просрочки	0,652	0,257	0,000	173510	97,238	2,762
Максимальный срок, на который заключался договор, в годах	0,539	0,083	0,000	135529	97,886	2,114
Наличие кредитной истории	0,600	0,201	0,000	131648	97,998	2,002
Общее количество запросов в КБ	0,537	0,074	0,000	136244	97,769	2,231
Сумма кредитных лимитов	0,544	0,078	0,000	153215	97,637	2,363
Сумма договора	0,651	0,275	0,000	20133	98,425	1,575
Количество потребляемых банковских продуктов	0,590	0,168	0,000	117790	97,925	2,075
Пол	0,542	0,084	0,000	0	0,000	0,000
Социальная группа	0,615	0,210	0,000	89220	98,525	1,475
Код подразделения	0,689	0,304	0,000	0	0,000	0,000
Автомобиль год выпуска	0,571	0,119	0,000	194937	96,974	3,026
Число авто	0,508	0,015	0,100	0	0,000	0,000
Есть авто	0,507	0,015	0,100	0	0,000	0,000
Срок кредита в днях	0,635	0,283	0,000	1	100,000	0,000
Ежедневный платеж	0,538	0,083	0,000	12330	97,405	2,595
Столица	0,522	0,044	0,000	0	0,000	0,000
Областной центр	0,529	0,059	0,000	0	0,000	0,000

Примечание – Источник: собственная разработка.

Таблица Ж.2 – Оценки классифицирующей численных предикторов и их производных

Показатель	AUC	KS	p-значение	Пропуски	Пропуски Нет дефолта%	Пропуски Дефолт%
1	2	3	4	5	6	7
Автомобиль год выпуска1	0,572	0,121	0,000	195030	96,972	3,028
Автомобиль год выпуска1 (бинарный)	0,507	0,014	0,134	0	0,000	0,000
Автомобиль год выпуска1 (пропуск => 0)	0,501	0,015	0,084	0	0,000	0,000
Автомобиль год выпуска1 (разбивка по KS)	0,520	0,029	0,000	195030	96,972	3,028
Автомобиль год выпуска2	0,613	0,239	0,000	234900	96,935	3,065
Автомобиль год выпуска2 (бинарный)	0,502	0,005	0,999	0	0,000	0,000
Автомобиль год выпуска2 (пропуск => 0)	0,502	0,004	1,000	0	0,000	0,000
Автомобиль год выпуска2 (разбивка по KS)	0,502	0,005	0,999	234900	96,935	3,065
Автомобиль год выпуска3	0,768	0,530	0,013	236396	96,924	3,076
Автомобиль год выпуска3 (бинарный)	0,500	0,001	1,000	0	0,000	0,000
Автомобиль год выпуска3 (пропуск => 0)	0,500	0,000	1,000	0	0,000	0,000
Автомобиль год выпуска3 (разбивка по KS)	0,501	0,001	1,000	236396	96,924	3,076
Количество действующих кредитных договоров	0,559	0,072	0,000	145639	97,743	2,257

Продолжение таблицы Ж.2

1	2	3	4	5	6	7
Количество действующих кредитных договоров (бинарный)	0,585	0,170	0,000	0	0,000	0,000
Количество действующих кредитных договоров (пропуск => 0)	0,598	0,184	0,000	0	0,000	0,000
Количество действующих кредитных договоров (разбивка по KS)	0,592	0,170	0,000	145639	97,743	2,257
Количество запросов в КБ за последние 30 дней	0,570	0,109	0,000	132363	97,989	2,011
Количество запросов в КБ за последние 30 дней (бинарный)	0,600	0,200	0,000	0	0,000	0,000
Количество запросов в КБ за последние 30 дней (пропуск => 0)	0,580	0,150	0,000	0	0,000	0,000
Количество запросов в КБ за последние 30 дней (разбивка по KS)	0,612	0,200	0,000	132363	97,989	2,011
Количество иждивенцев	0,552	0,106	0,003	227405	96,927	3,073
Количество иждивенцев (бинарный)	0,501	0,002	1,000	0	0,000	0,000
Количество иждивенцев (пропуск => 0)	0,501	0,004	1,000	0	0,000	0,000
Количество иждивенцев (разбивка по KS)	0,502	0,003	1,000	227405	96,927	3,073

Продолжение таблицы Ж.2

1	2	3	4	5	6	7
Количество фактов просрочки по основному долгу	0,650	0,242	0,000	168714	97,274	2,726
Количество фактов просрочки по основному долгу(бинарный)	0,542	0,084	0,000	0	0,000	0,000
Количество фактов просрочки по основному долгу (пропуск => 0)	0,565	0,120	0,000	0	0,000	0,000
Количество фактов просрочки по основному долгу (разбивка по KS)	0,555	0,110	0,000	168714	97,274	2,726
Максимальное количество дней просрочки	0,652	0,257	0,000	173510	97,238	2,762
Максимальное количество дней просрочки (бинарный)	0,539	0,078	0,000	0	0,000	0,000
Максимальное количество дней просрочки (пропуск => 0)	0,566	0,126	0,000	0	0,000	0,000
Максимальное количество дней просрочки (разбивка по KS)	0,539	0,078	0,000	173510	97,238	2,762
Максимальный срок, на который заключался договор, в годах	0,539	0,083	0,000	135529	97,886	2,114
Максимальный срок, на который заключался договор, в годах(бинарный)	0,593	0,185	0,000	0	0,000	0,000

Продолжение таблицы Ж.2

1	2	3	4	5	6	7
Максимальный срок, на который заключался договор, в годах (пропуск => 0)	0,580	0,182	0,000	0	0,000	0,000
Максимальный срок, на который заключался договор, в годах (разбивка по KS)	0,603	0,185	0,000	135529	97,886	2,114
Общее количество запросов в КБ	0,537	0,074	0,000	136244	97,769	2,231
Общее количество запросов в КБ(бинарный)	0,582	0,164	0,000	0	0,000	0,000
Общее количество запросов в КБ(пропуск => 0)	0,586	0,151	0,000	0	0,000	0,000
Общее количество запросов в КБ(разбивка по KS)	0,590	0,164	0,000	136244	97,769	2,231
Срок кредита в днях	0,635	0,283	0,000	1	100,000	0,000
Срок кредита в днях(бинарный)	0,500	0,000	1,000	0	0,000	0,000
Срок кредита в днях (пропуск => 0)	0,635	0,283	0,000	0	0,000	0,000
Срок кредита в днях (разбивка по KS)	0,641	0,283	0,000	1	100,000	0,000
Сумма договора	0,651	0,275	0,000	20133	98,425	1,575
Сумма договора (бинарный)	0,521	0,043	0,000	0	0,000	0,000
Сумма договора (пропуск => 0)	0,653	0,286	0,000	0	0,000	0,000
Сумма договора (разбивка по KS)	0,631	0,262	0,000	20133	98,425	1,575

Окончание таблицы Ж.2

1	2	3	4	5	6	7
Сумма кредитных лимитов	0,544	0,078	0,000	153215	97,637	2,363
Сумма кредитных лимитов (бинарный)	0,578	0,155	0,000	0	0,000	0,000
Сумма кредитных лимитов (пропуск => 0)	0,588	0,173	0,000	0	0,000	0,000
Сумма кредитных лимитов (разбивка по KS)	0,584	0,166	0,000	153215	97,637	2,363

Примечание – Источник: собственная разработка.

Таблица Ж.3 – Оценки классифицирующей способности показателей вошедших на этап моделирования

Показатель	AUC	KS	p-значение	Пропуски	Пропуски Нет дефолта%	Пропуски Дефолт%
1	2	3	4	5	6	7
Воинская служба	0,651	0,210	0,000	0	0	0
Недвижимость	0,530	0,059	0,000	0	0	0
Образование	0,671	0,259	0,000	0	0	0
Работа занимаемая должность	0,653	0,294	0,000	0	0	0
Семейное положение	0,661	0,224	0,000	0	0	0
Собственная квартира	0,514	0,029	0,000	0	0	0
Собственный дом	0,511	0,021	0,005	0	0	0
Уголовная ответственность	0,516	0,033	0,000	0	0	0
Адрес проживания - Тип населенного пункта	0,633	0,249	0,000	0	0	0
Гражданин РБ	0,604	0,207	0,000	0	0	0
Дата рождения	0,566	0,101	0,000	0	0	0

Продолжение таблицы Ж.3

1	2	3	4	5	6	7
Был ли хоть один договор прекращен досрочно	0,619	0,213	0,000	0	0	0
Наличие кредитной истории	0,614	0,228	0,000	0	0	0
Сумма договора	0,651	0,275	0,000	0	0	0
Количество потребляемых банковских продуктов	0,596	0,176	0,000	0	0	0
Пол	0,543	0,085	0,000	0	0	0
Социальная группа	0,628	0,237	0,000	0	0	0
Код подразделения	0,705	0,337	0,000	0	0	0
Количество действующих кредитных договоров	0,602	0,196	0,000	0	0	0
Количество фактов просрочки по основному долгу	0,568	0,126	0,000	0	0	0
Максимальное количество дней просрочки	0,568	0,130	0,000	0	0	0
Сумма кредитных лимитов	0,592	0,184	0,000	0	0	0
Автомобиль год выпуска1	0,516	0,032	0,000	0	0	0
Количество запросов в КБ за последние 30 дней	0,625	0,227	0,000	0	0	0
Максимальный срок, на который заключался договор, в годах	0,615	0,211	0,000	0	0	0

Окончание таблицы Ж.3

1	2	3	4	5	6	7
Общее количество запросов в КБ	0,602	0,192	0,000	0	0	0
Срок кредита в днях	0,661	0,322	0,000	0	0	0

Примечание – Источник: собственная разработка.

ПРИЛОЖЕНИЕ И
Алгоритм оценки параметров

```
import torch.nn as nn
class ResultNet(nn.Module):
   '''Нейронная сеть в общем виде'''
   def __init__(self, neur_counts):
      super(ResultNet, self).__init__()
      # предполагаяется что выходной нейрон
      # всего один и пользователю не надо
      # его объявлять
      neur_counts = neur_counts.copy() + [1]
      # динамическое
      # добавление слоев
      layers = OrderedDict()
      for i in range(len(neur_counts) - 1):
         layers[str(i)] = (
            nn.Linear(
               neur_counts[i],
               neur_counts[i+1]
            )
         )
         layers[str(i)].weight = \
         nn.Parameter(
            torch.rand(
               layers[str(i)].weight.size()
            )
         )
         layers[str(i)].bias = \
         nn.Parameter(
            torch.rand(
               layers[str(i)].bias.size()
            )
         )
      self.layers = nn.Sequential(layers)

   def forward(self, x):
      for layer in self.layers[:-1]:
         x = F.relu(layer(x))

      x = torch.sigmoid(self.layers[-1](x))
      return x
```

Листинг И.1 – Класс описывающий модель использованную в работе

Примечания

1. Источник: собственная разработка на основе [16 с. 47],
2. Для корректной работы функций предварительно необходимо установить библиотеку numpy.

```
from copy import deepcopy

import numpy as np

class model_trainer():
    '''Класс реализует алгоритм обучения сети'''
    def __init__(
        self, model, optimizer,
        loss_fn, lr_scheduler = None
    ):
        # inputs:
        # model - модель которая подлежит обучению
        # optimizer - оптимизатор, который педполагается использовать
        # loss_fn - функция потерь
        # lr_shceduler - планировщих learning rate
        self.model = model
        self.optimizer = optimizer
        self.loss_fn = loss_fn

        self.train_loss = np.array([])
        self.test_loss = np.array([])

        # тут храним лучшую модель
        # из полученных если ошиентироваться
        # на ошибку на тестовых данных
        self.best_model = deepcopy(model)
        self.best_epoch = 0

        self.lr_scheduler = lr_scheduler


    def append_loses(self, train_loader, test_loader):
        '''Добавление ошибок рассчитанных
        на тренировочном и тестовом загрузчиках'''
        self.train_loss = np.append(
            self.train_loss,
            get_loss_value(
                self.loss_fn,
                self.model,
                train_loader
            )
        )
        self.test_loss = np.append(
            self.test_loss,
            get_loss_value(
                self.loss_fn,
                self.model,
                test_loader
            )
        )
```

```
def fit(self, train_loader, test_loader,
        epochs = 20, check_epoch = 1):
    '"Провести тренировку модели"'
    # inputs:
    # train_loader - загрузчик тренировочных данных
    # test_loader - загрузчик тестовых данных
    # epochs - число эпох для обучения алгоритма
    # check_epoces - число эпох после чего алгорим
    #               может быть оставлен и ведется
    #               регистрация лучшей модели

    # получаем начальную ошибку до
    # какого либо смещения весов
    self.append_loses(train_loader, test_loader)

    for epoch in range(epochs):

        self.model.train()
        for batch in train_loader:
            self.optimizer.zero_grad()
            inputs, targets = batch
            output = self.model(inputs)
            loss = self.loss_fn(output, targets)
            loss.backward()
            self.optimizer.step()
        if self.lr_scheduler:
            self.lr_scheduler.step()

        # вычисление ошибок на тестовой и
        # обучающих выборках на каждой эпохе
        self.append_loses(train_loader, test_loader)


        # работа с критериями остановки и
        # сохранения модели
        if epoch > check_epoch:
            # проверяем критерий остановки
            if self.test_loss[-1] > self.test_loss[-(check_epoch-1)]:
                return

            # в том случае, если последняя полученная
            # ошибка на тестовых данных наименьшая
            # то нужно запомнить модель как наилучшую
            if self.test_loss[-1] == np.min(self.test_loss):
                self.best_model = deepcopy(self.model)
                self.best_epoch = epoch
```

Листинг И.2 – Класс описывающий алгоритм обучения модели

Примечания:

1. Источник: собственная разработка на основе [16 с.47],
2. Для корректной работы функций предварительно необходимо установить библиотеку numpy.

```
from torch.utils.data import Dataset

class My_data_set(Dataset):
    '''Набор данных'''
    def __init__(self, X, Y):
        self.X = X
        self.Y = Y

    def __len__(self):
        return self.X.shape[0]

    def __getitem__(self, idx):
        return [self.X[idx,:], self.Y[idx, :]]
```

Листинг И.3 – Класс набора данных для создания загручиков

Примечания:

1. Источник: собственная разработка,
2. Для корректной работы функций предварительно необходимо установить библиотеку pytorch.

```
train_data = My_data_set(
    torch.tensor(X_train.astype('float32')),
    torch.tensor(y_train.astype('float32'))
)
train_data_loader =\
torch.utils.data.DataLoader(
    train_data, batch_size=500
)


test_data = My_data_set(
    torch.tensor(X_test.astype('float32')),
    torch.tensor(y_test.astype('float32'))
)
test_data_loader =\
torch.utils.data.DataLoader(
    test_data, batch_size=500
)

# создание модели
torch.manual_seed(0)
model_1lay = ResultNet([X.shape[1], X.shape[1]])

# оптимизатор
optimizer = optim.Adam(
    model_1lay.parameters(),
    lr = 0.01
)

lay1_trainer = model_trainer(
    model_1lay, optimizer, loss_fn
)

lay1_trainer.fit(
```

```
    train_data_loader,
    test_data_loader
)
```

Листинг И.4 – Первый запуск модели на обучение
Примечание – Источник: собственная разработка.

ПРИЛОЖЕНИЕ К
Исследование обучения по эпохам

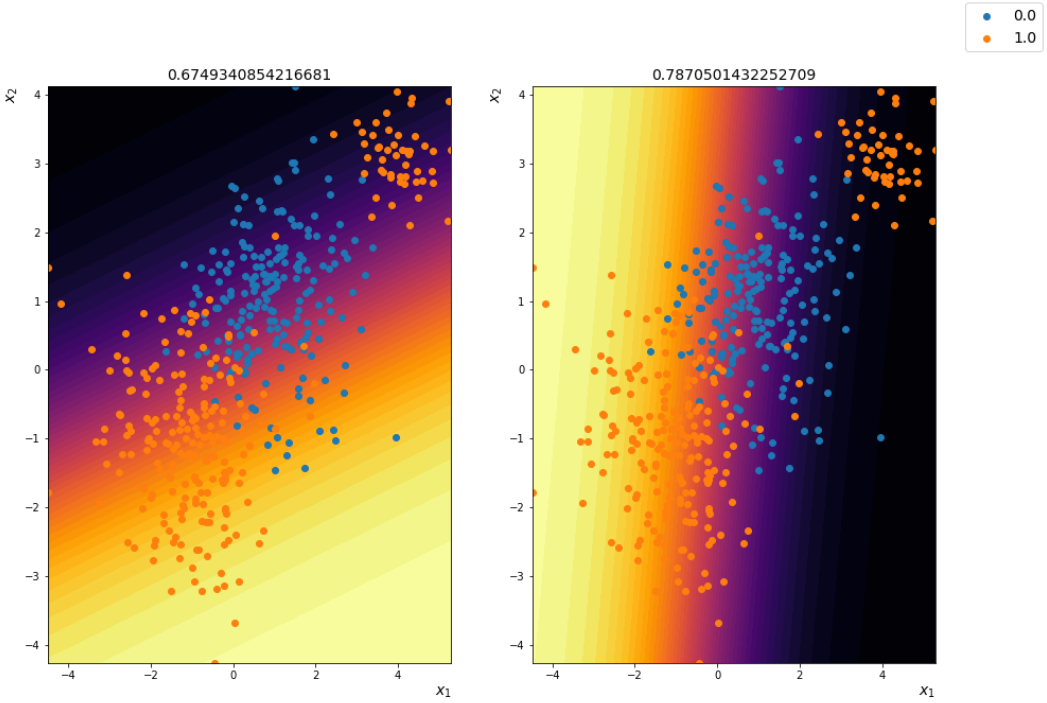

Рисунок К.1 – Логит для полных данных
Примечание – Источник: собственная разработка.

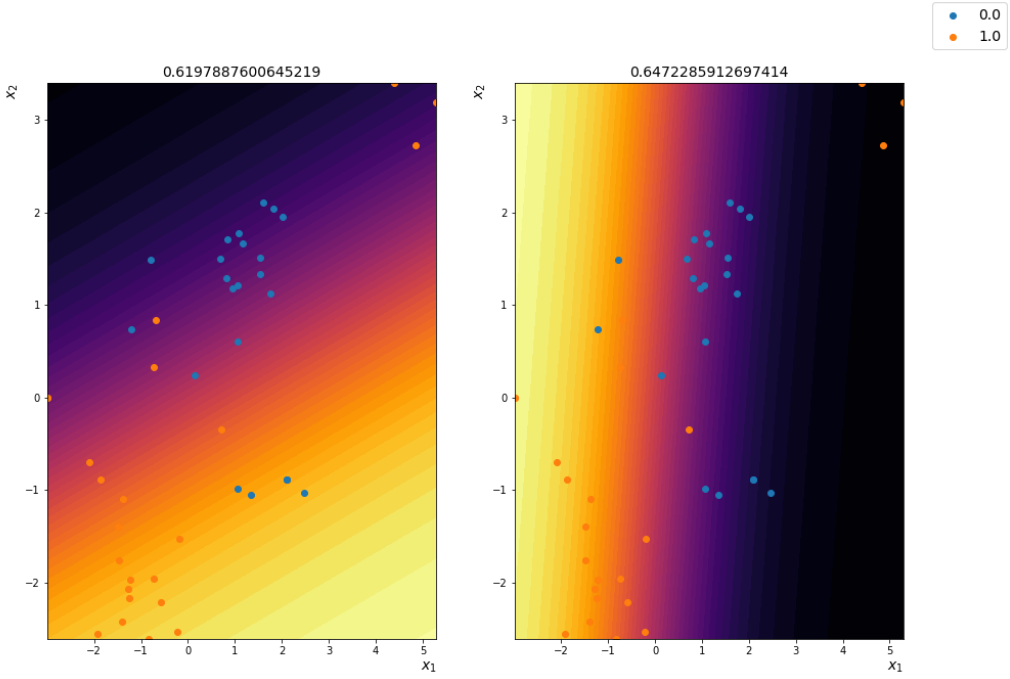

Рисунок К.2 – Логит для случайного батча
Примечание – Источник: собственная разработка.

ПРИЛОЖЕНИЕ Л
Характеристики модели при обучении

Таблица Л.1 – AUC модели на этапах обучения

Число эпох	«gamma»	AUC тестовых данных	AUC тренировочных данных
20	-	0,74708	0,74528
50	-	0,77535	0,77506
100	-	0,78039	0,77922
300	0,95	0,75631	0.75451
300	0,99	0,79879	0,80248
1000	0.99	0,79978	0,80382

Примечание – Источник: собственная разработка.

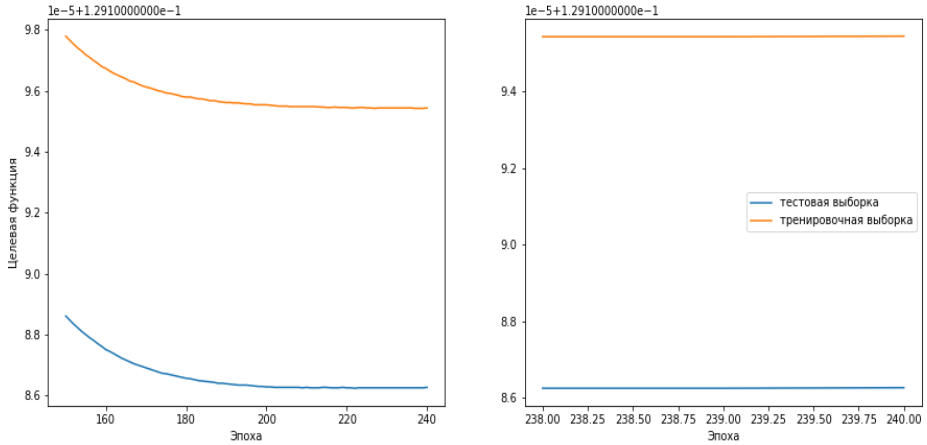

Рисунок Л.1 – Быстрый выход на плато
Примечание – Источник: собственная разработка.

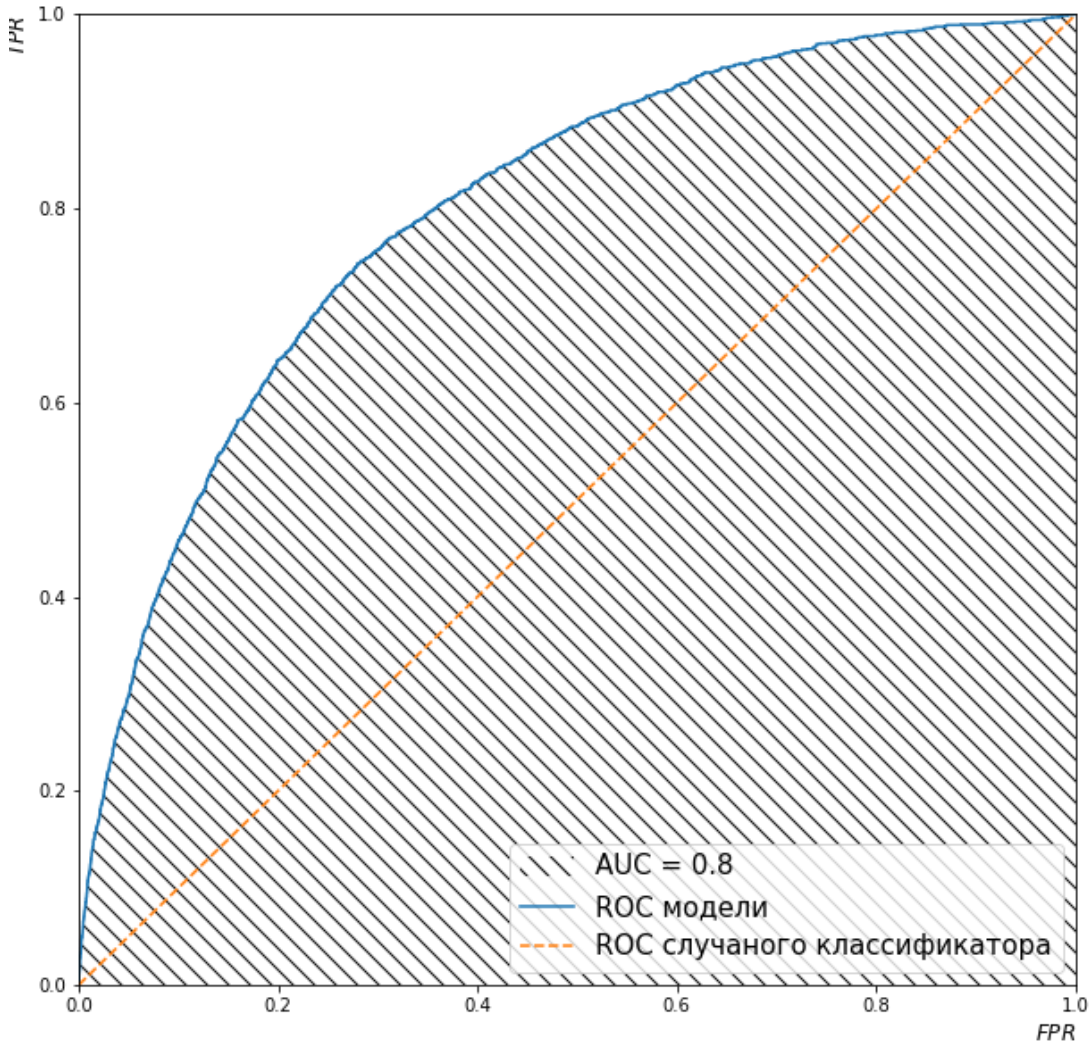

Рисунок Л.2 – ROC кривая финальной модели

Примечание – Источник: собственная разработка.