

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ
РОССИЙСКОЙ ФЕДЕРАЦИИ»
(ФИНАНСОВЫЙ УНИВЕРСИТЕТ)

Департамент анализа данных
и машинного обучения

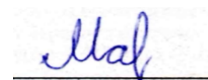
Дисциплина: «Технологии анализа данных и машинного обучения»
Направление подготовки: «Прикладная информатика»
Профиль: «Анализ данных и принятие решений в экономике и финансах»
Факультет информационных технологий и анализа больших данных
Форма обучения очная
Учебный 2022/2023 год, 6 семестр

Курсовая работа на тему:
«Искусственные нейронные сети для классификации держателей кредитных карт по
вероятности дефолта»

Выполнил:

студент группы ПИ20-3

Маркович В. С.



Научный руководитель:

доцент к.т.н. Кублик Е. И.

Москва 2023

СОДЕРЖАНИЕ

I. ВВЕДЕНИЕ.....	3
II. НАБОР ДАННЫХ.....	4
III. АНАЛИЗ ДАННЫХ И КОМПЛЕКСНАЯ ПРЕДОБРАБОТКА ДАННЫХ	7
<i>Пример обработки числового признака.....</i>	<i>8</i>
<i>Пример преобразования категориального признака</i>	<i>10</i>
<i>Пример заполнения пустых значений для категориального поля</i>	<i>11</i>
<i>Целевая переменная</i>	<i>11</i>
IV. РАЗДЕЛЕНИЕ НАБОРА ДАННЫХ	13
<i>Определение размеров обучающей, тестовой и валидационной выборок..</i>	<i>14</i>
<i>Разделение набора данных на три выборки</i>	<i>14</i>
<i>Создание объектов «Dataset»</i>	<i>15</i>
<i>Объекты класса DataLoader</i>	<i>16</i>
V. ОПРЕДЕЛЕНИЕ ПРИНЦИПИАЛЬНОЙ АРХИТЕКТУРЫ НЕЙРОСЕТЕВОЙ МОДЕЛИ ГЛУБОКОГО ОБУЧЕНИЯ.....	18
<i>Определение констант.....</i>	<i>18</i>
<i>Определение архитектуры модели</i>	<i>19</i>
<i>Настройка дополнительных параметров обучения.....</i>	<i>20</i>
<i>Цикл обучения</i>	<i>21</i>
VI. ОЦЕНКА КАЧЕСТВА МОДЕЛИ	23
<i>Графики метрик.....</i>	<i>23</i>
<i>Пример интерпретации выходных данных модели.....</i>	<i>26</i>
VII. ПОСТРОЕНИЕ АЛЬТЕРНАТИВНЫХ МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ.....	28
VIII. ЗАКЛЮЧЕНИЕ.....	30
IX. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	31

I. ВВЕДЕНИЕ

В современном мире финансовый риск играет огромную роль в деятельности кредитных организаций. Каждый день банки и другие кредитные учреждения вынуждены принимать решения, связанные с выдачей кредитов, и неизбежно сталкиваются с риском дефолта заёмщиков. Определение вероятности дефолта является одним из важных аспектов риск-менеджмента.

В настоящее время искусственные нейронные сети являются одним из наиболее перспективных инструментов для анализа данных и принятия решений в финансовой сфере. Искусственные нейронные сети — это алгоритмы машинного обучения, которые позволяют компьютерам анализировать и обрабатывать данные, обучаясь на примерах. Они имеют широкий спектр применения в финансовом риск-менеджменте, включая классификацию и прогнозирование вероятности дефолта заёмщика.

Цель данной курсовой работы - исследовать возможность применения искусственных нейронных сетей для классификации держателей кредитных карт по вероятности дефолта. В работе будет рассмотрен пример использования нейронных сетей для определения вероятности дефолта заёмщиков и выделения групп риска. Будут проанализированы факторы, влияющие на вероятность дефолта, и предложен метод оценки риска на основе нейронных сетей.

Таким образом, использование искусственных нейронных сетей в финансовом риск-менеджменте может улучшить финансовые показатели компаний, снизить риски и повысить точность оценки вероятности дефолта. Однако, необходимо учитывать сложность и чувствительность модели к выборке данных, а также проблемы интерпретируемости результата, которые также будут рассмотрены в работе.

II. НАБОР ДАННЫХ

Мною был выбран набор данных с интернет ресурса kaggle.com. Ссылка на набор данных: <https://www.kaggle.com/datasets/marcbuji/loan-default-prediction?datasetId=2410810&sortBy=dateRun&tab=profile>

Набор данных имеет 87500 строк и 30 признаков. Каждая строка описывает определённого заёмщика. Из 30 признаков сразу хочется выделить 2: «ID» – уникальный идентификатор, который станет индексом в таблице, а также «Default» - целевая переменная. Далее попробуем поподробнее разобраться с тем, что означает каждый признак:

- 1) ID - Уникальный идентификатор держателя кредитной карты
- 2) Asst_Reg - Совокупная стоимость всех активов зарегистрированных за заёмщиком
- 3) GGGrade: Оценка группы выдачи кредита
- 4) Experience - Общий стаж работы заемщика в годах
- 5) Validation - Статус проверки заемщика
- 6) Yearly Income - Общий годовой доход заемщика
- 7) Home Status - Статус проживания заемщика
- 8) Unpaid 2 years - Количество раз, когда заемщик допустил просрочку в течение последних двух лет
- 9) Already Defaulted - Количество других кредитов, по которым заемщик допустил просрочку
- 10) Designation - Должность заемщика
- 11) Debt to Income - Соотношение долга к доходу
- 12) Postal Code - Почтовый индекс заемщика
- 13) Lend Amount - Общая сумма кредита, выданная заемщику
- 14) Deprecatory Records - Запись, которая может рассматриваться кредиторами как негативная, так как она указывает на риск и ухудшает возможность получения кредита или других услуг
- 15) Interest Charged - Процентная ставка по общей сумме кредита
- 16) Usage Rate - Обработочные сборы на сумму кредита

- 17) Inquiries - Запросы за последние 6 месяцев
- 18) Present Balance - Текущий баланс на счету заемщика
- 19) Gross Collection - Общая сумма, подлежащая выплате в рамках урегулирования или судебного решения по искам, исключая любые расходы
- 20) Sub GGGrade - Оценка подгруппы группы выдачи кредита
- 21) File Status - Статус заявки на кредит
- 22) State: Штат, к которому принадлежит заемщик
- 23) Account Open - Общее количество открытых счетов на имя заемщика
- 24) Total Unpaid CL - Неоплаченные задолженности по всем другим кредитам
- 25) Duration - Срок, на который выдан кредит заемщику
- 26) Unpaid Amount - Неоплаченный баланс по кредитной карте
- 27) Reason - Причина подачи заявки на кредит
- 28) Claim Type - Тип заявки заемщика среди всех типов заявок. (I - Индивидуальный счет, J - Совместный счет)
- 29) Due Fee - Штрафные санкции за задержку выплаты по кредиту
- 30) Default - Целевая переменная - наличие/отсутствие дефолта (1 - наличие/0-отсутствие)

Так как целевая переменная принимает значение 0 или 1, то это задача бинарной классификации. Для оценки качества модели на задаче бинарной классификации можно использовать различные метрики. Одним из наиболее распространенных является ассигасу (точность), которая показывает долю правильных ответов модели. Однако, ассигасу может быть не очень информативной метрикой, если классы несбалансированные, т. е. один из классов преобладает над другим. Для более удобной интерпретации результата работы модели целевую переменную стоит преобразовать таким образом, чтобы положительный класс имел значение целевой переменной 1, а отрицательный – 0. Соответственно, создан искусственный признак «Loan», который показывает платёжеспособность заёмщика (можно ли ему выдавать кредит), где 1 –

платёжеспособен, а 0 – неплатёжеспособен. Признак «Loan» становится целевой переменной вместо «Default».

В таком случае лучше использовать другие метрики, такие как precision (точность), recall (полнота), F1-score (F-мера). Precision показывает, какую долю из предсказанных моделью положительных классов действительно являются положительными. Recall показывает, какую долю из истинных положительных классов модель предсказала правильно. F1-score является гармоническим средним между precision и recall и показывает общую точность модели. Для моей задачи важно точно предсказывать отрицательный класс (неплатёжеспособность), то есть наилучшей метрикой для оценки качества модели будет точность (precision) и конечно же F1-score (для учёта несбалансированности классов, так как всё-таки гораздо больше платёжеспособных заёмщиков).

Точность (precision) — это метрика, которая оценивает долю верно предсказанных положительных классов (1) относительно всех положительных классов, которые модель предсказала. Задачей модели является точное предсказание негативного класса (0), поэтому точность — это метрика, которая может помочь мне понять, насколько точно модель предсказывает отрицательный класс.

III. АНАЛИЗ ДАННЫХ И КОМПЛЕКСНАЯ ПРЕДОБРАБОТКА ДАННЫХ

Предобработка данных - это важный этап в анализе данных, который включает в себя различные методы, направленные на улучшение качества данных и подготовку их к анализу. Предобработка данных включает в себя обработку каждого признака отдельно, анализ распределения данных, избавление от выбросов, заполнение пустых значений, преобразование категориальных признаков в числовые и многое другое.

Для числовых данных одним из наиболее популярных методов визуализации является гистограмма. Гистограмма позволяет анализировать распределение данных, определять наличие выбросов и идентифицировать потенциальные проблемы с данными. Ящик с усами также является важным инструментом визуализации данных, который помогает анализировать распределение данных, определять наличие выбросов и оценивать разброс данных. После обработки данных, гистограмма и ящик с усами позволяют сравнить распределение данных до и после обработки.

Для категориальных признаков, наиболее распространенными методами являются barplot и преобразование при помощи Encoder'ов. Barplot на основе таблицы `.value_counts()` позволяет анализировать распределение категориальных данных, идентифицировать наиболее часто встречающиеся значения и определять потенциальные проблемы с данными. Однако, для анализа категориальных данных необходимо преобразовать их в числовые. Для этого можно использовать различные методы, такие как `OrdinalEncoder` или `LabelEncoder`. `OrdinalEncoder` преобразует категории в числовые значения с сохранением порядка, а `LabelEncoder` просто заменяет каждую категорию уникальным числовым значением. Выбор между этими методами зависит от типа данных и характеристик признаков.

Ещё одним важным шагом при предобработке данных является заполнение пропущенных значений. Пропущенные значения могут возникать по разным причинам, например, из-за ошибок при вводе данных или отсутствия информации. Если пропущенных значений мало, то их можно просто удалить,

однако, если их много, то это может привести к потере большого количества данных. В данной работе для числовых признаков использовался пользовательский метод заполнения пустых значений (функция `fill_missings`). Сперва, стоит обозначить, что выбросы сначала заменялись на NaN-значения, а затем уже все NaN-значения (изначально отсутствующие + выбросы) заменялись при помощи функции `fill_missings()` по следующей логике: сначала выделялось распределение текущего признака для людей из группы с тем же “GGGrade”, это обусловлено тем, что у клиентов с одной и той же оценкой (GGGrade) более схожи другие показатели. Затем вычислялось медианное значение и 25% отклонения от медианного в этой группе. Далее генерировалось нормальное распределение начиная от $0,75 \times \text{медианное}$ до $1,25 \times \text{медианное}$ и затем бралось одно случайное значение из этого распределения. Такой способ позволяет сохранить дисперсию данных и приблизить их к реальному распределению. Такой подход может помочь в сохранении более точной и корректной информации в данных, что, в свою очередь, может улучшить качество анализа. Отдельного внимания заслуживает признак “Designation” – должность заёмщика, так как это категориальный признак. Для данного признака мною было принято использовать случаю должность из трёх наиболее распространённых должностей среди заёмщиков из той же оценки группы (GGGrade).

Пример обработки числового признака

В качестве примера рассмотрим признак «Debt_To_Income» - отношение долга к доходу.

Гистограмма на неочищенных данных

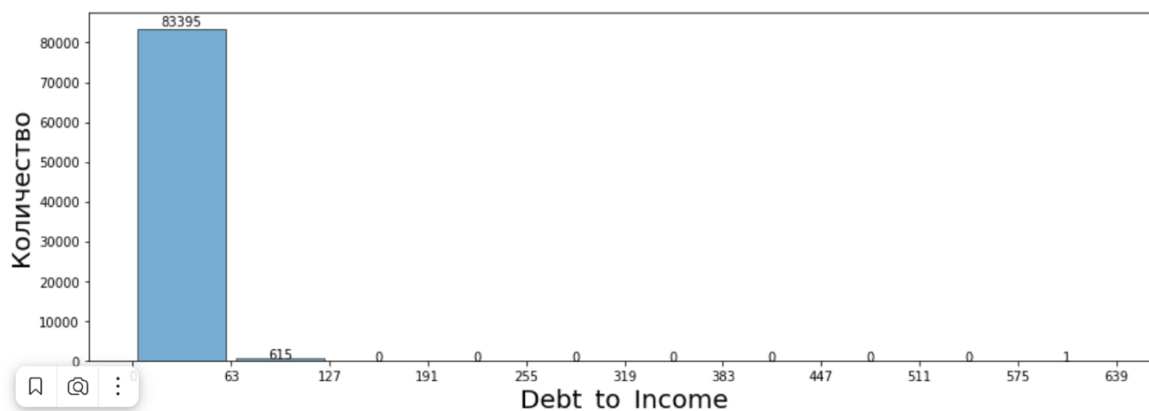


Рисунок 1 – гистограмма распределения значений отношений долга к доходу на неочищенных данных



Рисунок 2 – ящик с усами распределения значений отношений долга к доходу на неочищенных данных

Из двух данных графиков чётко видны выбросы в небольшом количестве.

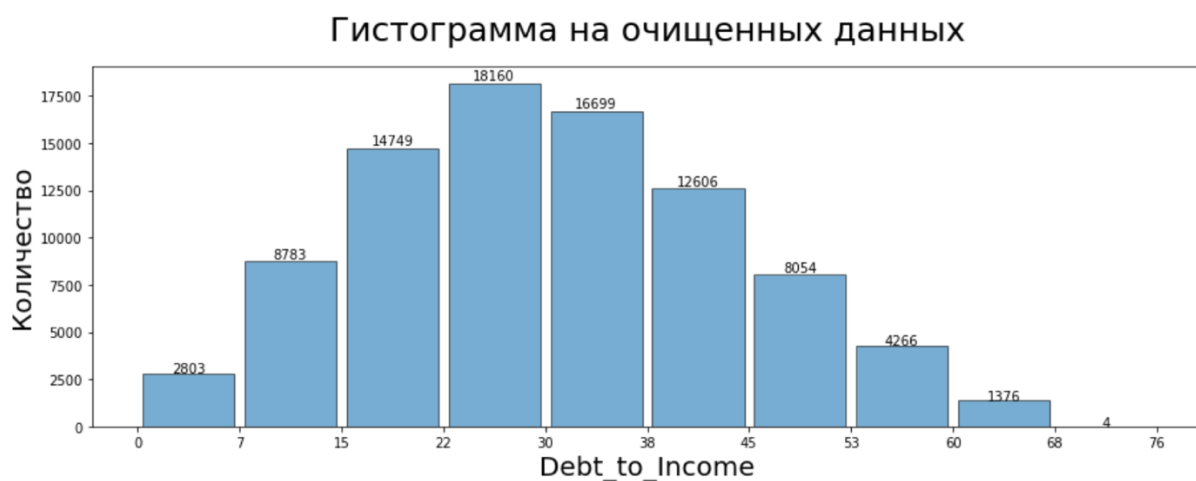


Рисунок 3 – графики распределения значений отношений долга к доходу после очистки данных

После очистки данных распределение стало похоже на нормальное, практически не наблюдается выбросов, но даже те, что кажутся выбросами вполне таковыми не являются, так как отношение долга к доходу может быть в диапазоне 68–76%.

Пример преобразования категориального признака

В качестве примера рассмотрим признак «Validation» - статус проверки заёмщика. Данный признак не имеет пустых значений и принимает одной из трёх возможных значений: «Not Vfied» - не подтверждён, «Source Verified» - подтверждён источник, «Vfied» - полностью подтверждён (и источник и статус).

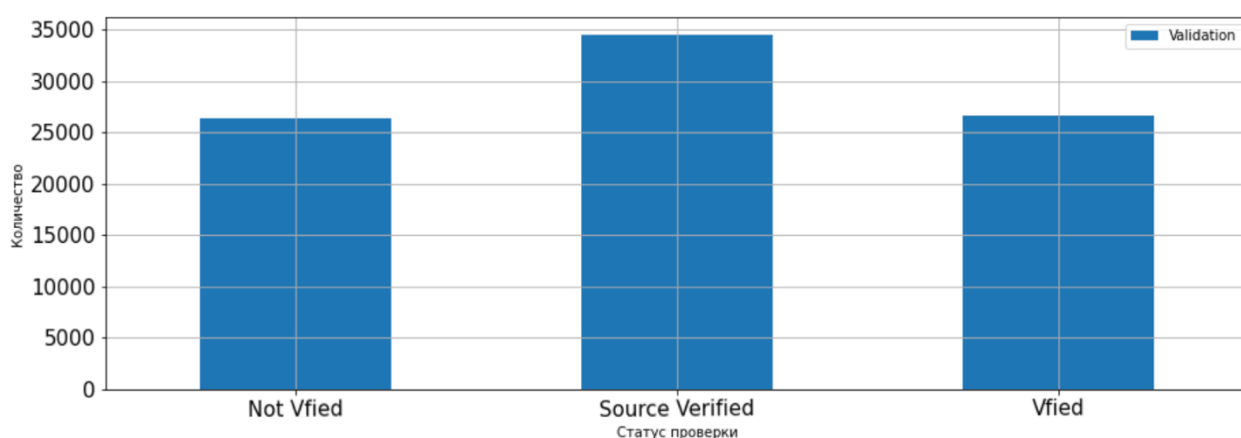


Рисунок 4 – график распределения значений поля «Статус проверки заёмщика»

В связи с тем, что значения данного поля имеют определённый порядок, то после замены значений на числовые порядок сохранится, то есть «Not Vfied» - 0, «Source Verified» - 1, «Vfied» - 2.

	Origin	Encoded
0	Vfied	2.00
1	Source Verified	1.00
2	Source Verified	1.00
3	Vfied	2.00
4	Source Verified	1.00
5	Source Verified	1.00
6	Source Verified	1.00
7	Source Verified	1.00
8	Vfied	2.00
9	Not Vfied	0.00

Рисунок 5 – пример первых 10 полей с заменой поля «Статус проверки заёмщика»

Пример заполнения пустых значений для категориального поля

Единственным категориальным полем, которое имеет пустые значения в моём наборе данных, является «Designation» - должность заёмщика. Данное поле имеет 1414 отсутствующих значений.

```
1 check_missings('Designation')
```

Поле Designation имеет 1414 пропусков

Рисунок 6 – подсчёт количества пустых значений поля «Designation»

Далее формируется вспомогательная таблица, которая будет содержать группу оценки заёмщика (1–6) и случайно выбранную профессию из трёх самых частых среди заёмщиков, имеющих такую же группу оценки.

Designation	
GGGrade	
0.00	School Teacher
1.00	School Teacher
2.00	Super Lead
3.00	Owner
4.00	School Teacher
5.00	Super Lead
6.00	School Teacher

Рисунок 7 – пример вспомогательной таблицы для заполнения признака «Designation»

Далее на основании такой таблицы для каждого заёмщика с пустым значением поля «Designation» в зависимости от его группы оценки выбирается должность. А уже далее, используя LabreEncoder все значения заменяются на соответствующие численные значения.

Целевая переменная

В оригинальном наборе данных присутствует поле «Default», означающее то, что объявил ли заёмщик себя неплатёжеспособным или нет. Объявление себя неплатёжеспособным означает потерю активов для финансовой организации. Поле «Default» принимает одно из двух значений: 0 или 1, где 0 – означает, что

заёмщик платёжеспособный (дефолт отсутствует), а 1 – что заёмщик объявил себя неплатёжеспособным (дефолт присутствует). В задаче, которую решает модель в этой работе, положительным классом являются заёмщики, которые платёжеспособны, то есть те, у которых поле «Default» принимает значение 0. Соответственно, для корректной интерпретации метрик качества модели введена новая целевая переменная «Loan», которая будет означать «платёжеспособен ли заёмщик» и принимать значение 1, если платёжеспособен и 0, если неплатёжеспособен. По сути, инвертируются значения поля «Default», а затем оно удаляется.

1	<code>clean_data.loc[:, 'Default'].value_counts().to_frame('Количество значений')</code>
Количество значений	
0	70988
1	16512
1	<code>clean_data['Loan'] = np.where(</code>
2	<code> clean_data.Default == 0,</code>
3	<code> 1,</code>
4	<code> 0</code>
5	<code>)</code>
6	<code>clean_data.drop(columns=['Default'], inplace=True)</code>

Рисунок 8 – преобразование целевой переменной.

После данных преобразований новой целевой переменной стало поле «Loan» означающее платёжеспособность заёмщика.

IV. РАЗДЕЛЕНИЕ НАБОРА ДАННЫХ

Разделение датасета на обучающую, тестовую и валидационную выборки - это одна из важных практик в машинном обучении. Это позволяет оценивать качество модели на реальных данных, не виденных моделью в процессе обучения.

Обучающая выборка используется для обучения модели. Модель подстраивается под данные, и на этом этапе параметры модели оптимизируются. Обучающая выборка должна содержать достаточно данных для обучения модели, и при этом не содержать избыточных данных, которые могут привести к переобучению.

Тестовая выборка используется для оценки качества модели после обучения. Она представляет собой набор данных, которые модель не видела при обучении, и она позволяет оценить, насколько модель хорошо обобщает данные.

Валидационная выборка используется для настройки гиперпараметров модели, таких как выбор оптимального количества эпох, скорости обучения и размера пакета. Она позволяет оценить, как настройки гиперпараметров влияют на качество модели, и выбрать оптимальные настройки для модели. Валидационная выборка должна быть достаточно большой, чтобы обеспечить статистическую значимость результатов, но при этом не должна быть слишком большой, чтобы не уменьшить размер обучающей выборки и не увеличить время обучения.

Использование валидационной выборки помогает следить за переобучением модели. Если модель переобучена, она будет хорошо работать на обучающей выборке, но плохо на тестовой выборке. Валидационная выборка позволяет оптимизировать гиперпараметры модели так, чтобы она хорошо работала на тестовой выборке и не переобучалась на обучающей выборке.

Переобучение происходит, когда модель очень хорошо работает на данных, которые она использовала для обучения, но работает плохо на новых данных. В этом случае, оценка производительности на тестовой выборке может быть завышена и не дать реального представления о том, как модель будет

работать на новых данных. Использование валидационной выборки позволяет нам оценить качество модели на данных, которые она не использовала для обучения, и выбрать наиболее оптимальные гиперпараметры.

Определение размеров обучающей, тестовой и валидационной выборок

Размер оригинального набора данных составляет 87500 точек данных. Для начала этот набор разбивается на тренировочный и тестовый в отношении 80/20. Тестовый набор данных имеет размер $87500 * 0.2 = 17500$ точек данных. Затем, тренировочный набор данных разбивается на обучающий и валидационный в отношении 90/10, тогда обучающая выборка имеет $87500 * 0.8 * 0.9 = 63000$ точек данных, а валидационная выборка имеет $87500 * 0.8 * 0.1 = 7000$ точек данных.

```
1 train_ratio = 0.8
2 validation_ratio = 0.1
3
4 train_amount = int(train_ratio*data.shape[0])
5 test_amount = data.shape[0] - train_amount
6 validation_amount = int(validation_ratio*train_amount)
7 train_amount -= validation_amount
8
9 print(f'Dataset size      = {data.shape[0]}')
10 print(f'Train size       = {train_amount}')
11 print(f'Validation size   = {validation_amount}')
12 print(f'Test size        = {test_amount}')
```

```
Dataset size      = 87500
Train size        = 63000
Validation size    = 7000
Test size         = 17500
```

Рисунок 9 – определение размеров выборок.

Разделение набора данных на три выборки

Существует несколько методов разделения выборки на тестовую, тренировочную и валидационную. Один из них - временной метод, при котором данные разбиваются по времени. Например, если у нас есть данные за 5 лет, то первые 4 года могут быть использованы для обучения, а последний год - для тестирования. Другой метод - это последовательное разбиение, при котором выборка делится на несколько подвыборок. При этом каждая подвыборка

используется в качестве тестовой выборки по очереди, а все остальные - для обучения. Третий метод - случайное разбиение, при котором выборка случайным образом разделяется на три непересекающиеся подвыборки.

Выбор метода разбиения зависит от особенностей данных и целей исследования. Например, если данные имеют временную составляющую и мы хотим оценить способность модели к прогнозированию будущих значений, то лучше использовать временной метод разбиения. Если данные равномерно распределены во времени, можно применить случайное разбиение. Если же мы работаем с данными, которые имеют определенный порядок, то лучше использовать последовательное разбиение.

В данной работе, поскольку нет особенностей, связанных с порядком или временным характером данных, использовался метод случайного разбиения. Этот метод дает возможность получить репрезентативные выборки для обучения, тестирования и валидации модели, а также минимизировать риск переобучения, что важно для бинарной классификации.

```
1 # Перемешиваем строки в оригинальном наборе данных случайным образом
2 data = data.sample(frac=1, random_state=5)
3
4 # Разделим dataframe на обучающий, валидационный и тестовый используя размеры, найденные на прошлом шаге как индексы
5 df_train = data.iloc[:train_amount].copy()
6 df_validation = data.iloc[train_amount:train_amount+validation_amount].copy()
7 df_test = data.iloc[train_amount+validation_amount:].copy()
```

Рисунок 10 – разбиение набора данных.

Создание объектов «Dataset»

Класс **CustomDataset** является пользовательской реализацией абстрактного класса **Dataset**, который предоставляет PyTorch. **Dataset** является базовым классом для всех пользовательских наборов данных PyTorch и предоставляет возможность использовать произвольные данные, например, изображения, текст, аудио и числовые данные.

Цель **CustomDataset** заключается в создании объектов, которые будут предоставлять данные в нужном формате для обучения модели. В данном случае, класс **CustomDataset** преобразует датафрейм с данными в PyTorch тензоры, которые могут быть использованы для обучения модели. Класс хранит признаки и целевую переменную, которые будут возвращаться по индексу при вызове

метода `__getitem__`, а также длину набора данных, которая будет возвращена методом `__len__`.

Объекты класса **Dataset** используются для инкапсуляции данных в PyTorch, таких как обучающие, тестовые и валидационные выборки. Для того, чтобы применять эти наборы данных для обучения модели, PyTorch предоставляет класс **DataLoader**, который позволяет загружать данные из набора данных в пакетах (batches), обрабатывать их параллельно и передавать на вход модели.

Использование класса **CustomDataset** и его объектов является удобным способом инкапсуляции данных и обеспечения их приведения в нужный формат для обучения модели в PyTorch.

```
1 class CustomDataset(Dataset):
2     def __init__(self, df):
3         self.features = torch.tensor((df.drop(['Loan'], axis=1)).values, dtype=torch.float32)
4         self.targets = torch.tensor((df['Loan']).values.reshape(-1,1), dtype=torch.float32)
5
6     def __getitem__(self, idx):
7         return self.features[idx], self.targets[idx]
8
9     def __len__(self):
10        return len(self.features)
```

```
1 train_dataset = CustomDataset(df_train)
2 val_dataset = CustomDataset(df_validation)
3 test_dataset = CustomDataset(df_test)
```

Рисунок 11 – реализация класса CustomDataset.

Объекты класса DataLoader

DataLoader - это класс PyTorch, который облегчает загрузку данных для обучения и инференса моделей машинного обучения. Он предоставляет возможность генерировать мини-пакеты (batches) из данных, автоматически применять преобразования к данным (например, нормализацию) и распараллеливать загрузку данных для ускорения обучения.

DataLoader принимает на вход объект класса **Dataset** и параметры для настройки загрузки данных, такие как размер мини-пакета, количество процессов загрузки данных и т.д. Затем он генерирует итерируемый объект, который возвращает мини-пакеты данных и соответствующие метки в заданном формате.

`DataLoader` может использоваться для обучения модели на обучающей выборке, для проверки модели на валидационной выборке и для применения обученной модели на тестовой выборке. Он также может быть полезен при предобработке данных, например, для сокращения размера изображений или выделения признаков.

Использование `DataLoader` может значительно упростить процесс загрузки и обработки данных для модели машинного обучения, повысить скорость обучения и улучшить качество модели.

```
1 train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
2 val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
3 test_loader = DataLoader(test_dataset, batch_size=10, shuffle=False)
```

Рисунок 12 – создание объектов класса `DataLoader`.

V. ОПРЕДЕЛЕНИЕ ПРИНЦИПИАЛЬНОЙ АРХИТЕКТУРЫ НЕЙРОСЕТЕВОЙ МОДЕЛИ ГЛУБОКОГО ОБУЧЕНИЯ.

Определение констант

Перед началом обучения необходимо определить некоторые константы, которые в дальнейшем будут передаваться в модель.

n_features: Это количество признаков в вашем датасете. В данном наборе данных **n_features** имеет значение 27, так как в наборе данных есть 27 различных признаков, которые используются для предсказания целевой переменной (в данном случае **Loan**).

n_outputs: Это количество выходов (output) нейронной сети. В данном случае **n_outputs** имеет значение 1, что означает, что у модели есть только один выход, который представляет собой вероятность того, что заёмщик окажется платёжеспособным и заявку на кредит можно одобрить.

BATCH_SIZE: Это количество образцов (samples) в каждой итерации обучения. Модель будет обновлять свои веса после каждой итерации обучения на **BATCH_SIZE** образцах. В данном случае **BATCH_SIZE** имеет значение 7000, что означает, что модель будет обновлять свои веса после каждой итерации обучения на 7000 образцах.

num_epochs: Это количество эпох (epochs), или проходов по всем образцам в датасете, которые модель будет обучаться. **num_epochs** имеет значение 120, что означает, что модель будет проходить через все образцы в датасете 120 раз.

n_hidden: Это количество скрытых нейронов (hidden neurons) в каждом слое нейронной сети. **n_hidden** имеет значение 108, что означает, что каждый слой в нейронной сети будет иметь 108 скрытых нейронов.

LR: Это скорость обучения (learning rate), которая определяет, насколько быстро модель будет менять свои веса в процессе обучения. В данном случае **LR** имеет начальное значение 0.1, что означает, что модель будет изменять свои веса с относительно большой скоростью в процессе обучения на первых эпохах обучения, однако в последствие скорость обучения будет изменяться.

```

1 n_features = train_dataset.features.shape[1] #27
2 n_outputs = train_dataset.targets.shape[1] #1
3 BATCH_SIZE = 7000
4 num_epochs = 120
5 n_hidden = 108
6 LR = 0.1

```

Рисунок 13 – определение констант.

Определение архитектуры модели

Моя модель представляет собой нейронную сеть с двумя скрытыми слоями и использует функцию активации Sigmoid для нелинейного преобразования.

Объяснение каждого слоя:

BatchNorm1d(n_features) - Этот слой принимает входные данные и нормализует их. Это позволяет модели быстрее сходиться, ускоряет обучение и уменьшает вероятность застревания в локальных минимумах. Слой BatchNorm1d применяет стандартное масштабирование данных, чтобы они имели среднее значение 0 и стандартное отклонение 1. Также нормализация позволяет бороться с тем, что данные имеют разную размерность.

nn.Linear(n_features, hidden_size) - Этот слой - линейный слой, принимает на вход тензор размера **n_features** и возвращает тензор размера **hidden_size**. В представленной модели, этот слой получает на вход 27 признаков и переводит их в скрытое представление заданного размера **hidden_size (108)**.

nn.Sigmoid() - Это функция активации, которая используется для нелинейного преобразования выходных данных линейного слоя. В данной модели, Sigmoid помогает ограничить значения выходного тензора в диапазоне [0, 1]. Выходные данные функции активации можно интерпретировать как вероятность быть платёжеспособным заёмщиком.

nn.Linear(hidden_size, n_output) - Этот слой - линейный слой, который принимает на вход скрытое представление и возвращает выходной тензор размера **n_output**. В данной модели, это размерность выходного тензора равна 1,

так как мы решаем задачу бинарной классификации (один выход - вероятность положительного класса).

nn.Sigmoid() - Это функция активации, которая снова применяется для нелинейного преобразования выходных данных линейного слоя, аналогично второму слою Sigmoid.

В целом, выбор данных слоев основывается на общих рекомендациях для построения простых нейронных сетей: используйте нормализацию данных, нелинейные функции активации.

```
1 class LoanClassifier(nn.Module):
2     def __init__(self, n_features, hidden_size, n_output):
3         super(LoanClassifier, self).__init__()
4
5         self.layers = nn.Sequential(
6             nn.BatchNorm1d(n_features),
7             nn.Linear(n_features, hidden_size),
8             nn.Sigmoid(),
9             nn.Linear(hidden_size, n_output),
10            nn.Sigmoid()
11        )
12
13    def forward(self, x):
14        probs = self.layers(x)
15        return probs
```

Рисунок 14 – архитектура нейронной сети

Настройка дополнительных параметров обучения

Конечная цель обучения нейронной сети - это минимизация функции ошибки (loss function).

nn.BCELoss(): это функция потерь, используемая для бинарной классификации, где каждый объект может принадлежать только к одному из двух классов. BCE (Binary Cross-Entropy) Loss обычно используется в таких задачах, где модель должна определить вероятность принадлежности объекта к одному из двух классов. В данном случае, BCE Loss используется для определения того, насколько хорошо модель прогнозирует вероятность выдачи кредита клиенту.

optim.Adam(model.parameters(), lr=LR): это оптимизатор Adam, который используется для обновления параметров модели на основе градиента функции потерь. Он относится к методам стохастической оптимизации, которые используют первый и второй моменты градиента для нахождения оптимального направления для обновления параметров модели. Оптимизатор обновляет параметры модели в соответствии с градиентом функции потерь, который вычисляется на каждом шаге.

MultiStepLR(optimizer, milestones=[1,4,8,12], gamma=0.5): это регулятор скорости обучения, который позволяет изменять скорость обучения в процессе обучения. MultiStepLR уменьшает скорость обучения на определенных этапах обучения. milestones определяет количество эпох, после которых нужно уменьшить скорость обучения, а gamma определяет, насколько нужно уменьшить скорость обучения. В данном случае, скорость обучения уменьшается на 50% после 1, 4, 8 и 12 эпох обучения. Это помогает модели сходиться к оптимальному решению быстрее и избежать переобучения.

```
1 model = LoanClassifier(n_features, n_hidden, n_outputs)
2 criterion = nn.BCELoss()
3 optimizer = optim.Adam(model.parameters(), lr=LR)
4 scheduler = MultiStepLR(optimizer, milestones=[1,4,8,12], gamma=0.5)
```

Рисунок 15 – настройка дополнительных параметров обучения

Цикл обучения

Цикл обучения - это основной процесс обучения нейросети, в котором происходит подача данных на вход нейросети, вычисление выходных значений и значения функции потерь, а также обновление параметров нейросети на каждой итерации.

На каждой итерации внутри цикла обучения происходит обработка одного батча данных, передаваемых в модель через DataLoader. Сначала инициализируются градиенты модели, обнуляется градиент оптимизатора и вычисляются предсказания модели для текущего батча. Затем вычисляется значение функции потерь, которая сравнивает предсказания модели и настоящие значения. Полученные значения передаются обратно через нейросеть для

вычисления градиента, который затем используется оптимизатором для обновления весов модели. Для регулирования скорости обучения используется MultiStepLR, который уменьшает скорость обучения на определенных этапах. Также на каждой эпохе сохраняются тренировочный и валидационный F1_score и ошибка для того, чтобы в последствие посмотреть динамику изменения этих метрик на графике.

Для каждой эпохи происходит подсчет метрик точности и f1_score на тренировочных и валидационных данных, которые сохраняются в списки.

Кроме того, в конце каждой 10-ой эпохи выводятся текущие значения функции потерь, точности и F1-меры для тренировочных и валидационных данных, а также текущая скорость обучения. Это помогает следить за переобучением: как только метрики на тренировочных данных растут, а метрики на валидационных данных начинают снижаться, то происходит переобучение.

Такой цикл обучения позволяет постепенно улучшать предсказания модели на каждой эпохе за счет подстройки параметров нейросети в соответствии с передаваемыми данными.

```
1 train_f1_s = []
2 validation_f1_s = []
3 loss_s = []
4
5 for epoch in range(num_epochs):
6     for i, (features, targets) in enumerate(train_loader):
7         optimizer.zero_grad()
8         predictions = model.forward(features)
9         loss = criterion(predictions, targets)
10        loss.backward()
11        optimizer.step()
12
13    scheduler.step()
14    current_lr = optimizer.state_dict()['param_groups'][0]['lr']
15
16    train_precision, train_f1 = get_metrics(train_loader, model)
17    val_precision, val_f1 = get_metrics(val_loader, model)
18
19    train_f1_s.append(train_f1)
20    validation_f1_s.append(val_f1)
21    loss_s.append(loss.item())
22
23    if epoch%10==0:
24        print(f'''
25        Epoch:{epoch}
26        \tLoss = {loss:.2f}
27        \tTrain precision      = {train_precision:.2%}
28        \tTrain F1_score       = {train_f1:.2%}
29        \tValidation precision = {val_precision:.2%}
30        \tValidation F1_score  = {val_f1:.2%}
31        \tLearning rate       = {current_lr:.5f}''')
32
```

Рисунок 16 – Цикл обучения

VI. ОЦЕНКА КАЧЕСТВА МОДЕЛИ

Оценка качества модели является важным этапом в машинном обучении, так как она позволяет оценить, насколько хорошо модель работает на новых данных. Для этого используются метрики, которые позволяют измерить качество предсказаний модели. Существует множество метрик, в зависимости от задачи, которую решает модель. В данном случае, для оценки качества модели на тестовой выборке, были выведены метрики `precision` и `f1_score`.

Метрика **accuracy** измеряет долю правильных ответов модели от общего числа предсказаний. Она хорошо работает в случае, когда классы сбалансированы, но не является подходящей для несбалансированных данных.

Метрика **recall** (полнота) показывает, как много объектов положительного класса было правильно определено моделью от общего числа объектов положительного класса в тестовой выборке.

Метрика **precision** (точность) показывает, как много объектов, которые модель отнесла к положительному классу, действительно являются положительными.

Метрика **f1_score** представляет собой гармоническое среднее между `precision` и `recall`, и является хорошей метрикой для несбалансированных данных, так как учитывает и полноту, и точность.

В данном случае, `precision` и `f1_score` были выбраны в качестве основных метрик, так как они наиболее важны для решаемой задачи: точность определения отрицательного класса (неплатежеспособных) и учет несбалансированности классов.

Графики метрик

В первую очередь следует посмотреть на метрики эффективности после завершения цикла обучения модели. Точность (`precision`) и `F1_score` на тестовой выборке даже превышают соответствующие метрики на обучающей выборке!

Сравнение метрик эффективности модели на трёх выборках

	Обучающая	Валидационная	Тестовая
Precision	85.97%	84.96%	86.26%
F1_score	81.76%	80.67%	82.04%

Исходя из исключительно этой таблицы можно сделать вывод что модель хорошо обучена и показывает высокую точность на всех трех выборках - обучающей, валидационной и тестовой. Метрики на тестовой выборке выше, чем на валидационной выборке, что говорит о том, что модель не переобучена, а, наоборот, имеет хорошую обобщающую способность и работает эффективно на новых данных.

Далее хотелось бы посмотреть динамику изменения F1_score во время обучения на обучающей и валидационной выборках.

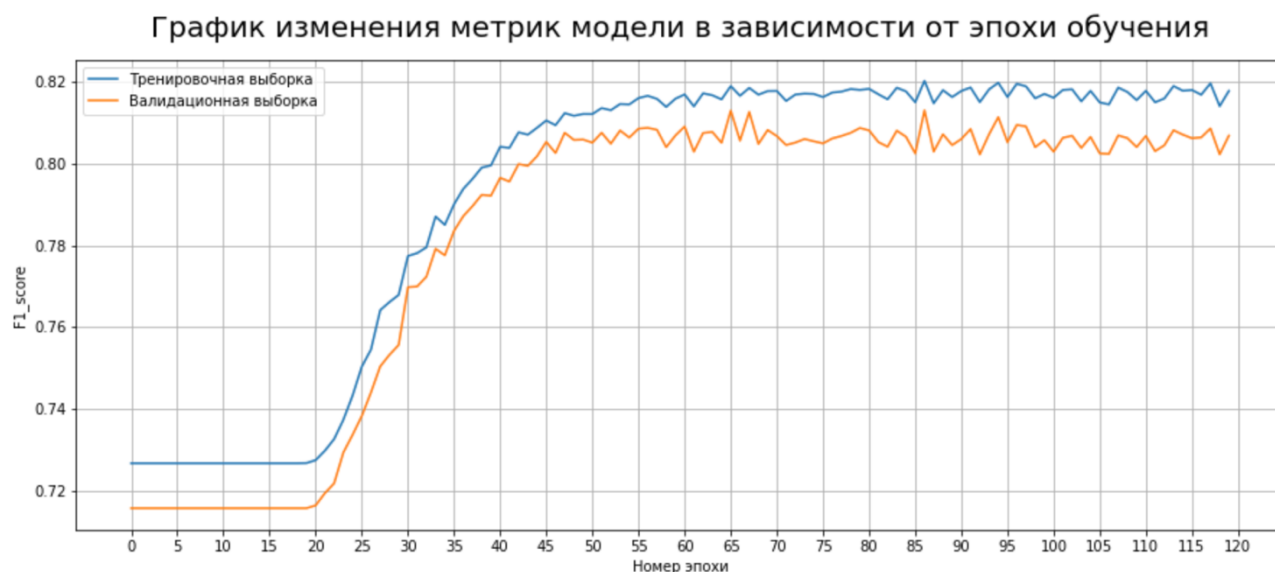


Рисунок 17 – динамика изменения F1_score во время обучения

Исходя из графика выше можно сделать несколько выводов:

- В начале обучения модель не была достаточно уверенной в своих предсказаниях, поэтому метрики не изменялись. Но по мере улучшения весов и более точного предсказания, метрики начали расти.

- В первых эпохах модель просто запоминала обучающие данные и переобучалась, что приводило к нестабильным метрикам. Но в дальнейшем модель начала обобщать лучше и давать более точные предсказания, что привело к росту метрик.
- После примерно 90-й эпохи скорость обучения модели снизилась и ей уже было тяжело оптимизировать свои веса.
- Ни на каком этапе обучения модели не наблюдается расхождения функций таким образом, что метрики на обучающей выборке росли, а на валидационной бы падали. Это свидетельствует об отсутствии переобучения модели.

График изменения ошибки модели в зависимости от эпохи обучения

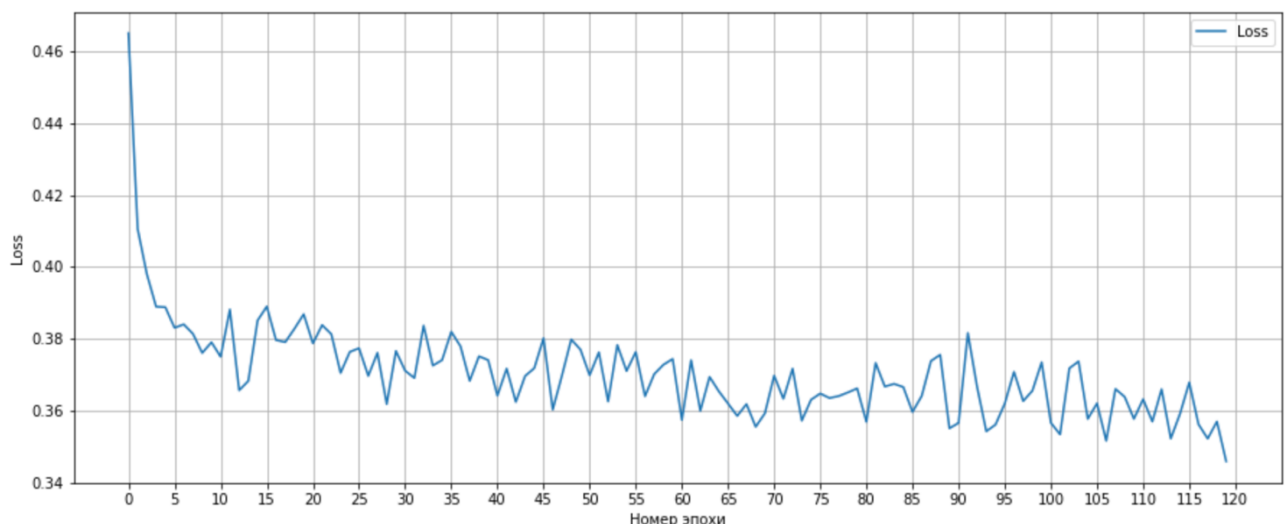


Рисунок 18 – динамика изменения значения функции ошибки

Когда модель обучается, то необходимо, чтобы функция ошибки как можно быстрее сходилась к минимуму, т. е. значение ошибки должно уменьшаться с каждой эпохой. В идеале график функции ошибки должен быть гладким и монотонно убывающим, без резких скачков или плато.

Однако, в реальности, график функции ошибки может иметь различные формы. Например, в начале обучения мы можем наблюдать резкий спад значения функции ошибки, что свидетельствует о том, что модель быстро находит общие закономерности в данных. После этого спада значение функции ошибки может медленно убывать, что говорит о том, что модель постепенно уточняет свои предсказания.

Также возможна ситуация, когда функция ошибки не убывает на протяжении нескольких эпох и затем резко падает. Это может быть связано с тем, что оптимизатор модели столкнулся с локальным минимумом и нашел новый оптимум на более низком уровне функции ошибки. Кроме того, функция ошибки может иметь плато, т. е. значение ошибки не изменяется на протяжении нескольких эпох, что говорит о том, что модель может находиться в локальном минимуме.

В целом, это хорошо, что график функции ошибки монотонно убывает, хоть и наблюдаются некоторые резкие скачки то вверх, то вниз, однако всё же эти скачки формируют нисходящий тренд.

Пример интерпретации выходных данных модели

В данной части работы продемонстрировано, как можно было бы интерпретировать результат работы модели наглядно. А именно: взять один батч из тестовой выборки размеров 10 точек данных и подать его на вход модели, сравнить предсказания модели с настоящим значением целевой переменной.

После того, как модель произвела предсказания на выбранном батче, собираются эти предсказания и реальные значения целевой переменной в DataFrame.

Каждая строка в DataFrame соответствует одному примеру из батча. Столбец «True Loan» содержит настоящие значения целевой переменной. Столбец «Positive class probability» содержит вероятность (выходные данные модели) принадлежности к классу «1» (положительному классу) для каждого примера из батча. А столбец «Predicted class» содержит предсказанные классы моделью для каждого примера из батча.

	True Loan	Positive class probability	Predicted class
0	1.00	88.76%	1.00
1	1.00	99.13%	1.00
2	1.00	99.09%	1.00
3	1.00	82.66%	1.00
4	1.00	91.70%	1.00
5	0.00	25.69%	0.00
6	1.00	97.74%	1.00
7	1.00	90.87%	1.00
8	1.00	29.70%	0.00
9	1.00	97.80%	1.00

Рисунок 19 – визуализация результатов работы модели

Из DataFrame можно увидеть, какие примеры были предсказаны моделью правильно, а какие нет, а также какова вероятность принадлежности каждого примера к положительному классу. Например, в строке 0 можно заметить, что настоящее значение целевой переменной равно 1, а модель предсказала 1 с вероятностью 88,76%. То есть модель уверенно предсказала принадлежность к положительному классу. Аналогично для остальных примеров из данного батча. Однако на примере с индексом 8 можно заметить, что модель с уверенностью 29,70% предсказала положительный класс, то есть посчитала заёмщика неплатёжеспособным, однако он оказался платёжеспособным.

VII. ПОСТРОЕНИЕ АЛЬТЕРНАТИВНЫХ МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ

В этой главе своей работы будут построены несколько моделей классического машинного обучения и сравнятся результаты этих моделей с метриками нейронной сети. Наиболее распространенными моделями классического машинного обучения для задач бинарной классификации являются:

1. Логистическая регрессия (Logistic Regression) - это одна из самых простых и широко используемых моделей. Она используется для предсказания вероятности бинарной классификации, основываясь на линейной комбинации входных признаков. Параметр `solver='liblinear'` означает, что для решения оптимизационной задачи будет использоваться алгоритм линейной оптимизации (LIBLINEAR), который хорошо работает на небольших выборках.
2. Случайный лес (Random Forest) - это алгоритм, который комбинирует несколько деревьев решений, чтобы увеличить точность и снизить переобучение. Он работает путем создания нескольких деревьев решений и объединяя их, чтобы принять решение о классификации. В модели `RandomForestClassifier` в параметр `n_estimators` передается число, которое указывает на количество деревьев, которые будут построены в лесу случайных деревьев (Random Forest). Параметр `n_estimators=100` означает, что в лесу будет 100 деревьев.
3. Градиентный бустинг (Gradient Boosting) - это алгоритм, который комбинирует несколько простых моделей, таких как деревья решений, и использует градиентный спуск, чтобы минимизировать ошибку классификации. Он работает путем построения модели постепенно, улучшая ее на каждом шаге. В модели `GradientBoostingClassifier` по умолчанию используется 100 деревьев решений (decision trees).

Для более справедливого и честного процесса обучения в качестве обучающей выборки для этих моделей используются таблицы `df_train` и `df_test`.

Для обучения моделей используется пользовательская функция **train_classic_ML**, которая на вход принимает модель, затем обучает её, делает предсказания на тестовой выборке, а затем вычисляет метрики эффективности Precision и F1_score. Затем, все эти метрики собираются в одну таблицу, добавив туда метрики нейронной сети для сравнения.

	CustomNeuralNetwork	LogisticRegression	RandomForest	GradientBoosting
Precision	86.26%	82.81%	85.34%	85.58%
F1_score	82.04%	90.09%	90.83%	91.08%

Рисунок 20 – сравнение метрик нейросети и альтернативных моделей

Опираясь на полученные результаты, можно сделать несколько выводов. Исходя из того, что в текущей задаче крайне важно точно предсказывать отрицательный класс (неплатежеспособных людей), Precision играет более важную роль, чем Recall, так как лучше отказать надежному заемщику, чем выдать кредит ненадежному. Поэтому, на основе таблицы, мы можем сделать вывод, что моя нейронная сеть имеет наилучший Precision среди всех моделей, включая модели классического машинного обучения, а это значит, что она качественнее выделит неплатёжеспособных людей.

Однако, F1-мера является более комплексной метрикой, которая учитывает как Precision, так и Recall, и может быть полезна, когда Precision и Recall имеют одинаковое значение для задачи. Модели классического машинного обучения, особенно RandomForestClassifier и GradientBoostingClassifier, имеют более высокий F1-меру, чем моя нейронная сеть, что может означать, что они более точны в предсказании обоих классов, но немного менее точны в отрицательном классе.

Таким образом, в целом, на основе Precision я могу сказать, что моя нейронная сеть хорошо справляется с задачей и может быть предпочтительнее тогда, когда финансовой организации важнее точно определять неплатежеспособных заемщиков.

VIII. ЗАКЛЮЧЕНИЕ

В данной работе исследованы возможности использования искусственных нейронных сетей для классификации держателей кредитных карт по вероятности дефолта, то есть сегментации заёмщиков на платёжеспособных и неплатёжеспособных. В процессе выполнения работы были использованы основные библиотеки для работы с нейронными сетями и машинным обучением: pandas, numpy, random, torch, matplotlib, seaborn, sklearn, tqdm.

Первым этапом данной работы был комплексный анализ и предобработка набора данных. Каждый признак изучен отдельно, удалены неважные признаки, преобразованы категориальные признаки в численные, обработаны отсутствующие значения и удалены выбросы. Изучены распределения значений каждого признака путём визуализации данных с помощью гистограммы и ящика с усами.

Затем набор данных разделён на обучающую, валидационную и тестовую выборки. Для решения задачи предсказания платёжеспособности заёмщика я разработал свою архитектуру нейронной сети. В процессе обучения модели использовался оптимизатор Adam, регулятор скорости обучения MultiStepLR и функцию ошибки BCELoss.

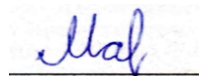
После обучения модели были проанализированы метрики эффективности модели (Precision и F1_score), а также построены графики динамики изменения этих метрик в зависимости от эпохи обучения. Также построены график функции ошибки, чтобы можно было убедиться, что она монотонно убывает.

В дальнейшем был проведён сравнительный анализ пользовательской нейронной сети с тремя моделями классического машинного обучения: LogisticRegression, RandomForestClassifier и GradientBoostingClassifier. Результаты показали, что нейронная сеть имеет более высокий Precision, что говорит о том, что нейронная сеть лучше предсказывает отрицательный класс (неплатёжеспособных людей), чем модели классического машинного обучения.

В заключение, я хотел бы подчеркнуть, что данная работа доказывает, что использование искусственных нейронных сетей в задаче классификации

держателей кредитных карт по вероятности дефолта является не менее эффективным подходом, чем использование классических моделей машинного обучения. Моя модель имеет чуть более высокий Precision, что говорит о том, что она лучше предсказывает неплатёжеспособных людей, что является важным критерием для финансовой организации.

Вся работа выполнена собственноручно.



IX. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. М.В. Коротеев. Об основных задачах дескриптивного анализа данных.
2. М.В. Коротеев. Учебное пособие по дисциплине “Анализ данных и машинное обучение” - 2018.
3. Jason Brownlee. "Deep Learning with Python," Machine Learning Mastery, 2017.
4. Aurelien Geron. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems," O'Reilly Media, 2019.
5. P. Y. Simard, K. Stechly, "Deep learning and asset allocation", Journal of Risk and Financial Management, 2020.
6. "PyTorch Deep Learning Hands-On: Build CNNs, RNNs, GANs, reinforcement learning, and more, quickly and easily" by Sherin Thomas and Sudhanshu Passi (2020)
7. Matplotlib // Matplotlib documentation URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.text.html (дата обращения: 18.03.2023).
8. Pandas // Pandas documentation URL: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html> (дата обращения: 15.03.2023).
9. PyTorch // PyTorch documentation URL: <https://pytorch.org/docs/stable/tensors.html> (дата обращения: 23.04.2023).
10. Scikit // Scikit documentation URL: <https://scikit-learn.org/stable/index.html> (дата обращения: 10.04.2023).