**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет»**
Институт вычислительной математики и информационных технологий
Кафедра системного анализа и информационных технологий

Направление подготовки: 02.03.02 — Фундаментальная информатика и информационные технологии

Профиль: Системный анализ и информационные технологии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

# ПОИСК ОПТИМАЛЬНОГО ПОДХОДА К РЕШЕНИЮ ЗАДАЧИ ОЦЕНКИ БЛАГОНАДЕЖНОСТИ КЛИЕНТА БАНКА

Обучающийся 4 курса
группы 09-131 (Алпатов И.А.)

Руководитель
канд. физ-мат. наук, доцент (Васильев А.В.)

Консультант
ст. преподаватель (Денисов М.П.)

Заведующий кафедрой системного анализа
и информационных технологий,
канд. физ-мат. наук, доцент (Васильев А.В.)

Казань – 2025

**ОГЛАВЛЕНИЕ**

# ВВЕДЕНИЕ

В настоящее время основной из задач банковского блока рисков является управление системой принятия решений и ее сопровождение. Эта система придумана для упрощения взаимодействия клиента с банком и отвечает за рассмотрение онлайн-заявок на всевозможные кредиты различных банковских сегментов. Чаще всего внутри этой системы встроены жесткие правила, по которым определяется благонадежность человека, подающего заявку. Эти правила постоянно регулируются в зависимости от экономической ситуации в стране. Все они существуют для отсечения неблагонадежных заемщиков, чаще всего – мошенников.

Данные правила составляются на основе информации, имеющейся у банка, однако такой информации может быть недостаточно, поэтому банк прибегает к помощи сторонних сервисов, которые предоставляют дополнительные признаки по субъекту. С помощью них ранжировать заемщиков становится проще.

Существует множество подобных сервисов, поэтому перед банком встает вопрос: какой сервис лучше выбрать? Данная проблема решается с помощью ретровыборок. Ретровыборка – это набор заявок, выданных банком определенное время назад, по которым уже есть информация о том, насколько исправно платятся кредиты. Эти заявки отправляются в сервис, сервис же в свою очередь присылает признаки по заемщикам. Банк не знает значения этих признаков, однако может проверить их влияние на работу различных моделей, выделяющих неблагонадежных заемщиков из данного набора заявок. Если признаки оказываются полезными, банк заключает с сервисом договор и встраивает их в свою систему принятия решений.

Данная работа направлена на решение задачи оценки полезности признаков присылаемых сервисов. Так как система принятия решений представляет собой жесткие правила, помимо полезности признаков нужно понять, как именно их следует встроить в систему принятия решения.

Цель данной работы – разработать интерпретируемую модель, решающую задачу оценки благонадежности клиента банка по входным признакам. Основной проблемой в данном случае является высокий дисбаланс среднего потока заявок банка, так как подавляющее большинство заявок не имеют просроченных платежей (примерно 95%).

Для достижения цели были поставлены следующие задачи:

1) исследование и внедрение способов улучшения предобработки данных;

2) исследование, разработка и реализация различных моделей машинного обучения;

3) сравнительная оценка различных характеристик реализованных моделей;

4) интерпретация лучших из моделей.

## 1. Используемые наборы данных

Как было сказано выше, основной проблемой при работе с банковскими данными является высокий дисбаланс классов, поэтому в качестве используемых наборов данных (далее - НД) было представлено 3 варианта:

1) НД Titanic для первоначальной проверки работоспособности разрабатываемых алгоритмов с количеством строк 1000 и соотношением классов 38%-62%, позволяющий оценить качество классификации на сбалансированном НД;

2) банковский НД 1 – 1762 строки, отношение классов 65/1697 (3.7%-96.3%);

3) банковский НД 2 – 61006 строк, отношение классов 660/60346 (1%-99%).

Как видно, во 2 банковском НД более ощутимо выражен дисбаланс классов, однако количество строк в нем при этом больше. Добавление этого НД было необходимо, так как в НД 1 при разделении на выборки оставалось слишком мало заявок целевого класса для тестовой:

$$65 * 0.2 = 13,$$

где 0.2 – коэффициент разбиения на выборки.

В связи с этим становилось достаточно тяжело считать и сравнивать метрики моделей.

## 2. Предобработка банковских данных

В практической работе с различными данными в связи с невозможностью реализовать какую-либо систему так, чтобы она была полностью совместима с какой бы то ни было другой системой, часто возникают различные вопросы предобработки данных.

### 2.1. Разделение строки сервисных признаков

После того, как банк послал идентификаторы заемщиков ретровыборки, сервис возвращает набор признаков в определенном формате (рисунок 1):

| MAINRULES |
| --- |
| 1.11=>0;1.12=>0;1.13=>0;1.14=>0;1.15=>0;1.16=>... |
| 1.11=>0;1.12=>0;1.13=>0;1.14=>0;1.15=>0;1.16=>... |
| 1.11=>0;1.12=>0;1.13=>0;1.14=>0;1.15=>0;1.16=>... |
| 1.11=>0;1.12=>0;1.13=>0;1.14=>0;1.15=>0;1.16=>... |
| 1.11=>0;1.12=>0;1.13=>0;1.14=>0;1.15=>0;1.16=>... |

Рисунок 1 – Формат признаков

Возникает проблема правильного парсинга этих входных признаков. Алгоритм парсинга приведен на рисунке 2:

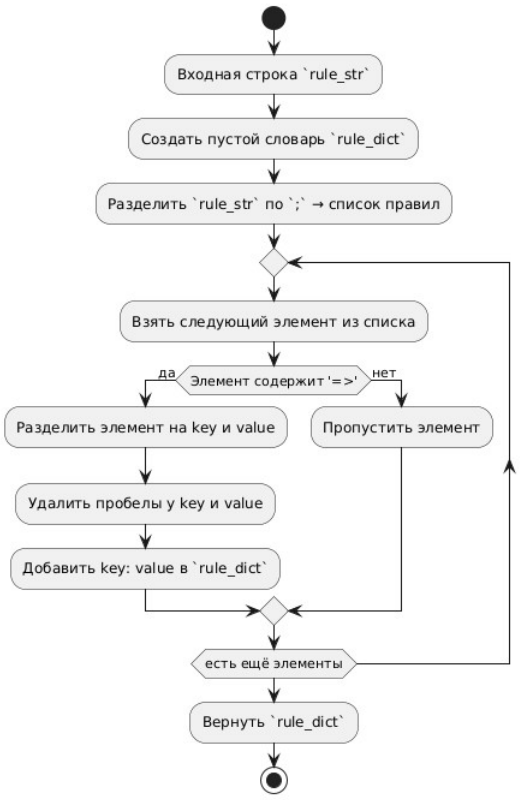


Рисунок 2 – Парсинг входных признаков

### 2.2. Формирование входных признаков

К полученному набору данных добавляются внутренние банковские признаки и целевая переменная (target). Если на рассматриваемом сервисе в сочетании с внутренними входными признаками удается достичь целей заказчика (процент отрезаемого портфеля, процент риска отрезаемого портфеля), то сервис считается полезным.

После добавления банковских признаков общее их количество может превышать 200 штук, признаки представлены бинарными, категориальными и скоринговыми переменными. Так как большинство моделей машинного обучения работают с числовыми нормализованными данными, для корректного их использования нужна предварительная обработка данных:

1) one-hot encoding для категориальных признаков, где каждое из значений категориального признака выносится в качестве отдельного бинарного поля;

2) фильтрация признаков, чтобы ускорить обучение и убрать несущественные;

3) проработка полей, которые не будут использоваться в качестве признаков (изначальные категориальные признаки, целевая переменная, лишние данные, например, дата выдачи заявки).

Алгоритм фильтрации признаков:

1) удаление полей, которые имеют 30 и более процентов пустых значений;

2) вычисление коэффициента GINI для каждого из оставшихся признаков как $GINI = AUC\_ROC * 2 - 1$, где AUC_ROC – Area Under the ROC Curve (площадь под ROC-кривой);

3) удаление признаков имеющих $GINI <= 0.1$ как малоинформативных.

Получившийся НД затем передается в модель для обучения. Пример ROC-кривой можно увидеть на рисунке 3.

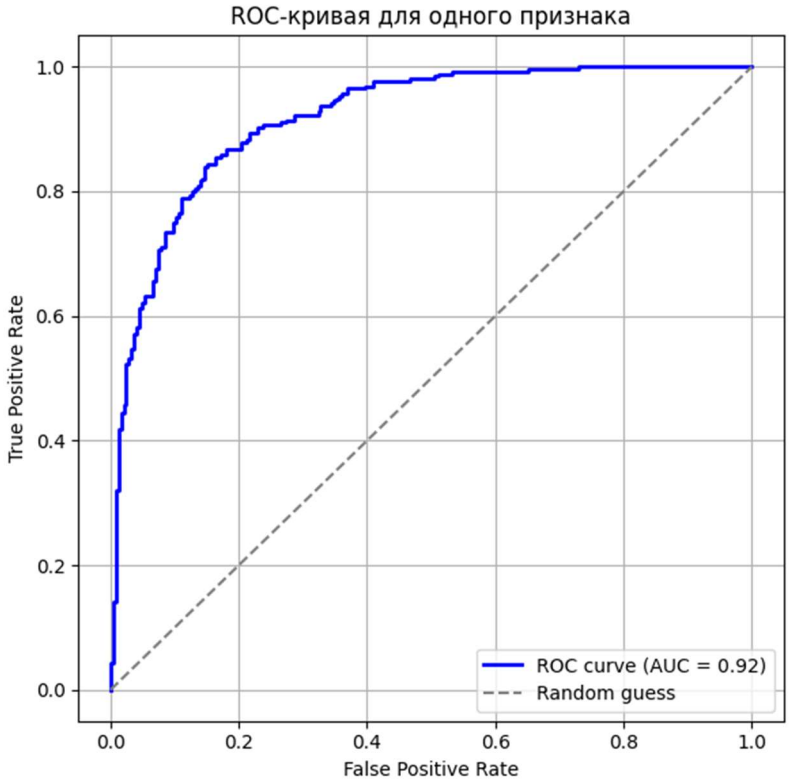


Рисунок 3 – Пример ROC-кривой

### 2.3. Исследование и внедрение способов предобработки данных

Как было сказано выше, основной проблемой в случае работы с банковскими данными является высокий дисбаланс классов среднего потока заявок, так как подавляющее их большинство не имеют просроченных платежей (примерно 95%).

В подобных случаях в обучающей выборке применяют методы искусственного увеличения количества экземпляров класса меньшинства (oversampling) или сокращения количества экземпляров класса большинства (undersampling) для уменьшения дисбаланса [1].

В качестве решения была выбрана техника oversampling, позволяющая увеличить представительность меньшего класса за счёт дублирования или генерации новых элементов. Данная техника была выбрана вследствие необходимости увеличить количество экземпляров 1 класса без изменения количества 0 класса, чтобы не потерять несущие информационную ценность данные. Таким образом, техника undersampling для решения данной проблемы не подходила.

Для реализации данной технологии был использован модуль RandomOverSampler из библиотеки imblearn.over_sampling:

```
# оверсэмплинг
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(sampling_strategy=0.1, random_state=42)
X_res, y_res = ros.fit_resample(X_train, y_train)
```

## 3. Описание алгоритмов

Как только возникла задача о создании интерпретируемого алгоритма машинного обучения, первым делом взгляд был обращен на деревья решений. Рассмотрим подробнее этот алгоритм машинного обучения.

### 3.1. Деревья решений

Дерево решений — это метод визуализации и анализа, используемый для систематизации и упрощения процесса принятия решений. Он представляет информацию в виде древовидной структуры с корнем, ветвями и узлами, где каждый узел представляет собой вопрос или условие, а ветви — возможные ответы или исходы [2].

Чаще всего в качестве условия в узле выступают три характеристики: information gain (далее – информационный выигрыш), основанный на понятии энтропии, или Gini impurity (далее – примесь Джини).

Энтропия измеряет степень нечистоты, неопределенности или неожиданности точек данных. Она колеблется от 0 до 1. Рассмотрим кривую энтропии (рисунок 4):

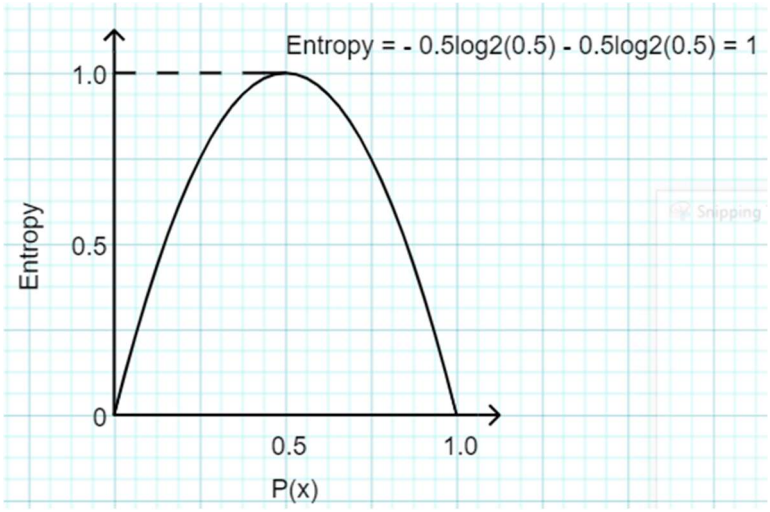


Рисунок 4 – Пример кривой энтропии

Видно, что энтропия равна 0, когда вероятность равна 0 или 1. В итоге получается максимальная энтропия 1 при вероятности 0,5, что означает, что данные идеально рандомизированы.

Формула энтропии:

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i,$$

где $c$ – количество классов, $p_i$ – вероятность выбора точки данных класса. Таким образом, чем выше энтропия, тем «грязнее» данные.

Информационный прирост — это разница между до и после разбиения по энтропии. При построении дерева решений важно найти признак с максимальным значением информационного выигрыша. Такой признак обеспечивает лучшее разделение, лучше классифицирует обучающий набор данных по целевой переменной [3]. Формула информационного прироста:

$$Gain(S, a) = E(S) - \sum_{v \in values(a)} \frac{|S_v|}{|S|} E(S_v),$$

где:

1) $a$ – атрибут или метка класса,

2) $E(S)$ – энтропия НД $S$,

3) $|S_v|/|S|$ – отношение количества значений в $S_v$ к количеству значений в $S$.

Примесь Джини — это вероятность неправильной классификации случайной точки данных в наборе данных. Это метрика примеси, поскольку она показывает, чем модель отличается от чистого деления.

В отличие от энтропии, примесь Джини имеет максимальное значение 0,5 (очень нечистая классификация) и минимальное 0 (чистая классификация) (рисунок 5):

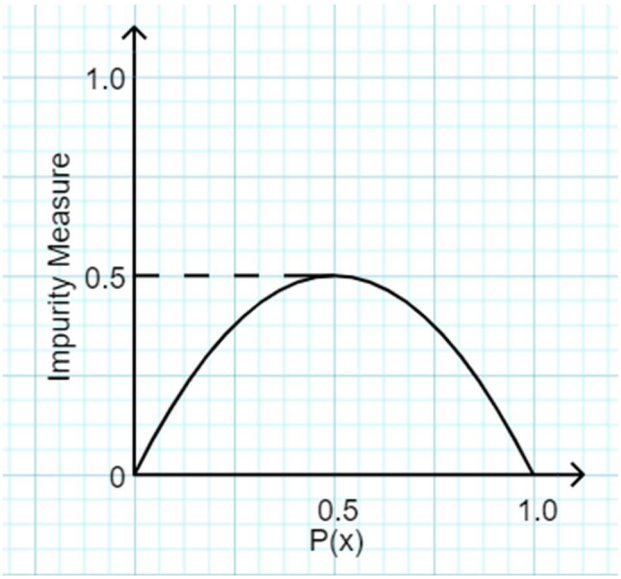


Рисунок 5 – Примесь Джини

Формула примеси Джини:

$$Gini = 1 - \sum_{i=1}^{c}(P_i)^2,$$

где $P_i$ – вероятность выбора точки данных класса $i$. Чем меньше значение Gini, тем более «чистое» разбиение дает данный признак.

На предварительном тестировании алгоритмов дерева на НД 1 не получилось получить никаких приемлемых результатов (рисунок 6):

```
Матрица ошибок:
[[339   0]
 [ 14   0]]

Classification report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98       339
         1.0       0.00      0.00      0.00        14

    accuracy                           0.96       353
   macro avg       0.48      0.50      0.49       353
weighted avg       0.92      0.96      0.94       353
```

Рисунок 6 – Результаты дерева решений на тестовой выборке

В этот момент появилось предложение заменить критерий разделения в построении дерева. В дальнейших исследованиях деревья решений не использовались.

### 3.2. Жадное дерево

Жадное дерево на самом деле нельзя назвать деревом как таковым. Этот алгоритм представляется двумя блоками, первый из которых представляет собой один спуск в глубину, то есть составление одной ветки. В качестве задаваемых ограничений он принимает риск (отношение просроченных заявок в отсекаемом портфеле ко всем заявкам этого портфеля) и минимальное значение порога (процент просроченных заявок отсекаемого портфеля ко всем просроченным заявкам). Блок-схема алгоритма составления ветки показана на рисунке 7.


Рисунок 7 – Составление ветки

В данном случае условие наилучшего разделения можно интерпретировать как условие наилучшего спуска и записать как:

$$BestI, BestJ, BestK \stackrel{\text{def}}{=} i, j, k: \max \left\{ \frac{prosr_{ijk}}{prosr_{ijk} + neprosr_{ijk}}, i = \overline{1, N}, j = \overline{1, M_i}, k = \overline{1,2} \right\},$$

где:

1) $prosr_{ijk}$ – количество просроченных заявок в случае, если спуститься i-ому признаку, j-ому порогу и k-ому знаку;

2) $neprosr_{ik}$ – количество непросроченных заявок в случае, если спуститься i-ому признаку, j-ому порогу и k-ому знаку;

3) $N, M_i$ – количество признаков и порогов i-ого признака соответственно;

4) $BestI, BestJ, BestK$ – номер признака, порога и знака, по которым происходит наилучший спуск. Если эти величины не найдены или ветка достигла максимально допустимой заданной глубины, то цикл подбора заканчивается и алгоритм завершает свою работу.

Важным условием является также проверка остающегося после спуска НД. Если $prosr/full\_neprosr$, где $full\_neprosr$ – количество просроченных заявок во всем НД, не больше заданного порога, то такой спуск не рассматривается.

В связи с тем, что в рассматриваемых НД используются в том числе скоринговые признаки, необходимо было уменьшить количество рассматриваемых порогов для них. Было принято решение в случае количества порогов большего 20 разбивать отрезок [минимальный порог, максимальный порог] на 20 равных частей и в дальнейшем такое разбиение рассматривать в качестве порогов этого признака.

После разработки алгоритма составления ветки появилась идея о том, что, во-первых, необходимо создать возможность находить в оставшемся после отсечения НД новые ветки, а во-вторых, как-то ограничить минимальный порог, а также риск (отношение просроченных заявок ко всем заявкам портфеля). Так появился алгоритм подбора веток (рисунок 8):

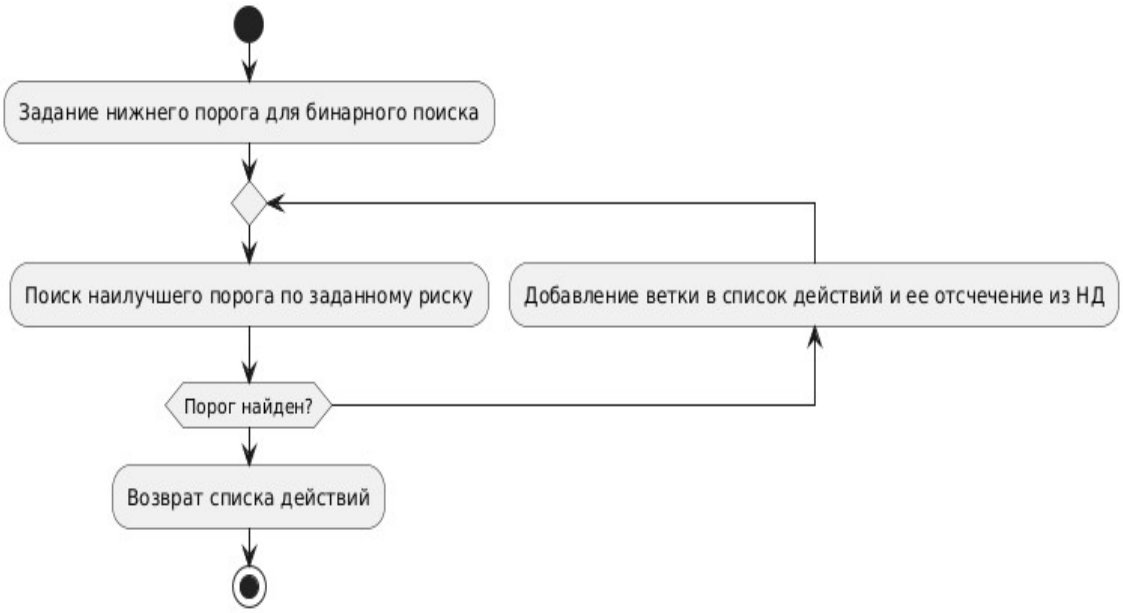


Рисунок 8 – Подбор веток по заданному риску

Начальный порог выбирается как $np = (1 + const\_low)/2$, где $const\_low$ – заданное минимальной значение порога. Затем запускается цикл:

1) выполнение алгоритма поиска ветки с заданным порогом;

2) если ветка нашлась и риск в ней больше заданного значения, то обновляем нижний порог текущим значением. Если нет, то обновляем верхний порог текущим значением;

3) продолжаем, пока разница между верхним и нижним порогом будет меньше 0.01;

4) продолжаем пункты 1-3, пока в случае отсутствия искомой ветки нижний порог не «скатится» к значению $const\_low$.

### 3.3. Перцептрон

Перцептрон — одна из простейших архитектур нейронных сетей, представленная Фрэнком Розенблаттом в 1957 году. В основном он используется для бинарной классификации [4].

Перцептрон состоит из ключевых компонентов, которые работают вместе для обработки информации и составления прогнозов:

- перцептрон принимает несколько входных признаков, каждый из которых представляет собой характеристику входных данных;
- каждому входному объекту присваивается вес, который определяет его влияние на выходные данные. Эти веса корректируются во время тренировки для поиска оптимальных значений;
- перцептрон вычисляет взвешенную сумму своих входных данных, объединяя их с соответствующими весами;
- взвешенная сумма пропускается через ступенчатую функцию, сравнивая ее с пороговым значением для получения двоичного результата (0 или 1);
- окончательные выходные данные определяются функцией активации, часто используемой для задач бинарной классификации.

Пример архитектуры перцептрона с двумя скрытыми слоями можно посмотреть на рисунке 9.

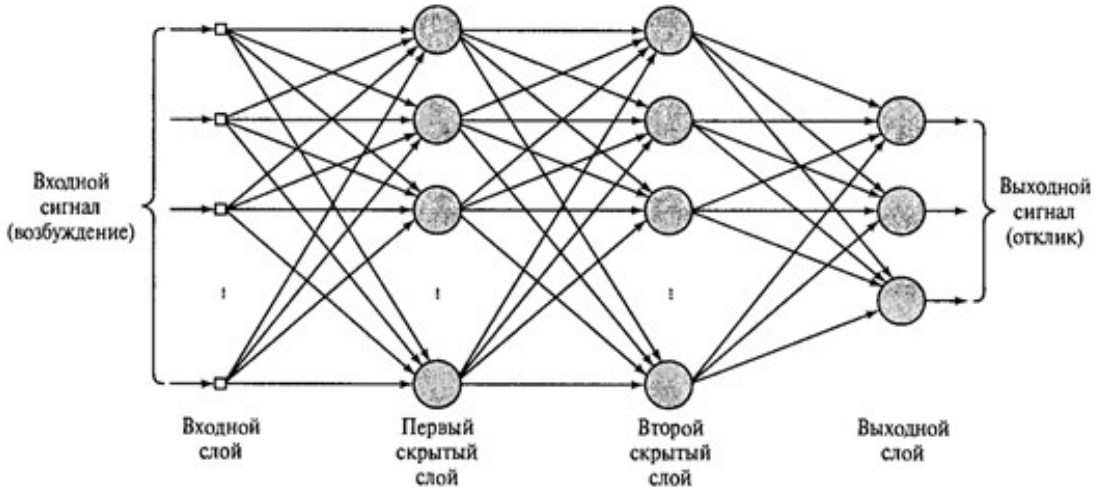


Рисунок 9 – Пример перцептрона

Как видно из рисунка, каждый из входов подается в каждый из нейронов первого скрытого слоя. При этом каждому из нейронов присваиваются веса, а внутри нейрона задана функция взвешенного суммирования и функция активации, результат которой подается нейронам следующего слоя. Весь этот процесс показан на рисунке 10:

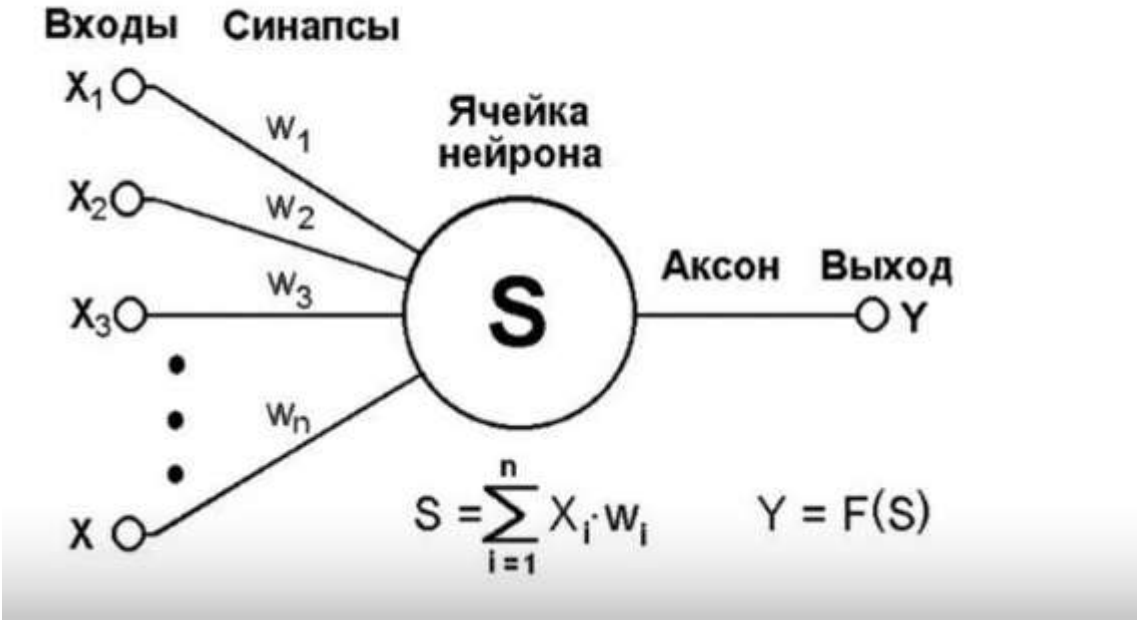


Рисунок 10 – Работа одного нейрона

Для того, чтобы модель обучалась, необходимо обеспечить обновление весов после каждой эпохи, учитывающее ошибку предсказания, которую нужно минимизировать. Для перцептрона на каждой итерации веса w

обновляются с использованием уравнения алгоритма обратного распространения ошибки:

$$w = w + learning\_rate * (expected - predicted) * x,$$

где:

1) $w$ – обновляемые веса;

2) $learning\_rate$ – скорость обучения, которую нужно установить (например, 0.1);

3) $expected - predicted$ – ошибка прогнозирования для модели в обучающих данных, относящихся к весу;

4) $x$ – входное значение.

При построении архитектуры перцептрона задаются в том числе три параметра: $learning\_rate$, количество эпох – количество раз, которая модель пропустит через себя обучающую выборку и $batch\_size$ – размер партии, после каждой из которых в модели обновляются веса. Разбиение на партии необходимо в связи с тем, что обучающая выборка представляет собой достаточно большой массив данных, который сложно обрабатывать за одну итерацию.

В данной работе использовался двухслойный перцептрон, реализованный с помощью библиотеки TensorFlow/Keras. Структура модели использует полносвязные слои с функцией активации ReLU и выходной слой с сигмоидной функцией. Функция активации ReLU: $ReLU(x) = \max(0, x)$, сигмоидная функция: $sigmoid(x) = \frac{1}{1+e^{-x}}$. Ее результат показывает вероятность принадлежности элемента к определенному классу. Обычно модель относит элемент к классу, если эта вероятность больше или равна 0.5. Количество эпох – 40, $learning\_rate$ – 0.001, $batch\_size$ – 16.

## 4. Эксперименты

После реализации модуля предобработки данных, включающего в себя кодирование категориальных признаков, а также фильтрацию несущественных признаков, выбора моделей, их разработки и реализации, настало время проводить эксперименты. В экспериментах участвовали две модели: перцептрон и жадное дерево.

Для корректного проведения экспериментов, аналогично логике работы перцептрона, в жадном дереве было реализовано разделение на обучающую и тестовую выборки. Алгоритм работы данного модуля можно увидеть на рисунке 11:

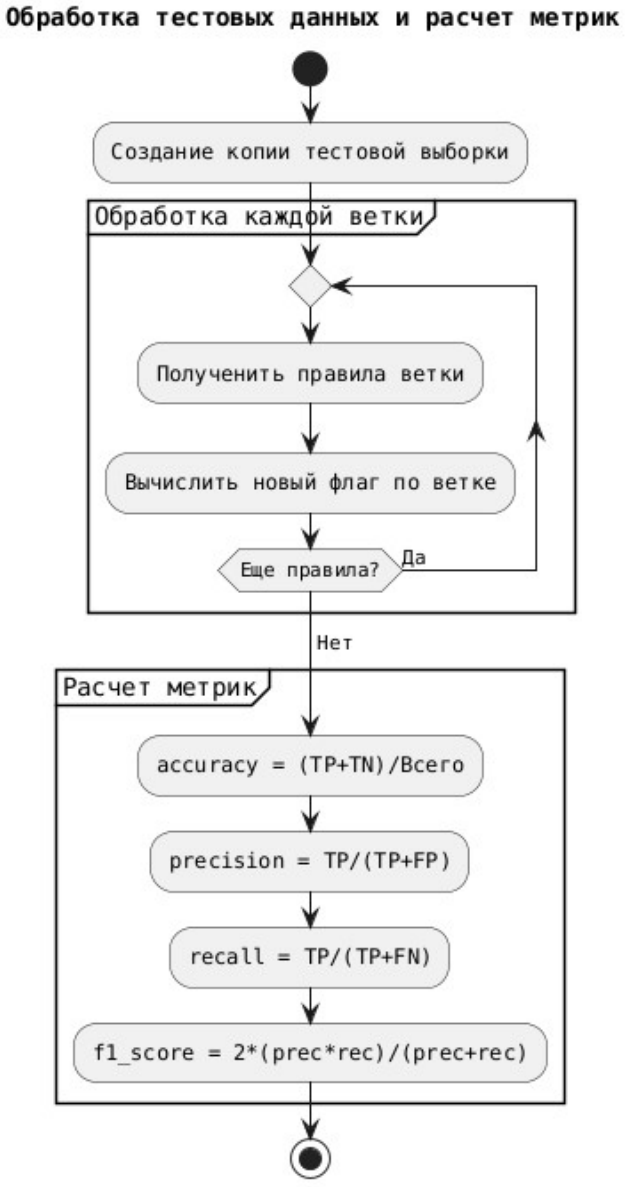


Рисунок 11 – Тестовый модуль жадного дерева

Все результаты, приведенные ниже, были получены на тестовых выборках моделей.

В качестве НД участвовали:

1) НД Titanic для первоначальной проверки работоспособности разрабатываемых алгоритмов с количеством строк 1000 и соотношением классов 38%-62%, позволяющий оценить качество классификации на сбалансированном НД;

2) банковский НД 1 – 1762 строки, отношение классов 65/1697 (3.7%-96.3%);

3) банковский НД 2 – 61006 строк, отношение классов 660/60346 (1%-99%).

### 4.1. Набор данных Titanic

Результаты на НД Titanic приведены на рисунке 12:

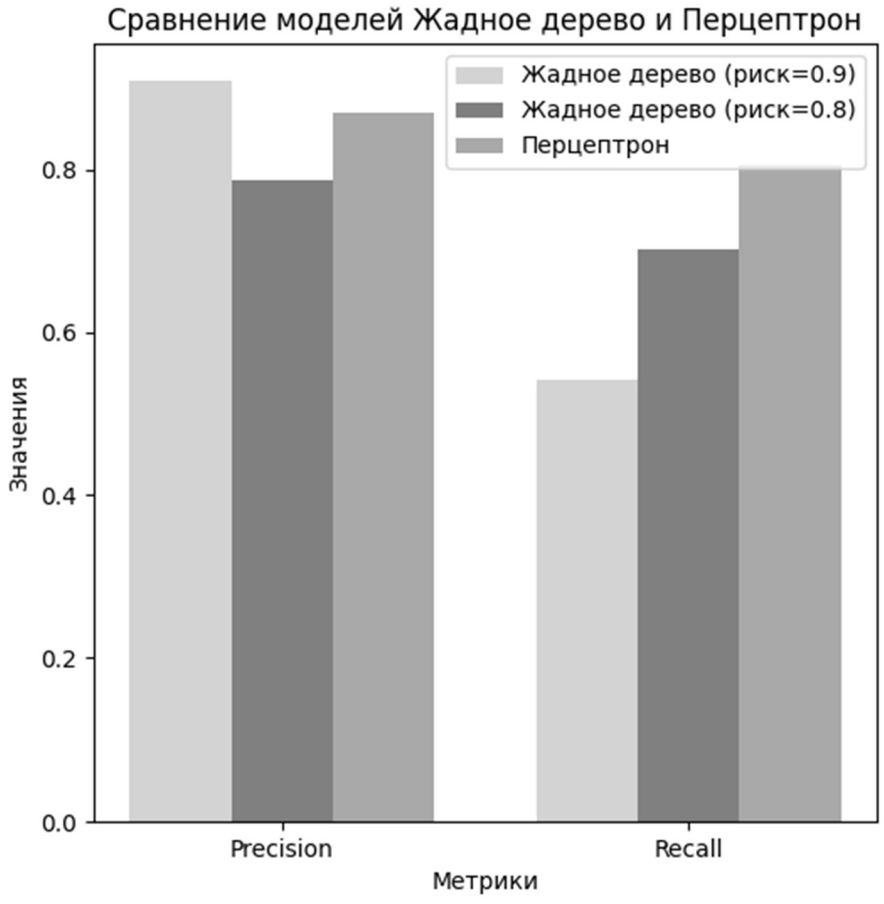


Рисунок 12 – Сравнение моделей на НД Titanic

Для более точного понимания приведена также таблица 1 этого эксперимента:

Таблица 1 – Сравнение моделей на НД Titanic

| Модели / Метрики | Перцептрон | Жадное дерево (риск=0.9) | Жадное дерево (риск=0.8) |
|---|---|---|---|
| Precision | 87.08% | 90.9% | 78.78% |
| Recall | 80.6% | 54.05% | 70.27% |
| F1-score | 83.71% | 67.8% | 74.29% |

Как видно из рисунка и таблицы, перцептрон показывает более высокие интегральные характеристики на рассматриваемом НД. Однако положительным для жадного дерева является тот факт, что при изменении задаваемого риска можно легко контролировать важнейшую метрику precision. Она является таковой, потому что, если говорить в терминологии банковского сектора, очень важно не столько отрезать большую часть просрочек, сколько отрезать действительно мошеннические заявки среди просроченных. Максимизировать эту метрику (конечно, в разумных пределах относительно recall) является первым приоритетом.

### 4.2. Набор данных 1

Результаты тестирования моделей на первом банковском НД приведены на рисунке 13 и в таблице 2:

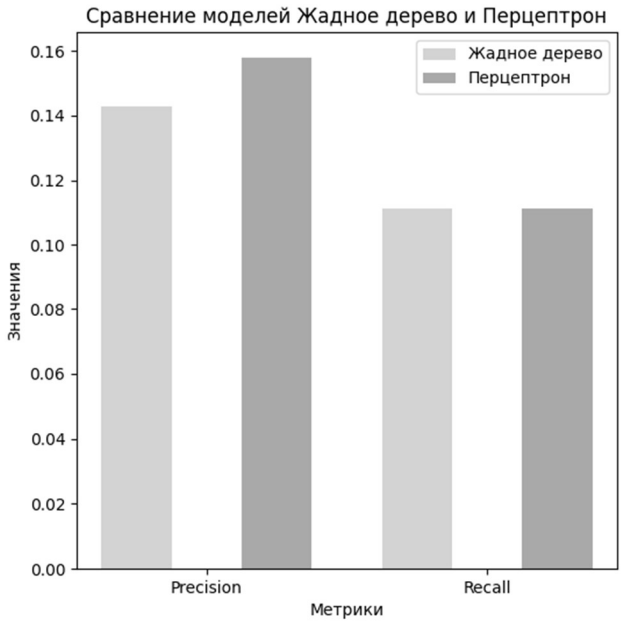


Рисунок 13 – Результаты моделей на НД 1

Таблица 2 – Результаты моделей на НД 1

| Модели / Метрики | Перцептрон | Жадное дерево |
|---|---|---|
| Precision | 15.78% | 14.28% |
| Recall | 11.11% | 11.11% |

Как видно из таблицы и рисунка, метрики моделей можно назвать похожими, но все же жадное дерево немного уступает перцептрону в precision. Важно отметить, что по сравнению с НД Titanic на обеих моделях результаты значительно хуже. Появилось предположение, что в НД 1 было слишком мало экземпляров первого класса. При разделении на выборки оставалось всего около 13 заявок. Для этого было предложено попробовать еще один НД.

### 4.3. Набор данных 2

В НД 2 был гораздо более сильно выражен дисбаланс классов даже относительно НД 1 (в 3.7 раза меньше заявок целевого класса), однако в то же время сама выборка размерами превосходила НД 1 почти в 35 раз. Таким образом, проблема малого количества заявок целевого класса в тестовой выборке исчезала, но более остро стояла проблема дисбаланса классов, поэтому на этом НД использовался oversampling. Результаты моделей показаны в таблицах 3 и 4:

Таблица 3 – Результаты жадного дерева на НД 2

| Oversampling | Риск | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| 0.15 | 0.6 | 97.73% | 12.63% | 16.31% | 14.24% |
| 0.15 | 0.5 | 95.07% | 7.56% | 29.08% | 12.01% |
| 0.15 | 0.4 | 89.98% | 5.07% | 43.26% | 9.08% |

Таблица 4 – Результаты перцептрона на НД 2

| Эпохи | Oversampling | Accuracy | Precision | Recall |
|---|---|---|---|---|
| 40 | 0.05 | 98.84% | 0.0% | 0.0% |
| 40 | 0.10 | 98.84% | 0.0% | 0.0% |

Продолжение таблицы 4

| Эпохи | Oversampling | Accuracy | Precision | Recall |
|---|---|---|---|---|
| 40 | 0.15 | 96.93% | 7.33% | 14.18% |
| 30 | 0.15 | 68.72% | 0.75% | 19.86% |
| 40 | 0.20 | 20.57% | 0.65% | 44.68% |

Из таблиц видно, что модель перцептрона уступает жадному дереву в предсказаниях. Лучший результат жадного дерева – precision = 12.63% и recall = 16.31%, при лучшем результате перцептрона – precision = 7.33% и recall=14.18%.

Кроме того, хорошо видно, как легко контролируется работа модели жадного дерева за счет изменения риска (при понижении риска уменьшается precision и повышается recall).

По полученным результатам было принято решение не пытаться интерпретировать перцептрон [5], так как он не улучшает работу модели априори интерпретируемого жадного дерева.

Важным является и то, что относительно работы моделей на НД 1 улучшения добиться не удалось. Результаты жадного дерева с precision = 15.78% и recall = 11.11% изменились на precision = 12.63% и recall = 16.31%. Было выдвинуто предположение о том, что результаты хуже относительно результатов на НД Titanic вследствие того, что просрочка как таковая еще не означает мошеннического замысла заемщика. Это значит, что просрочка чаще всего является выбросом, а не закономерным случаем, что делает невозможным их объединение в группу заявок с высокой долей риска. Для того, чтобы проверить, что выделяемые моделями заявки имеют похожую природу было решено провести дополнительное исследование по пересечению заявок. Если отсекаемые заявки обеих моделей будут по большей части идентичные, что при условии достаточно низкого recall будет маловероятно, то можно будет говорить о мошеннической природе этих заявок.

### 4.4. Доработка перцептрона

Прежде чем приступить к проверке гипотезы о похожей природе отсекаемых моделями перцептрона и жадного дерева заявок, нужно было доработать перцептрон, чтобы он показал сравнимый с жадным деревом результат, так как он либо не отсекал ничего вообще, что делало пересечение невозможным, либо отсекал, но со столь низкой точностью, что пересечение заявок становилось бессмысленным.

Новая структура перцептрона включала в себя 3 слоя, также увеличено количество эпох обучения до 100, добавлена валидация для раннего выхода из обучения, добавлена переменная class_weight для указания отношения классов, а также доработана переменная loss (добавлен параметр alpha, также отвечающий за дисбаланс классов и влияющий на обучение) [6]. Параметры class_weight и alpha влияют на обучение следующим образом (пример при alpha=0.75, class_weight={0:1, 1:5}):

1) $loss\_class\_1 = 5 * [0.75 * focal\_term]$ – усиление влияния ошибки в 3.75 раза;

2) $loss\_class\_0 = 1 * [0.25 * focal\_term]$ – ослабление влияния ошибки в 0.25 раза.

После доработки структуры перцептрона был запущен модуль перебора гиперпараметров (alpha менялся в пределах [0.5, 0.9] с шагом 0.1, gamma менялся в пределах [0.2, 0.3] с шагом 0.1, class_weight для 1 класса менялся в пределах [5, 15] с шагом 1). Процедура заняла 11 часов, в результате был сформирован НД с метриками для различных гиперпараметров, лучшие результаты приведены в таблице 5:

Таблица 5 – Лучшие результаты перцептрона

| alpha | gamma | class_weight | precision | recall | tp | tn | fp | fn |
|---|---|---|---|---|---|---|---|---|
| 0,6 | 3 | 13 | 17,00% | 1% | 1 | 12056 | 5 | 140 |
| 0,5 | 2 | 5 | 13,00% | 3% | 4 | 12034 | 27 | 137 |
| 0,6 | 2 | 8 | 5,00% | 1% | 2 | 12021 | 40 | 139 |

Продолжение таблицы 5

| alpha | gamma | class_weight | precision | recall | tp | tn | fp | fn |
|---|---|---|---|---|---|---|---|---|
| 0,8 | 3 | 7 | 2,00% | 4% | 5 | 11784 | 277 | 136 |
| 0,7 | 3 | 7 | 1,00% | 2% | 3 | 11857 | 204 | 138 |

Как видно из таблицы, удалось достичь относительно приемлемого precision, но recall все также остался низким. Эксперимент по пересечению все же было решено провести.

### 4.5. Пересечение заявок перцептрона и жадного дерева

Модуль пересечения на первом этапе представляет собой пометку в изначальном НД предсказаний перцептрона (рисунок 14):


Рисунок 14 – Пометка предсказаний перцептрона

Затем получившийся НД соединяется по идентификатору каждой строки с подобным НД с результатами жадного дерева.

После перезапуска перцептрона на параметрах (alpha=0.5, gamma=2, class_weight=5) удалось получить true positive (далее – tp)=5, false positive

(далее – fp)=15, т.е. precision=25%, но стоит отметить, что recall при этом остается маленьким ($5/136 \sim 0.04 = 4\%$).

Модуль пересечения же показал tp=5, fp=8 (precision=62.5%). Можно видеть, что логика выделения заявок совпадает достаточно сильно, поэтому выделенные моделями сегменты действительно можно отнести к сегментам высокого риска. Остальные просроченные заявки остаются непомеченными либо из-за случайной природы их появления, либо из-за несовершенства моделей.

# ЗАКЛЮЧЕНИЕ

Данная работа представляет собой попытку автоматизировать решение проблемы оценки полезности признаков, присылаемых сервисами, а, соответственно, и полезность самих этих сервисов. С помощью разработанной модели жадного дерева эта задачу можно решать, поэтому цель работы можно считать достигнутой. Вопрос о совершенстве данного подхода, безусловно, остается открытым. Существует как минимум еще один большой класс моделей сверточных нейронных сетей, которые могут показать лучшие результаты относительно разработанной модели и для которых также существуют алгоритмы интерпретации. По итогам работы были получены следующие выводы:

1) был разработан, внедрен и протестирован модуль предобработки данных, включающий в себя кодирование категориальных признаков, фильтрацию несущественных признаков и oversampling;

2) был разработан и протестирован собственный алгоритм жадного дерева;

3) разработанный алгоритм был сопоставлен с перцептроном в качестве конкурента на трех различных наборах данных;

4) был проведен эксперимент по пересечению заявок, внутри которого была показана похожая природа выделения заявок целевого класса.

Результатом работы можно считать модель жадного дерева, представляющую собой конкурентноспособное решение поставленной проблемы. Так как в качестве цели работы была прежде всего разработка интерпретируемой модели, в сравнении с не интерпретируемыми сложными нейронными сетями модель жадного дерева, безусловно, должна уступать в качестве предсказаний, несмотря на сравнимые характеристики с перцептроном. Поэтому в качестве дальнейших исследований стоит рассмотреть реализацию сверточной нейронной сети и ее интерпретацию.

За время выполнения выпускной квалификационной работы были приобретены следующие компетенции (таблица 6):

Таблица 6 – Компетенции

| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
|---|---|---|
| УК-1 | Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач | Анализ существующих методов машинного обучения и фильтрации признаков |
| УК-2 | Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений | Разработка алгоритмов фильтрации признаков и машинного обучения |
| УК-3 | Способен осуществлять социальное взаимодействие и реализовывать свою роль в команде | Согласование этапов разработки с научным руководителем |
| УК-4 | Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах) | Проведение разговоров с коллегами о банковских данных |
| УК-5 | Способен воспринимать межкультурное разнообразие общества в социально-историческом, этическом и философском контекстах | Забота о конфиденциальности персональных данных |
| УК-6 | Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни | Выполнение всех этапов проекта в заданные сроки |
| УК-7 | Способен поддерживать должный уровень физической подготовленности для обеспечения полноценной социальной и профессиональной деятельности | Занятия спортом во время написания выпускной квалификационной работы |

Продолжение таблицы 6

| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
|---|---|---|
| УК-8 | Способен создавать и поддерживать в повседневной жизни и в профессиональной деятельности безопасные условия жизнедеятельности для сохранения природной среды, обеспечения устойчивого развития общества, в том числе при угрозе и возникновении чрезвычайных ситуаций и военных конфликтов | Следил за временем, проведенным за написанием работы, чтобы не травмировать глаза. Выполнял технику безопасности по содержанию рабочего места |
| УК-9 | Способен принимать обоснованные экономические решения в различных областях жизнедеятельности | Работа помогает принимать правильное экономическое решение банку |
| УК-10 | Способен формировать нетерпимое отношение к проявлениям экстремизма, терроризма, коррупционному поведению, противодействовать им в профессиональной деятельности | Забота о конфиденциальности персональных данных |
| ОПК-1 | Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности | Применены знания по анализу метрик моделей |
| ОПК-2 | Способен применять компьютерные/суперкомпьютерные методы, современное программное обеспечение, в том числе отечественного происхождения, для решения задач профессиональной деятельности | Разработка велась на современной библиотеке языка python pandas |

Продолжение таблицы 6

| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
|---|---|---|
| ОПК-3 | Способен к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям | Разработан модуль пересечения результатов жадного дерева и перцептрона |
| ОПК-4 | Способен участвовать в разработке технической документации программных продуктов и комплексов с использованием стандартов, норм и правил, а также в управлении проектами создания информационных систем на стадиях жизненного цикла | Перед реализацией модуля пересечения была сформирована блок-схема разрабатываемого алгоритма |
| ОПК-5 | Способен инсталлировать и сопровождать программное обеспечение информационных систем и баз данных, в том числе отечественного происхождения, с учетом информационной безопасности | Часть работы велась на PyCharm, установленного на личный ноутбук |
| ОПК-6 | Способен понимать принципы работы современных информационных технологий и использовать их для решения задач профессиональной деятельности | Использование PostgreSql |
| ПК-1 | Проверка работоспособности и рефакторинг кода программного обеспечения | Разработка и тестирование модулей предобработки данных, oversampling и экспериментальных моделей (жадное дерево и перцептрон); отладка кода и оптимизация параметров моделей |

Продолжение таблицы 6

| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
|---|---|---|
| ПК-2 | Интеграция программных модулей и компонент и верификация выпусков программного продукта | Сбор и объединение различных источников данных, формирование финальных таблиц для обучения, интеграция методов oversampling и анализа результатов в рабочий код проекта |
| ПК-3 | Разработка требований и проектирование программного обеспечения | Формулирование критериев выбора целевого класса и структуры входных данных, определение гипотез для проведения экспериментов и построение плана тестирования моделей |
| ПК-4 | Оценка и выбор варианта архитектуры программного средства | Сравнительный анализ двух архитектур: жадное дерево и перцептрон, оценка их применимости для задачи классификации просрочек, обоснование выбора модели с учетом интерпретируемости и метрик качества |
| ПК-5 | Разработка тестовых случаев, проведение тестирования и исследование результатов | Тестирование моделей на различных наборах данных |
| ПК-6 | Обеспечение и оптимизация функционирования баз данных | Выгрузка признаков для последующей обработки наборов данных |
| ПК-7 | Обеспечение информационной безопасности на уровне базы данных | Конфиденциальность персональных данных |
| ПК-8 | Выполнение работ по созданию (модификации) и сопровождению информационных систем, автоматизирующих задачи организационного управления и бизнес-процессы | Разработка и тестирование моделей машинного обучения |

Продолжение таблицы 6

| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
|---|---|---|
| ПК-9 | Создание и сопровождение требований и технических заданий на разработку и модернизацию систем и подсистем малого и среднего масштаба и сложности | Формирование требований к разрабатываемым моделям |
| ПК-10 | Способен к коммуникации, восприятию информации, умению логически верно, аргументировано и ясно строить устную и письменную речь для решения профессиональных задач | Проведение разговоров с научным руководителем, составление документации выпускной квалификационной работы |
| ПК-11 | Способен использовать действующее законодательство и другие правовые документы в своей деятельности, демонстрировать готовность и стремление к совершенствованию и развитию общества на принципах гуманизма, свободы и демократии | Конфиденциальность персональных данных |

## СПИСОК ЛИТЕРАТУРЫ

1) 5 главных алгоритмов сэмплинга: Habr [Электронный ресурс]. – 2019. – URL: https://habr.com/ru/articles/461285/ (дата обращения 15.04.2025).

2) Дерево решений: что это простыми словами и как его создать: Синергия [Электронный ресурс]. – 2024. – URL: https://synergy.ru/akademiya/upravlenie/derevo_reshenij_chto_eto_prostyimi_slovami_i_kak_ego_sozdat (дата обращения 05.05.2025).

3) Entropy, information gain, and Gini impurity(Decision tree splitting criteria): machinelearningnuggets.com [Электронный ресурс] – 2024. – URL: https://www.machinelearningnuggets.com/splitting-criteria-in-decision-trees/ (дата обращения 07.05.2025).

4) What is Perceptron | The Simplest Artificial neural network: GeeksForGeeks [Электронный ресурс] – 2024. – URL: https://translated.turbopages.org/proxy_u/en-ru.ru.9c059ddb-6820be12-9f51b40e-74722d776562/https/www.geeksforgeeks.org/what-is-perceptron-the-simplest-artificial-neural-network/ (дата обращения 07.05.2025).

5) Rule Extraction Algorithm for Deep Neural Networks: A Review [Текст]: Tameru Hailesilassie. – (IJCSIS) International Journal of Computer Science and Information Security, 2016 – 6 с.

6) Анализ функций потерь и метрик в машинном обучении: ZENTYX [Электронный ресурс]. – 2024. – URL: https://zentyx.ru/posts/analiz-funktsij-poter-i-metrik-v-mashinnom-obuchenii/?ysclid=maaw2ebhp0188417713 (дата обращения 07.05.2025).

## ПРИЛОЖЕНИЕ

```
def manipulate_cat_features(data, non_categorial_features,
categorial_features):
    dumm = pd.get_dummies(data[categorial_features], columns =
categorial_features)
    new_features = list(dumm.columns)
    features = non_categorial_features+new_features
    data = pd.concat([data, dumm], axis=1)
    return data, features

# оверсэмплинг
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(sampling_strategy=0.1, random_state=42)
X_res, y_res = ros.fit_resample(X_train, y_train)


# Определение модели двуслойного перцептрона
model = keras.Sequential([
    # Первый скрытый слой с 128 нейронами и функцией
активации Leaky ReLU
    layers.Dense(64, use_bias=False,
input_shape=(X_train.shape[1],)),
    layers.LeakyReLU(),
    # Второй скрытый слой с 64 нейронами и функцией
активации Leaky ReLU
    layers.Dense(32, use_bias=False),
    layers.LeakyReLU(),
    # Выходной слой с одним нейроном и линейной функцией
активации
    layers.Dense(1)
])

# Добавляем метрики в модель
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.BinaryFocalCrossentropy(gamma=3.0),
    metrics=[
        'accuracy',
        Precision(),
        Recall(),
        AUC(),
        TruePositives(name='tp'),
        TrueNegatives(name='tn'),
        FalsePositives(name='fp'),
        FalseNegatives(name='fn')
    ]
)

def greedy_exit50proc(data, features, global_target_sum,
segment='нет', thresh=0.25, target='Survived', depth=1):
    res_df = pd.DataFrame({'Сегмент': [], 'action_list': [],
```

```
'Глубина': [], 'Отсекаем': [],
                            'Отсекаем таргет': [], 'Отсекаем
нетаргет': [], 'Доля таргета отсекаемых': [],
                            'Всего таргета': [], 'Доля отсекаемых
из таргета портфеля': []})
    df = data
    action_list = []
    # считаем просрочки и непросрочки по таргету
    cur_prosr = len(df[df[target] == 1])
    cur_neprosr = len(df[df[target] == 0])

    depth_it = 0

    # социнженерия
    # our_segments = segments+segments_industry+segments_regions

    while True:

        # переменные лучших резов для записи в action_list
        flag_find_better = False
        best_threshold = 0
        best_feature = 0
        best_prosr_del_neprosr = cur_prosr / cur_neprosr
        best_prosr = cur_prosr
        best_neprosr = cur_neprosr
        best_znak = '<='

        # условие выхода из цикла - непросрочки == 0 или
просрочки >= непросрочки
        if cur_neprosr == 0 or depth_it > depth:
            break

        # перебор всех фичей
        for col_name in features:
            # получение значений признака, перебор
            values = df[col_name]
            thresholds = values.unique()

            steps_len = 20
            if len(thresholds) > steps_len:
                step = (max(values) - min(values)) / steps_len
                thresholds = [min(values) + i * step for i in
range(steps_len + 1)]

            for threshold in thresholds:
                # смотрим левую и правую часть
                prosr1 = len(df[(df[col_name] <= threshold) &
(df[target] == 1)])
                neprosr1 = len(df[(df[col_name] <= threshold) &
(df[target] == 0)])
                prosr2 = len(df[(df[col_name] > threshold) &
(df[target] == 1)])
                neprosr2 = len(df[(df[col_name] > threshold) &
```

```
(df[target] == 0)])
                coef1 = prosr1 / neprosr1 if neprosr1 else
prosr1/10
                coef2 = prosr2 / neprosr2 if neprosr2 else
prosr2/10
                coef = max(coef1, coef2)
                prosr = prosr1 if abs(coef - coef1) < 0.01 else
prosr2
                neprosr = neprosr1 if abs(coef - coef1) < 0.01
else neprosr2
                znak = '<=' if abs(coef - coef1) < 0.01 else '>'
                if coef > best_prosr_del_neprosr and prosr >=
thresh * global_target_sum:
                    if not flag_find_better:
                        flag_find_better = True
                    best_threshold = threshold
                    best_prosr_del_neprosr = coef
                    best_feature = col_name
                    best_prosr = prosr
                    best_neprosr = neprosr
                    best_znak = znak
                    """
                    if (best_threshold==0 and
best_feature=='2.43'):
                        print(best_prosr, best_neprosr,
best_znak)
                    """
                    # print(best_znak, best_threshold,
best_feature, best_prosr, best_neprosr)
        if not flag_find_better:
            break
        # доп_условие на увеличение глубины (социнженерия)
        """
        if best_feature not in our_segments and best_feature not
in [row[0] for row in action_list]:
            depth+=1
        """
        depth_it += 1
        action_list.append([best_feature, best_threshold,
best_znak])
        it_list = action_list.copy()
        res_df = pd.concat([res_df, pd.DataFrame({'Сегмент':
[segment], 'action_list': [it_list],
                                                  'Глубина':
[len(action_list)],
                                                  'Отсекаем':
[best_neprosr + best_prosr],
                                                  'Отсекаем
таргет': [best_prosr], 'Отсекаем нетаргет': [best_neprosr],
                                                  'Всего
таргета': global_target_sum,
                                                  'Доля таргета
отсекаемых': [best_prosr / (best_prosr + best_neprosr)],
```

```
                                                    'Доля
отсекаемых из таргета портфеля': [
                                                        best_prosr
/ global_target_sum]})], axis=0)
        if best_znak == '<=':
            df = df[df[best_feature] <= best_threshold]
        else:
            df = df[df[best_feature] > best_threshold]
        # exec(f"""df =
df[df[best_feature]{best_znak}best_threshold]""")
        # print(len(df))
        cur_prosr = best_prosr
        cur_neprosr = best_neprosr
    return res_df


def steps_by_best_threshes(segments_dict, global_target_sum):
    df_results = pd.DataFrame({'Сегмент': [], 'action_list': [],
'Глубина': [], 'Отсекаем': [], 'Отсекаем таргет': [], 'Отсекаем
нетаргет': [], 'Доля таргета отсекаемых': [],'Всего таргета':
[], 'Доля отсекаемых из таргета портфеля': []})

    for segment_name, df in segments_dict.items():
        it_df = df
        it = 0
        while True:
            # подбор наилучшего порога итерации
            const_low = 0.05
            low = const_low
            high = 1
            res_df = pd.DataFrame()
            # ограничение для случая отсутствия нужного порога
            for _ in range(10):
                cur = (low + high) / 2
                res_df_it = greedy_exit50proc(it_df, features,
global_target_sum, segment=str(it) + ' ' + segment_name,
                                              thresh=cur, depth=3)
                flag_is_validate = branch_criteria(res_df_it)
                if flag_is_validate:
                    if high - low < 0.01:
                        break
                    res_df = res_df_it
                    low = cur
                else:
                    if high - low < 0.01 and low == const_low:
                        break
                    high = cur
                print(high, low)
            if len(res_df)==0:
                break
            else:
                action_list = res_df.iloc[-1]['action_list']
                # print(action_list)
```

```
                df_results = pd.concat([df_results, res_df])
                # s = 'it_df =
it_df[~('+'&'.join([f'(it_df["{lst[0]}"]{lst[2]+str(lst[1])})'
for lst in action_list])+')]'
                s = 'it_df[~(' +
'&'.join([f'(it_df["{lst[0]}"]{lst[2] + str(lst[1])})' for lst
in action_list]) + ')]'
                # it_df =
it_df[~('&'.join([f'(it_df["{lst[0]}"]{lst[2]+str(lst[1])})' for
lst in action_list])]
                it_df = eval(s)
                it += 1
    return df_results

def branch_criteria(res_df):
    risk = 0.9
    if len(res_df)!=0 and res_df.iloc[-1]['Доля таргета
отсекаемых']>=risk:
        return True
    return False


def steps_by_best_threshes(segments_dict, global_target_sum):
    df_results = pd.DataFrame({'Сегмент': [], 'action_list': [],
'Глубина': [], 'Отсекаем': [],
                               'Отсекаем таргет': [], 'Отсекаем
нетаргет': [], 'Доля таргета отсекаемых': [],
                               'Всего таргета': [], 'Доля
отсекаемых из таргета портфеля': []})

    for segment_name, df in segments_dict.items():
        it_df = df
        it = 0
        while True:
            # подбор наилучшего порога итерации
            const_low = 0.05
            low = const_low
            high = 1
            res_df = pd.DataFrame()
            # ограничение для случая отсутствия нужного порога
            for _ in range(10):
                cur = (low + high) / 2
                res_df_it = greedy_exit50proc(it_df, features,
global_target_sum, segment=str(it) + ' ' + segment_name,
                                              thresh=cur, depth=3)
                flag_is_validate = branch_criteria(res_df_it)
                if flag_is_validate:
                    if high - low < 0.01:
                        break
                    res_df = res_df_it
                    low = cur
                else:
                    if high - low < 0.01 and low == const_low:
```

```
                        break
                    high = cur
                print(high, low)
            if len(res_df)==0:
                break
            else:
                action_list = res_df.iloc[-1]['action_list']
                # print(action_list)
                df_results = pd.concat([df_results, res_df])
                # s = 'it_df =
it_df[~('+'&'.join([f'(it_df["{lst[0]}"]{lst[2]+str(lst[1])})'
for lst in action_list])+')]'
                s = 'it_df[~(' +
'&'.join([f'(it_df["{lst[0]}"]{lst[2] + str(lst[1])})' for lst
in action_list]) + ')]'
                # it_df =
it_df[~('&'.join([f'(it_df["{lst[0]}"]{lst[2]+str(lst[1])})' for
lst in action_list])]
                it_df = eval(s)
                it += 1
    return df_results

# считаем данные для тестовой выборки
def cut_test_df(df_results, test_df):
    it_df = test_df
    for i in range(len(df_results)):
        action_list = df_results.iloc[i]['action_list']
        s = '&'.join([f'(it_df["{lst[0]}"]{lst[2] +
str(lst[1])})' for lst in action_list])
        it_df[f'new_surv_{i}'] = eval(s)
    return it_df

# Структура перцептрона
# Масштабируем признаки
scaler = StandardScaler()
X_res = scaler.fit_transform(X_res)
X_test_scal = scaler.transform(X_test)

# Определение модели перцептрона
model = keras.Sequential([
    # Первый скрытый слой с 128 нейронами и функцией
активации Leaky ReLU
    layers.Dense(128, use_bias=False,
input_shape=(X_res.shape[1],)),
    layers.BatchNormalization(),  # Добавим BatchNorm для
стабилизации обучения
    layers.LeakyReLU(),
    layers.Dropout(0.3),  # Добавим Dropout для регуляризации

    # Второй скрытый слой с 64 нейронами и функцией
активации Leaky ReLU
    layers.Dense(64, use_bias=False),
    layers.BatchNormalization(),
```

```
    layers.LeakyReLU(),
    layers.Dropout(0.2),

    # Выходной слой с одним нейроном и сигмоидной активацией
    layers.Dense(1, activation='sigmoid')  # Изменим
на sigmoid для бинарной классификации
])

# Добавляем Early Stopping
early_stopping = EarlyStopping(
    monitor='val_loss',  # Мониторим PR-AUC (лучше для
дисбаланса)
    mode='min',            # Стремимся максимизировать метрику
    patience=20,           # Количество эпох без улучшения
    restore_best_weights=True,  # Восстановим лучшие веса
    verbose=1
)

# Добавляем метрики в модель
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss=tf.keras.losses.BinaryFocalCrossentropy(
        gamma=2.0
        ,alpha=0.5  # Больший вес для minority class
    ),
    metrics=[
        'accuracy',
        Precision(name='precision'),
        Recall(name='recall'),
        AUC(name='auc'),
        AUC(name='pr_auc', curve='PR'),  # PR-AUC важнее для
дисбаланса
        TruePositives(name='tp'),
        TrueNegatives(name='tn'),
        FalsePositives(name='fp'),
        FalseNegatives(name='fn')
    ]
)

# Обучаем модель с validation split и callback'ами
history = model.fit(
    X_res,
    y_res,
    epochs=100,  # Увеличим число эпох, так
как EarlyStopping остановит раньше
    batch_size=32,
    validation_split=0.2,  # Добавим validation split для
EarlyStopping
    callbacks=[early_stopping],
    class_weight={0: 1., 1: 5.},  # Веса классов
    verbose=1
)
```