НАЦИОНАЛЬНЫЙ БАНК КАЗАХСТАНА

АНАЛИЗ РИСКОВ ПОТРЕБИТЕЛЬСКИХ КРЕДИТОВ С ПОМОЩЬЮ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ

Департамент денежно-кредитной политики
Экономическое исследование №2021-4

Шалкар Байкулаков

Зангар Белгибаев

Экономические исследования и аналитические записки Национального Банка Республики Казахстан (далее – НБРК) предназначены для распространения результатов исследований НБРК, а также других научно-исследовательских работ сотрудников НБРК. Экономические исследования распространяются для стимулирования дискуссий. Мнения, высказанные в документе, выражают личную позицию автора и могут не совпадать с официальной позицией НБРК.

Анализ рисков потребительских кредитов с помощью алгоритмов машинного обучения

NBRK – WP – 2021 – 2

Анализ рисков потребительских кредитов с помощью алгоритмов машинного обучения

Байкулаков Шалкар[1]
Белгибаев Зангар[2]

Аннотация

Данное исследование представляет собой попытку оценки кредитоспособности физических лиц с помощью алгоритмов машинного обучения на основе данных, предоставляемых банками второго уровня Национальному Банку Республики Казахстан. Оценка кредитоспособности заемщиков позволяет НБРК исследовать качество выданных кредитов банками второго уровня и прогнозировать потенциальные системные риски.

В данном исследовании были применены два линейных и шесть нелинейных методов классификации (*линейные модели* - логистическая регрессия, стохастический градиентный спуск, и *нелинейные* - нейронные сети, k-ближайшие соседи (kNN), дерево решений (decision tree), случайный лес (random tree), XGBoost, наивный Байесовский классификатор (Naïve Bayes)) и сравнивались алгоритмы, основанные на правильности классификации (accuracy), точности (precision) и ряде других показателей. Нелинейные модели показывают более точные прогнозы по сравнению с линейными моделями. В частности, нелинейные модели, такие как случайный лес (random forest) и k-ближайшие соседи (kNN) на передискредитированных данных (oversampled data) продемонстрировали наиболее многообещающие результаты.

Ключевые слова: *потребительские кредиты, машинное обучение, банковское регулирование, стохастический градиентный спуск, логистическая регрессия, k-ближайшие соседи, классификатор случайных лесов, дерево решений, gaussian NB (Гауссовский наивный Байесовский классификатор), XGBoost, нейронные сети (многослойный персептрон)*

Классификация JEL: G21, G28, E37, E51

---

[1]Байкулаков Шалкар – Директор проектного офиса, Центр развития платёжных и финансовых технологий НБРК
E-mail: sh.baikulakov@payfintech.kz

[2]Зангар Белгибаев – Главный специалист-аналитик, Управление монетарного анализа, Департамент денежно-кредитной политики, НБРК
E-mail: zanggar.belgibayev@nationalbank.kz

Содержание

1. Введение

Расширение потребительских кредитов является основным фактором роста розничных кредитов в последние годы. Выдача потребительских кредитов может привести к экономическому росту в стране, но в то же время следует отметить, что чрезмерное финансирование банками может вызвать появление системных рисков. Быстрый рост потребительского кредитования можно объяснить появлением детальных данных о физических лицах и эффективных инструментов оценки кредитного риска, с одной стороны, а также стабилизацией экономической ситуации и ростом благосостояния населения, с другой.

Динамичное расширение потребительского кредитования несет в себе ряд системных рисков, связанных, прежде всего, с ростом уровня долговой нагрузки на отдельные слои населения. В случае падения реальных доходов населения банковская система может столкнуться с массовыми дефолтами по потребительским кредитам. Как следствие, это может привести к снижению совокупного спроса в экономике, что в итоге может негативно повлиять на состояние экономики.

На основе изучения исследовательских работ по применению алгоритмов машинного обучения, можно сделать вывод, что наиболее популярным методом анализа кредитных рисков потребительских кредитов на больших массивах данных является использование нелинейных моделей, таких как нейронные сети, k-ближайшие соседи, дерево решений, случайный лес, XGBoost, наивный Байесовский классификатор, а также такие линейные модели, как логистическая регрессия, стохастический градиентный спуск. Большинство авторов использовали данные кредитных бюро и банков второго уровня. Учитывая, что данное исследование проводилось на основе регуляторных данных, собранных центральным банком, могут быть некоторые расхождения в подходах, с возможными минимальными отличиями в результатах.

Первый раздел представляет собой обзор литературы, в котором рассматриваются аналогичные работы других авторов. **Во втором разделе** описывается методология исследования, а также список использованных метрик. **Далее следует раздел** обсуждения результатов, в котором авторы описывают результаты моделирования. **В заключительном разделе** работы описываются основные выводы исследования.

2. Обзор литературы

По результатам исследования Grier (2012) было выявлено, что для анализа потребительского кредитования необходимо учитывать 5 параметров. Первый параметр – это репутация заемщика. В настоящее время кредитные менеджеры имеют доступ к отчетам кредитного бюро, которые показывают кредитную историю потребителя до принятия решения о выдаче кредита. Второй фактор – это платежеспособность потребителя, связанная с его доходами и существующими финансовыми обязательствами. Капитал – третий параметр, указывающий на первоначальный взнос, который может себе позволить заемщик. Под четвертым фактором подразумевают внешние факторы, такие как состояние рынка труда и общие экономические условия. Последним параметром является залог.

Методы кредитного скоринга оценивают вероятность погашения кредита потребителем, используя информацию из отчета кредитного бюро и кредитной заявки (Grier, 2012). Скоринг приложений – это первый тип модели кредитного скоринга, который оценивает заявки новых потребителей на основе таких параметров, как капитал, мощность и т. д. Другая модель, такая как модель оценки поведения, оценивает платежеспособность потребителей, предупреждая о потенциальных просроченных счетах.

Одна из целей использования метода кредитного скоринга – снизить риски невыплат по выдаваемым кредитам в будущем. Помимо традиционных методов, банки недавно начали интегрировать алгоритмы машинного обучения (ML) в кредитный скоринг. Так, Henley and Hand (1996) сравнили методы машинного обучения, такие как метод классификации k-ближайшего соседа (kNN), с традиционными методами кредитного скоринга, такими как линейная, логистическая регрессия и дерево решений. Алгоритмы были протестированы на предмет риска безнадежных долгов, и лучшим методом был определен метод с наименьшим ожидаемым уровнем риска. Несмотря на нечувствительность классификации k-ближайшего соседа к параметрам, она дала лучший результат. Кроме того, для проведения оценки потребителя на предмет платежеспособности по кредиту требуется несколько секунд (Henley and Hand, 1996).

Далее, Addo, Gueran, and Hassani (2018) применили логистическую регрессию, классификатор случайного леса и повышения градиента, а также модель глубокого обучения для анализа кредитного риска компаний. Были использованы различные наборы данных, а затем было выбрано 10 наиболее важных параметров и те же методы использовались для сравнения результатов. Предполагается, что лучший алгоритм показывает наибольшую площадь под кривой (AUC) и наименьшую среднеквадратичную ошибку (RMSE). Бинарные классификаторы превзошли модели глубокого обучения, и лучшая производительность принадлежит классификатору, такому как градиентный бустинг.

Для оценки заявок на получение потребительского кредита также используются регрессионные модели машинного обучения. Munkhdalai et al. (2019) сравнили алгоритмы машинного обучения с такими моделями, как система кредитного рейтинга FICO. Данные Survey of Consumer Finances (SCF) были использованы в качестве набора данных, к которым были применены различные методы регрессии машинного обучения. При этом, глубокие нейронные сети и алгоритмы XGBoost показали более высокую точность. В результате определили, что кредитные убытки были бы ниже, если бы кредитные учреждения начали использовать модели машинного обучения с 2001 года. Кроме того, глубокие нейронные сети и алгоритм xgboost показали более высокую точность.

Brown and Mues (2012) протестировали пригодность и точность методов классификации, таких как логистическая регрессия, нейронная сеть, дерево решений, градиентный бустинг, метод опорных векторов с квадратичной функцией потерь (LS-SVM) и случайный лес, для несбалансированного набора данных о ссуде. К набору данных был применен метод недостаточной выборки, а затем дисбаланс набора данных постепенно увеличивался для оценки методов классификации машинного обучения. Результат показывает, что классификаторы случайного леса и градиента бустинга хорошо справляются с несбалансированными данными, в то время как дерево решений, квадратичный дискриминантный анализ (QDA) и k-ближайшие соседи (kNN) работают хуже, чем другие методы.

Помимо вышеперечисленных моделей, в машинном обучении также есть несколько алгоритмов. Baesens et al. (2003) протестировали метод опорных векторов на основе ядра (kernel-based SVM) и метод опорных векторов с квадратичной функцией потерь (LS-SVM), а также другие популярные классификаторы машинного обучения для реальных наборов данных кредитного скоринга, включая наборы данных финансовых учреждений Бенилюкса и Великобритании. Результат показывает, что классификатор нейронной сети и машины векторов поддержки на основе ядра работали очень хорошо. Автор также отметил хорошую производительность линейного дискриминантного анализа и логистической регрессии. 41 классификатор, включая новые методы оценки кредитоспособности, применяемые к одним и тем же наборам данных (Lesmann, Baesens, Seow, and Thomas, 2015). Результат исследования показывает, что искусственные нейронные сети (ИНС) работают лучше, чем экстремальные методы обучения (ELM), и случайный лес (RF) лучше, чем ротационный лес (RotFor). Однако эти методы не могут дать экономическую интерпретацию модели, и для достижения этой цели необходимо провести дальнейшие исследования. Классификатор случайного леса был выбран в качестве эталона, так как он может дать пояснительную информацию для фундаментального анализа.

Tsai and Chen (2010) разработали четыре гибридных метода машинного обучения для сравнения с простыми классификаторами. Гибридный алгоритм – это комбинация двух методов машинного обучения. В этом исследовании

были выбраны методы классификации и кластеризации, а также четыре различных метода. Результат показывает, что комбинация логистической регрессии (LR) и нейронных сетей показала самый высокий прогноз, в то время как «двойная кластеризация» была худшим алгоритмом.

В 2015 году участники реализовали XGBoost в 17 из 29 решений Kaggle. Алгоритм был реализован для прогнозирования продаж магазина, классификации веб-текста, прогнозирования поведения клиентов, классификации вредоносных программ (Chen & Guestrin, 2016). В этом исследовании алгоритм будет использоваться для кредитного анализа и сравниваться с другими методами.

В этом исследовании к набору данных были применены два линейных и шесть нелинейных методов классификации, которые будут описаны в следующем разделе. Алгоритмы сравнивались по правильности классификации (accuracy), точности (precision) и нескольким другим показателям.

3. Методология исследования и исходные данные

Для анализа портфелей потребительских кредитов казахстанских банков использовались алгоритмы машинного обучения. Данные получены из Кредитного регистра, формируемого на основе информации, предоставленной банками второго уровня Национальному Банку Казахстана (НБК). Данные очищены от кредитов с недостающими и ненадежными параметрами. Кроме того, из данных были удалены ссуды с просрочкой платежа не более 90 дней. Если просрочка платежа по кредиту превышает 90 дней, ссуда признается неработающей ссудой (NPL). Работающие кредиты не имеют просрочки платежа со стороны потребителя.

Таблица 1

Параметры данных для исследования

Параметры	Виды
Регионы	Нур-Султан, Алматы и 15 регионов
Валюта	Тенге, Рубль, долл. США и Евро
Тип займа	Наличные займы и кредитная карта
Объект кредитования	Потребительские и автокредиты
Пол	Мужчина и женщина
Гражданство	Резидент и нерезидент
Процентная ставка	От 0% до 56%
Сумма займа	От 10 000 до 15 млн тенге
Возраст	От 18 до 99 лет

Источник: Национальный Банк Республики Казахстан

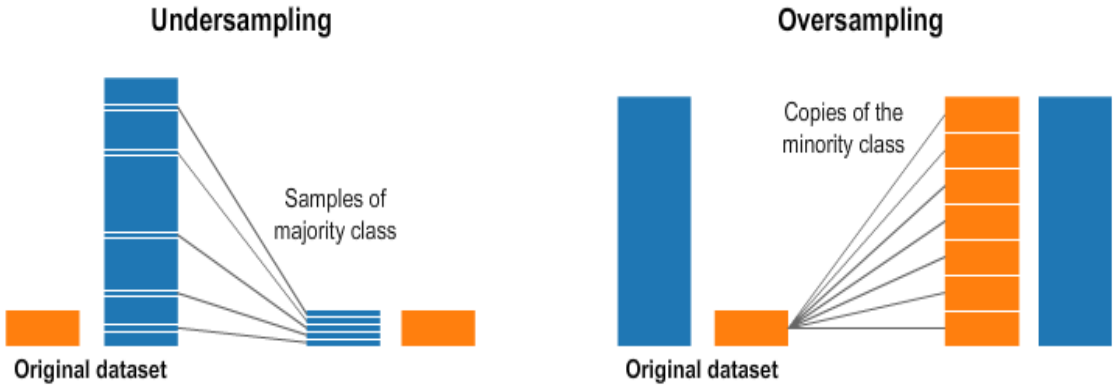
Стратегия сэмплинга для несбалансированных данных

Для повышения точности прикладных алгоритмов машинного обучения были использованы такие стратегии сэмплинга, как удаление примеров

мажоритарного класса (субдискретизация (undersampling)) и увеличение числа примеров миноритарного класса (передискретизация (oversampling)). Поскольку количество хороших ссуд было значительно больше, чем проблемных ссуд, данные были сбалансированы, и было проведено сравнение двух методов.

Рисунок 1

Методы ресэмплинга для несбалансированных данных



Источник: Alencar, 2017

Случайное удаление примеров мажоритарного класса (случайная субдискретизация (random undersampling)) и случайное дублирование примеров миноритарного класса (случайная передискретизация (random oversampling)) – самые популярные и простые стратегии сэмплинга. Первая стратегия создает новый класс с равным размером класса меньшинства, взятого из класса большинства, тогда как вторая стратегия дублирует класс меньшинства несколько раз, пока длина классов большинства и меньшинства не станет равной. Недостатком случайного удаления примеров мажоритарного класса (random undersampling) является потеря информации. Однако это единственная подходящая стратегия недостаточной выборки для набора данных, состоящего из категориальных и числовых данных. Так как, случайное дублирование примеров миноритарного класса (random oversampling) приводит к переобучению ввиду того, что он копирует класс меньшинства (Alencar, 2017), был выбран другой подходящий метод сэмплинга.

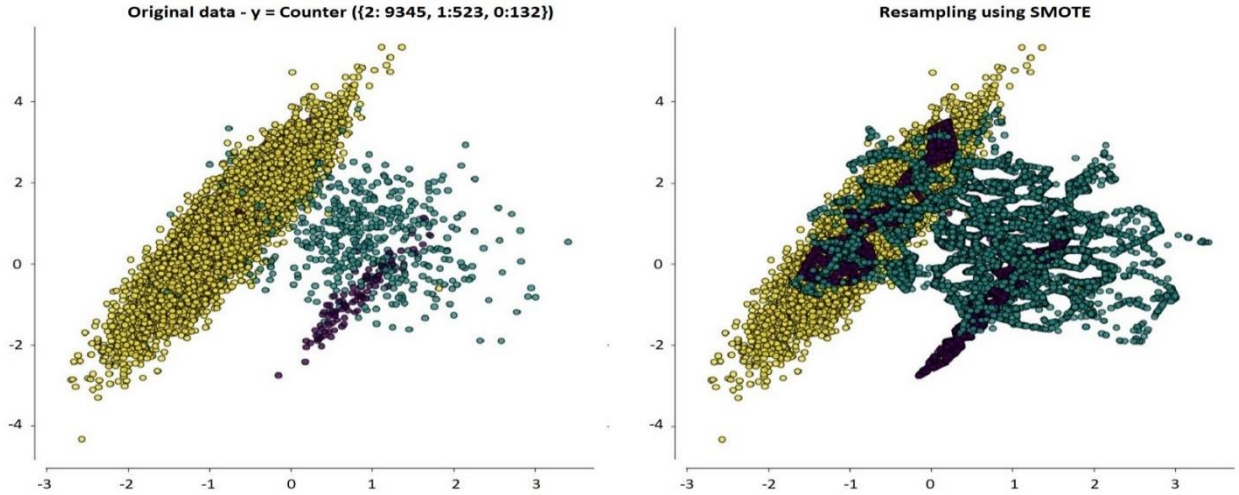
SMOTE (метод синтетической передискретизации меньшинства) – еще один популярный метод сэмплинга, который рисует новые выборки классов меньшинств, используя существующие данные. Он проводит линии между примером и ближайшими 5 соседями и создает новый искусственный образец вдоль этой линии. График 2 иллюстрирует, как создавались новые образцы, тем самым предоставляя дополнительную информацию для модели машинного обучения (Brownlee, 2020).

SMOTE нельзя применить к вышеупомянутому набору данных, поскольку он не применим к категориальным переменным. Поэтому в качестве стратегии сэмплинга выбран метод синтетической SMOTE-NC. Он

работает как метод SMOTE для непрерывного набора данных, а категориальной переменной новой выборки является значение большинства k-ближайших соседей (Chawla, Bowyer, Hall and Kegelmeyer, 2002).

Рисунок 2

Оверсемплинг миноритарного класса с использованием SMOTE



Источник: Lemaitre, Nogueira, Oliveira and Aridas, n.d.

Предварительная обработка (Preprocessing)

Предварительная обработка – важная задача, позволяющая сделать данные применимыми. Шесть параметров набора данных являются категориальными данными, и их необходимо преобразовать в числовые с помощью кодировщика. Библиотека Scikit-learn (Python) предоставляет методы, которые преобразуют дискретные данные в простой числовой массив. В нем также есть модели, которые разделяют данные на наборы данных для обучения и тестирования (du Boisberranger, n.d.).

Следующая важная часть предварительной обработки является шкалирование данных. Алгоритмы машинного обучения обычно используют евклидову меру расстояния. Таким образом они становятся чувствительными к величине параметров. Например, алгоритмы игнорируют другие параметры в наборе данных из-за больших значений суммы кредита. Следовательно, для нормализации набора данных требуется масштабирование функций. Это также снижает стоимость: методы кредитного скоринга быстрее анализируют нормализованный набор данных. Вышеупомянутая библиотека Python предоставляет стандартные методы шкалирования, которые нормализуют набор данных с помощью формулы:

$$z = \frac{x-\mu}{\sigma}$$ (Boisberranger et al., n.d.)

где, μ - среднее значение и σ является стандартной девиации данных.

Метод главных компонентов (Principal component analysis)

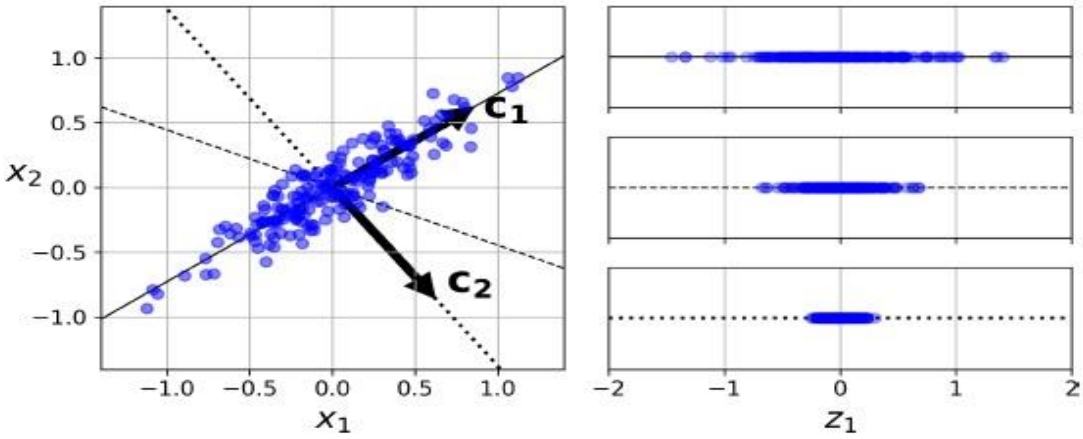
Метод главных компонентов (PCA) – один из хорошо известных методов уменьшения размерности. Это помогает уменьшить размер набора

данных с минимальной потерей информации (Mueller & Guido, 2017). Следовательно, это уменьшит время вычисления алгоритма.

Алгоритм выбирает правильную гиперплоскость и проецирует данные на эту гиперплоскость. После прогноза дисперсия данных вычисляется на основе каждой новой оси, называемой главным компонентом. Первый компонент сохраняет максимальную дисперсию, а последний компонент – наименьшую дисперсию.

Рисунок 3

Анализ главных компонентов

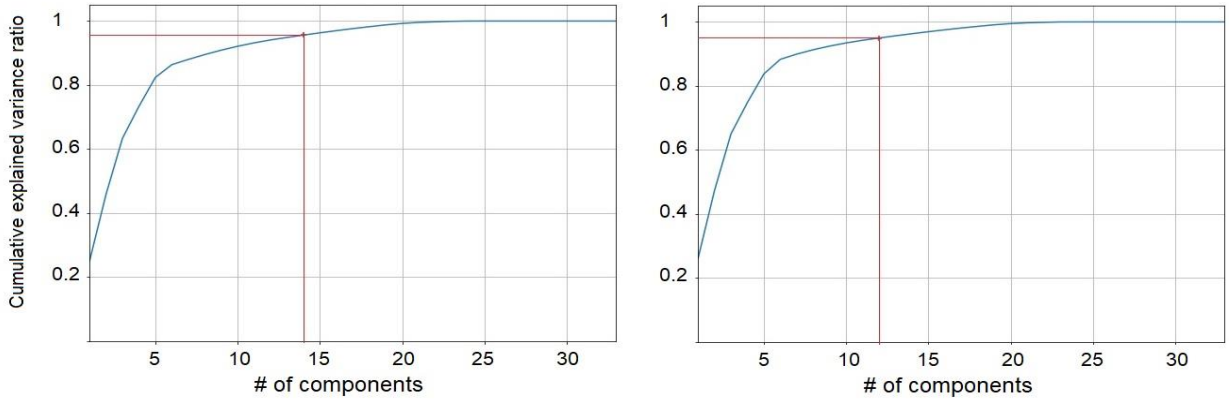


Источник: Geron, 2017

Метод главных компонентов выполняется для обоих сбалансированных наборов данных. Результат субдискретированных данных показывает, что первых 14 компонентов (столбцов) достаточно, чтобы охватывать 95% общей дисперсии набора данных, а остальные 19 компонентов можно удалить. Достаточно использовать только 12 компонентов набора передискретированных данных, чтобы охватывать 95% общей дисперсии.

Рисунок 4

Коэффициент кумулятивной дисперсии анализа главных компонентов



Источник: составлено авторами

В этом исследовании к набору данных были применены 2 линейных и 6 нелинейных методов машинного обучения. Основная цель – выяснить, какой

алгоритм превосходит другие. Кроме того, данное исследование помогает сравнивать и противопоставлять линейные и нелинейные алгоритмы.

Возможность обработки большого набора данных – главный критерий выбора алгоритма. Поэтому известные алгоритмы, такие как линейный классификатор и классификатор опорных векторов (SVC), в исследовании не рассматривались. Для целей анализа были использованы логистическая регрессия, классификатор стохастического градиентного спуска (SGD), наивный байесовский классификатор, k-ближайшие соседи (kNN), дерево решений (decision tree), случайный лес (random tree), классификатор многослойного персептрона (MLP) (классификатор нейронной сети) и XGBoost, и обсуждались результаты. Некоторые гиперпараметры нелинейных алгоритмов не будут учитываться из-за невозможности обработки больших наборов данных.

Логистическая регрессия (Logistic Regression)

Несмотря на название, логистическая регрессия используется для классификации. Алгоритм рассчитывает вероятность на основе обучающего набора данных по данной формуле:

$$P(y = 1|x) = \frac{1}{1+e^{-(w_0+\boldsymbol{W^T}x)}}$$ (Baesens et al., 2003)

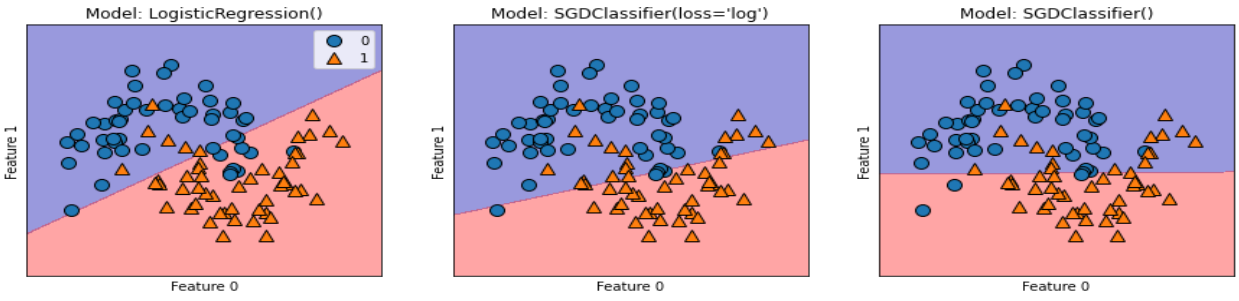
где x - входные данные, w_0 – скалярный вектор пересечения, а W - вектор параметров. Если вероятность больше 50%, входные данные классифицируются как положительные.

Для повышения точности были настроены значимые гиперпараметры, такие как параметры решателя (solver) и регуляризации. Решатель «Liblinear» обычно используется для небольшого набора данных, тогда как «sag» и «saga» – лучший выбор для анализа более крупных данных, поскольку он требует меньше времени для вычислений (du Boisberranger et al., N.d.).

Штраф (penalty) – это гиперпараметр регуляризации, используемый при наложении штрафа, и он имеет тесную связь с решателем. Решатели «Newton-cg», «sag» и «lbfgs» поддерживают только штрафы «l2», в то время как решатель «saga» поддерживает штраф «elasticnet». «C» – это величина, обратная силе регуляризации, которая должна быть положительной. Меньшее значение «C» указывает на более сильную регуляризацию (du Boisberranger et al., N.d.).

Рисунок 5

Модели линейной классификации



Источник: составлено авторами

Стохастический градиентный спуск (Stochastic Gradient Descent (SGD))

Классификатор SGD – один из эффективных методов линейной классификации, применяемых к большим наборам данных. Алгоритм использует процедуру обучения SGD первого порядка. Параметр метода обновляется итеративно на обучающих примерах по данной формуле:

$$\omega = \omega - \eta[\alpha \frac{\partial R(\omega)}{\partial \omega} + \frac{\partial L(\omega^T x_i + b, y_i)}{\partial \omega}]$$ (du Boisberranger et al., n.d.)

Где α - гиперпараметр, который контролирует силу регуляризации, R - выражение регуляризации, который снижает сложность модели. L - функция потерь, которая измеряет соответствие модели, ŋ - скорость обучения, а b - точка пересечения, которая обновляется аналогичным образом без регуляризации.

Градиентный спуск – один из важных алгоритмов, используемых для минимизации функции стоимости (Fuchs, 2019). В целом, существует три популярных типа градиентного спуска:

1. Общий градиентный спуск (Batch gradient descent);
2. Стохастический градиентный спуск (Stochastic gradient descent);
3. Градиентный спуск «mini-batch» (Mini-batch gradient descent).

Общий градиентный спуск вычисляет частную производную функции стоимости алгоритма с учетом его параметров. Другими словами, алгоритм обнаруживает, как различные значения параметров модели влияют на функцию стоимости алгоритма. Недостатком является то, что он использует целые обучающие данные для их вычисления на каждом этапе. Следовательно, алгоритм не может обрабатывать большие наборы данных (Geron, 2019).

Стохастический градиентный спуск решает эту проблему, поскольку он выбирает случайные экземпляры обучающих данных и вычисляет градиентный спуск на основе этого единственного экземпляра. С другой стороны, пакетный градиентный спуск будет плавно уменьшать функцию стоимости, в то время как она будет колебаться вверх и вниз, пока алгоритм не остановится. Алгоритм не может дать оптимальные значения параметров (Geron, 2019).

Градиентный спуск «mini-batch» берет небольшую выборку из обучающих данных и вычисляет алгоритм общего градиентного спуска. Следовательно, результат вычисления функции стоимости менее ошибочен по сравнению со стохастическим градиентным спуском (Geron, 2019).

Метод имеет много гиперпараметров, что делает его очень сложным. Важные параметры, такие как функция потерь (loss function) и параметры регуляризации, такие как штраф (penalty) и альфа (alpha), учитываются в процессе настройки модели. Метод с функцией потерь «Hinge» работает как линейная SVM, а с функцией «log» оперирует как логистическая регрессия (du Boisberranger et al., N.d.).

Алгоритм очень чувствителен к шкалированию. Однако его можно легко реализовать, и он очень эффективен, особенно для больших наборов данных

(du Boisberranger et al., N.d.). Кроме того, он продолжает работать без сохранения записи в ОЗУ (Fuchs, 2019).

Наивный Байесовский классификатор (Naïve Bayes classifier)

В общем, классификатор Наивного Байеса (NB) основан на теореме Байеса, предполагающей независимость каждого параметра:

$$P(y|x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1,\dots,x_n)}$$ (du Boisberranger et al., n.d.)

Есть несколько типов наивных Байесовских классификаторов. В этом исследовании Гауссовский наивный Байесовский классификатор (Gaussian Naïve Bayes (GaussianNB)) используется для оценки вероятности признаков на основе данной формулы:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}}$$ (du Boisberranger et al., n.d.)

где среднее значение и стандартное отклонение оцениваются через максимальное правдоподобие.

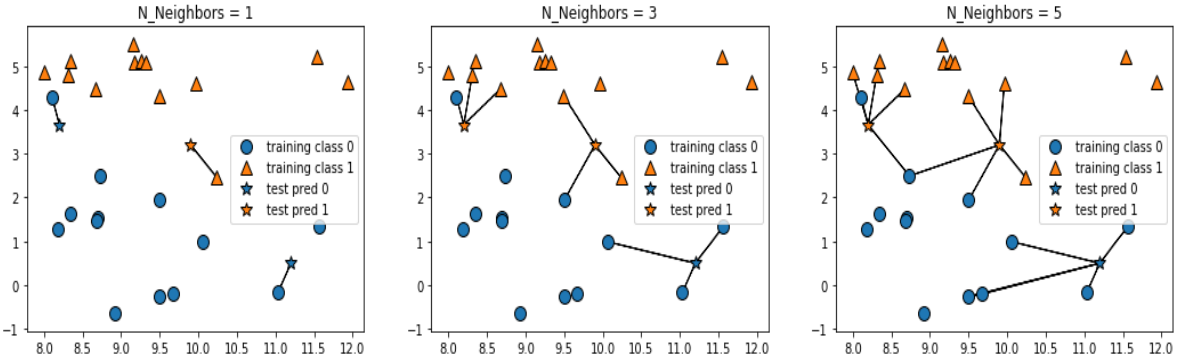
k-ближайшие соседи (kth Nearest Neighbors (kNN))

k-ближайшие соседи – один из простейших алгоритмов машинного обучения. Количество ближайших соседей (k) является основным параметром в этом алгоритме. Результат новых данных будет идентифицирован на основе большинства результатов ближайших соседей. Большое значение k подавит влияние шума, но будет на границе классификации (du Boisberranger et al., N.d.). Расстояние измеряется по формуле Евклидова расстояния:

$$d(x_i, x_j) = \|x_i - x_j\| = \left[(x_i - x_j)^T (x_i - x_j)\right]^{1/2}$$ (Brown & Mues, 2012)

Рисунок 6

Классификационная модель k ближайших соседей



Источник: Mglearn library (Mueller & Guido, 2017)

Дерево решений (Decision Tree classifier)

Дерево решений является простым алгоритмом, который прогнозирует целевой результат путем применения правил принятия решений на основе данных функций. Сильной стороной метода является способность

обрабатывать числовые, категориальные характеристики и задачи с несколькими выходами. Еще одно преимущество – простота модели. Но модель может создавать чрезмерно сложные деревья, которые приводят к переобучению модели (du Boisberranger et al., N.d.).

Критерий и максимальная глубина дерева учитываются в процессе настройки модели. Первый параметр – это функция, которая измеряет качество разделения. «Gini» (Gini impurity) и «entropy» (information gain) - два варианта выбора для функции критериев (du Boisberranger et al., N.d.).

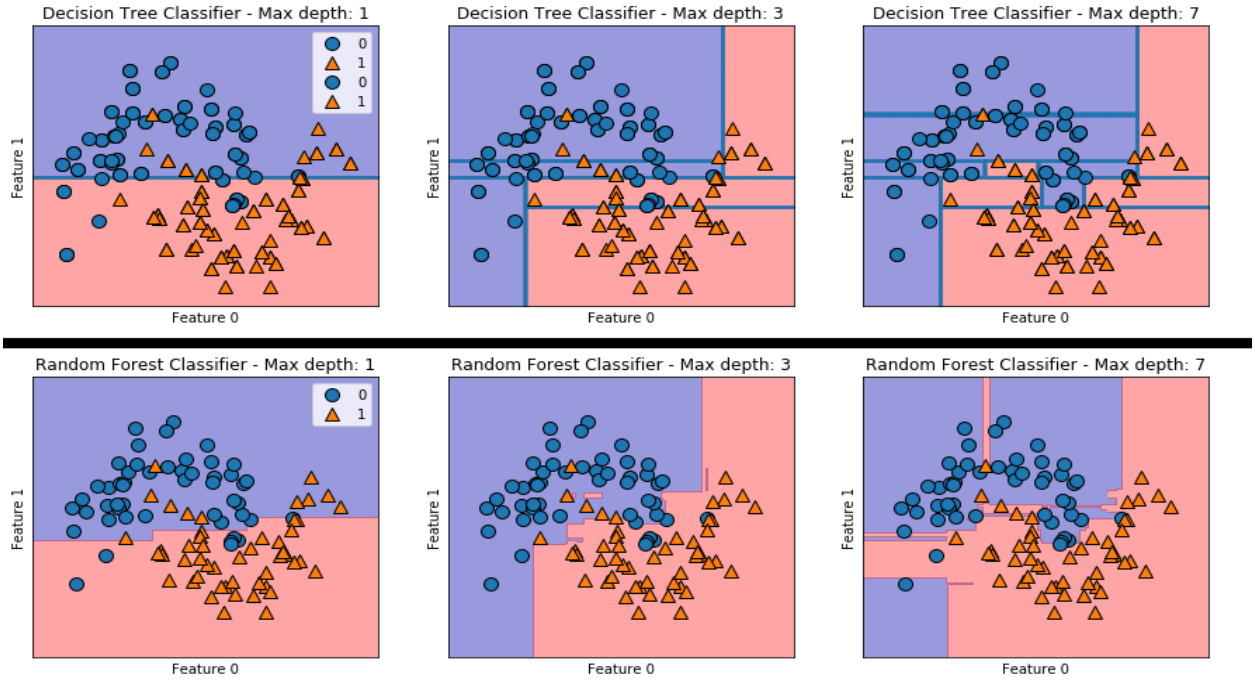
Случайный лес (Random Forest Classifier)

Классификатор случайного леса – это метод классификации ансамбля, который имеет все гиперпараметры классификатора дерева решений. Разница между двумя моделями заключается в том, что первая ищет лучшую функцию среди подмножества функций, а вторая ищет лучшую функцию при разбиении узла. Таким образом, случайный классификатор леса приводит к большему дереву (Geron, 2019).

Классификатор дерева решений показывает более высокую дисперсию, что приводит к переобучению. Классификатор случайных лесов объединяет разнообразные деревья, тем самым добиваясь уменьшения дисперсии. Впоследствии он создает лучшую модель, чем классификатор дерева решений (du Boisberranger et al., N.d.).

Рисунок 7

Дерево решений и классификатор случайного леса в наборе данных scikit-learn



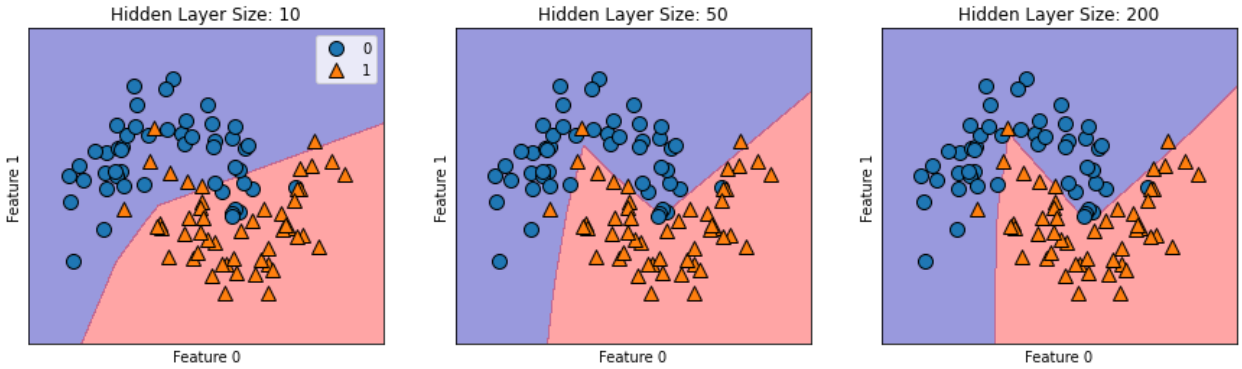
Источник: составлено авторами

Нейронные сети (Многослойный персептрон)

Нейронная сеть – это модель, вдохновленная структурой человеческого мозга. Основная структура алгоритма состоит из входных, выходных и скрытых слоев. Размер скрытых слоев – важный гиперпараметр алгоритма. На каждом уровне есть узлы, содержащие номер, и каждый узел получает сигнал от каждого узла предыдущего уровня и отправляет сигнал узлам следующего уровня. У каждого сигнала есть вес и смещение, на которое вход не влияет (Hansen, 2019).

Рисунок 8

Классификатор нейронной сети



Источник: составлено авторами

Одной из важных особенностей нейронной сети является алгоритм градиентного спуска. Он используется для минимизации отклонения между выводом и расчетным значением вывода. Нейронная сеть использует обратное распространение для вычисления градиентов. Затем он обновляет все смещения и корректирует веса всех узлов, начиная с вывода (Hansen, 2019).

XGBoost

Экстремальный бустинг градиента (XGBoost) – один из широко используемых алгоритмов. Это реализация деревьев решений с градиентным усилением. Скорость выполнения и производительность модели являются преимуществами перед другими алгоритмами повышения градиента (Brownlee, 2016).

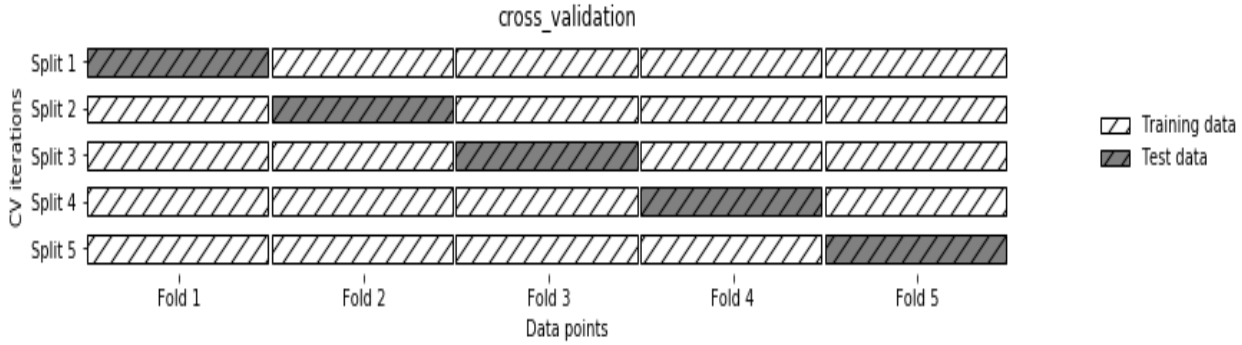
Преимущество метода – масштабируемость. Он использует новый алгоритм изучения дерева для обработки разреженных данных и может автоматически обрабатывать отсутствующие значения. Еще одна важная особенность метода – возможность обрезки деревьев, для более глубоких, оптимизированных деревьев. Параллельная и распределенная обработка данных делает его одним из самых быстрых алгоритмов. Алгоритм использует вычисления вне оперативной памяти, чтоб облегчает и ускоряет обработку данных. Следовательно, данный метод является наиболее оптимальным для обработки больших объемов данных. (Chen & Guestrin, 2016).

Кросс-валидация (Cross-validation)

Перекрестная проверка – это статистический метод, используемый для измерения производительности методов на основе набора данных обучения и тестирования. Часто используемой версией перекрестной проверки является перекрестная проверка в k-кратном размере, где количество складок составляет 5 или 10 (Mueller & Guido, 2017). В исследовании использовалась 5-кратная перекрестная проверка, как показано на Графике 9. Следовательно, набор данных делится на пять равных частей. Если первая часть данных используется в качестве тестового набора, остальные являются обучающей выборкой. То есть, цель состоит в том, чтобы проанализировать влияние наборов данных на точность модели на основе метрик.

Рисунок 9

Пятикратная перекрестная валидация



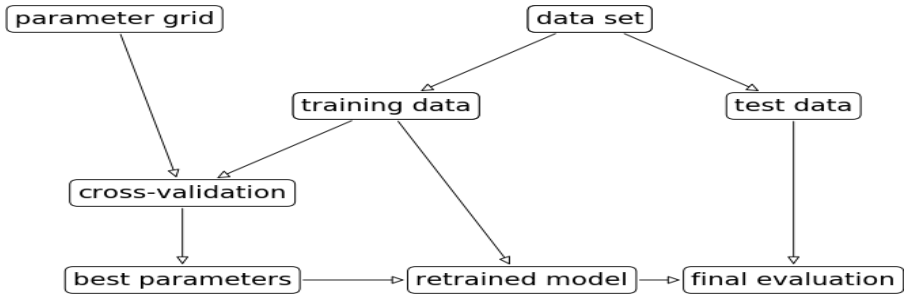
Источник: Mglearn library (Mueller & Guido, 2017)

Перебор по сетке (Grid Search)

Перебор по сетке – это заключительный этап построения модели. На этом этапе настраивается выбранная модель машинного обучения. Сетка параметров – это первый шаг процесса настройки, который включает набор всех возможных гиперпараметров алгоритмов. Затем модель с другим набором гиперпараметров применяется к обучающим и тестовым данным. Этот шаг помогает определить набор параметров, по которым модель получает наивысший показатель. Модель будет реконструирована с использованием набора лучших гиперпараметров (Mueller & Guido, 2017).

Рисунок 10

Grid Search



Источник: Mglearn library (Mueller & Guido, 2017)

Выбор гиперпараметров для настройки модели является значимым вопросом на данном этапе. Учитывая, что перебор параметров занимает много времени, сетка параметров была построена на основе наиболее важных параметров. Например, единственное количество ближайших соседей было параметром, выбранным для настройки модели kNN. Также были использованы сетки параметров, используемые на Kaggle (веб-сайт для специалистов по обработке данных).

Метрики (Metrics)

В машинном обучении алгоритмы регрессии, кластеризации и классификации используют разные показатели производительности. Алгоритмы классификации также имеют разные метрики для двоичной и мультиклассовой классификации. В данном исследовании были использованы 6 показателей эффективности моделей:

1. Правильность классификации (Accuracy score)
2. Точность (Precision score)
3. Полнота (Recall score)
4. F-мера (F1 score)
5. Мера Жаккара (Jaccard score)
6. Площадь под кривой ошибок (The area under the receiving operating characteristic (ROC) curve)
7. Доля ошибок второго рода (Type 2 error percentage)

Матрица ошибок (Confusion matrix)

Матрица ошибок – это матричная таблица, используемая для оценки эффективности классификатора. Обычно он указывает количество правильных и неправильных результатов алгоритма классификации. Он также предоставляет информацию об ошибках первого и второго рода, которые будут объяснены в этом разделе.

Таблица 2

Confusion matrix

	Actual class (observation)	
Predicted class (expectation)	TP (true positive) Correct result	FP (false positive) Unexpected result
	FN (false negative) Missing result	TN (true negative) Correct absence of result

Источник: Binary classification (du Boisberranger et al., n.d.)

1. Правильность классификации (Accuracy score)

Accuracy - самый важный показатель в исследовании. Он также используется в качестве показателя оценки для настройки модели при поиске по сетке. Метрика указывает долю правильных результатов модели:

$$Accuracy(y_{true}, y_{pred}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(if\ y_{true} = y_{pred})(du\ Boisberranger\ et\ al., n.d.)$$

$$= \frac{TP + TN}{TP + FP + FN + TN}\ (Geron, 2019;\ Mueller\ \&\ Guido, 2017)$$

2. Точность (Precision score)

Показатель точности – это количество правильных положительных прогнозов, деленное на количество общих положительных прогнозов.

$$Precision = \frac{TP}{TP + FP}\ (Geron, 2019; Mueller\ \&\ Guido, 2017)$$

3. Полнота (Recall score)

Полнота – это количество правильных положительных прогнозов, деленное на количество реальных положительных результатов.

$$Recall = \frac{TP}{TP + FN}\ (Geron, 2019; Mueller\ \&\ Guido, 2017)$$

4. F-мера (F1 score)

Точность и полнота – важные показатели. Однако ни один из них по отдельности не даст полной картины. F-мера - еще одна метрика, используемая для двоичной классификации, и это гармоническое среднее значение точности и полноты (Geron, 2019; Mueller & Guido; 2017).

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \times \frac{precision \times recall}{precision + recall} = \frac{TP}{TP + \frac{FN + FP}{2}}\ (Geron, 2019)$$

5. Мера Жаккара (Jaccard similarity coefficient score)

Мера Жаккара (JSC) – это пересечение фактического и прогнозируемого выхода для их объединения.

$$JSC = \frac{TP}{TP + FP + FN} = \frac{F_1}{2 - F_1}\ (Labatut\ \&\ Cherifi, 2011)$$

6. Площадь под кривой ошибок (Area under ROC curve)

Кривая ROC является важным инструментом для анализа производительности двоичных классификаторов. Это линейная кривая, которая отображает процентную ставку TP относительно ставки FP. Площадь под этой кривой - еще один способ сравнения классификаторов (Geron, 2019).

Значение этих показателей колеблется от 0 до 1: чем выше значение, тем больше подходит модель для анализа потребительского кредитования. Кроме того, эти метрики имеют очень тесную взаимосвязь.

7. Доля ошибок второго рода (Type 2 percentage)

Классификаторы машинного обучения как статистические модели также имеют ошибки первого и второго рода. Ошибка первого рода также известна как ложноположительный результат (FP) (Schmarzo, 2018). Например, один из коммерческих банков второго уровня внедряет модели кредитного скоринга. Модель отклоняет выдачу ссуды потребителю, поскольку заявляет, что в будущем она станет неработающей ссудой. Однако, на самом деле, потребитель способен полностью погасить ссуду. В этой ситуации банки отказываются выпускать долг, но это не окажет существенного влияния на ликвидность и платежеспособность банка.

Ложноотрицательные (FN) – это еще один тип ошибок, который также известен как ошибка второго рода (Schmarzo, 2018). Предположим, что банк выдает долг потребителю на основе результатов машинного обучения, в которых утверждается, что потребитель обязательно выплатит его. В действительности ссуда становится неработающей, что отрицательно сказывается на ликвидности банка. Более высокий процент ошибки 2-го рода показывает, насколько хуже модель.

4. Обсуждение полученных результатов

Как обсуждалось ранее, методы случайного удаления мажоритарного класса и случайного дублирования миноритарного класса - не лучшая стратегия сэмплинга. Первая приводит к потере информации, вторая - к проблемам переобучения. Кроме того, случайное удаление мажоритарного класса было единственным вариантом адаптации нормативных данных в данном исследовании. В этом разделе будет обсуждаться результат всех классификаторов на основе данных, к которым они были применены.

Субдискредитированные данные (*Undersampling data)*

Результаты исследования, линейные классификаторы и наивный байесовский классификатор не являются лучшими вариантами моделирования кредитного скоринга. Таблица 3 демонстрирует, что у всех классификаторов были проблемы с недостаточным соответствием: алгоритмы плохо моделировали обучающие данные и некорректно выполнили задачу на данных тестирования.

Таблица 3

Accuracy of classifiers applied to undersampled data

	Linear Models		Non-Linear Models					
	Logistic Regression	SGD	Naïve Bayes	kNN	Decision Tree	Random Forest	Neural Networks	XGB
Training	58,6%	58,7%	59,0%	68,6%	68,3%	64,9%	70,8%	69,6%
Testing	58,7%	58,9%	59,0%	67,1%	65,9%	64,6%	70,0%	67,2%

Источник: составлено авторами

Несмотря на низкие показатели по другим метрикам, классификатор стохастического градиентного спуска (SGD) показал одну из самых низких долей ошибок второго рода. В этом время, классификатор нейронных сетей показал лучший результат среди моделей, примененных к данным с недостаточной выборкой на основе большинства метрик. Однако высокий процент ошибок 2-го рода, а также необходимость большого количества времени для обработки данных не делает его фаворитом. В итоге, модель классификатор экстремального бустинга градиента (XGB) является наиболее подходящим вариантом моделирования кредитного скоринга, с оптимальными показателями метрик: второй по величине показатель правильности классификации и второй по величине процент ошибок второго рода.

Таблица 4

Models and performance results

		Metrics						
		Accuracy	*Precision*	*Recall*	*F1*	*JSC*	*AUC_ROC*	*Type* 2 error
Linear	Logistic Regression	58,7%	59,0%	58,5%	58,7%	41,6%	58,7%	20,8%
	SGD	58,9%	57,9%	66,5%	61,9%	44,8%	58,9%	16,8%
Non-Linear	Naïve Bayes	59,0%	60,1%	54,3%	57,0%	39,9%	59,0%	23,0%
	kNN	67,1%	68,4%	64,0%	66,1%	49,4%	67,1%	18,6%
	Decision Tree	65,9%	66,9%	63,5%	65,2%	48,3%	65,9%	18,3%
	Random Forest	64,6%	63,5%	69,4%	66,3%	49,6%	64,6%	15,4%
	Neural Networks	70,0%	72,8%	64,1%	68,2%	51,7%	70,0%	18,0%
	XGB	67,2%	67,4%	67,3%	67,4%	50,8%	67,2%	16,4%

Источник: составлено авторами

Передискредитированные данные (Oversampled data)

Классификаторы в целом намного лучше работали с данными с избыточной выборкой, сформированными на основе метода повторной

выборки SMOTE, который привнес в данные дополнительную информацию. При этом, наихудшие результаты показали линейные классификаторы и наивный байесовский классификатор. Другие классификаторы показали положительные результаты при работе с набором данных для обучения, а также данными с избыточной выборкой.

Таблица 5

Accuracy of classifiers applied to oversampled data

	Linear Models		Non-Linear Models					
	Logistic Regression	SGD	Naïve Bayes	kNN	Decision Tree	Random Forest	Neural Networks	XGB
Training	64,1%	64,1%	64,9%	99,9%	76,8%	99,6%	73,6%	74,6%
Testing	64,1%	64,1%	64,9%	83,5%	75,3%	84,8%	73,6%	74,4%

Источник: составлено авторами

Нейронные сети были многообещающей моделью, но есть незначительная разница между показателем модели для данных с недостаточной и избыточной выборкой. Несмотря на повышение точности данных с избыточной выборкой по сравнению с данными с недостаточной выборкой, нейронные сети не смогли превзойти нелинейные модели.

Согласно таблице 6, классификаторы k-ближайших соседей и случайных лесов превзошли другие классификаторы по всем показателям. Кроме того, обе модели продемонстрировали самый низкий процент ошибок 2-го рода. Однако классификатор случайного леса превзошел классификатор k-ближайших соседей по всем показателям, кроме точности.

Таблица 6

Models and performance results

		Metrics						
		Accuracy	*Precision*	*Recall*	*F1*	*JSC*	*AUC_ROC*	*Type* 2 error
Linear	Logistic Regression	64,1%	63,9%	65,1%	64,5%	47,6%	64,1%	17,5%
	SGD	64,1%	63,9%	65,1%	64,5%	47,6%	64,1%	17,5%
Non-Linear	Naïve Bayes	64,9%	62,6%	74,2%	67,9%	51,4%	64,9%	12,9%
	kNN	83,5%	85,2%	81,0%	83,0%	71,0%	83,5%	9,5%
	Decision Tree	75,3%	74,8%	76,2%	75,5%	60,7%	75,3%	11,9%
	Random Forest	84,8%	84,7%	85,0%	84,8%	73,6%	84,8%	7,5%
	Neural Networks	73,6%	71,2%	79,1%	75,0%	60,0%	73,6%	10,5%
	XGB	74,4%	73,5%	76,2%	74,9%	59,8%	74,4%	11,9%

Источник: составлено авторами

5. Заключение

Результаты исследования показали, что модели машинного обучения достаточно хорошо работают на основе регуляторных данных, собранных центральным банком. Кроме того, анализ потребительских кредитов с помощью алгоритмов машинного обучения продемонстрировал уникальное понимание особенностей (плохих и хороших) потребительских заемщиков, а также предоставил информацию для проверки правильности выданных потребительских кредитов банками.

В этом исследовании мы показали, что очень важно проверять качество данных (во время процедур упорядочения и очистки, чтобы исключить ненужные переменные), и работать с несбалансированным набором обучающих данных, чтобы избежать смещения в пользу большинства категорий.

В части оценки точности прогнозов моделей, данные с избыточной выборкой, скорректированные методом SMOTE, показали более многообещающие результаты по сравнению с данными, обработанными стратегией случайного удаления мажоритарного класса. Другими словами, передискредитированные данные, скорректированные с помощью SMOTE, помогли минимизировать потерю информации и повысить эффективность прогнозирования. Модели с хорошо подобранными обучающими данными лучше работали с передискредитированными данными по сравнению с субдискредитированными данными.

Кроме того, нелинейные модели иллюстрировали более точные прогнозы по сравнению с линейными моделями. В частности, нелинейные модели, такие как классификаторы случайного леса и k-ближайших соседей на данных, превзошли другие модели классификаторов. С другой стороны, линейные модели, такие как логистическая регрессия и классификатор SGD, показали наихудшие результаты среди восьми сравниваемых моделей.

Подводя итог, можно сделать вывод, что модели, основанные на регуляторных данных, могут стать адекватной основой для оценки кредитного риска по выданным потребительским кредитам банками второго уровня, а также могут помочь центральному банку прогнозировать потенциальные систематические риски. Итак, согласно результатам исследования, оценка кредитного риска с помощью машинного обучения может стать хорошим дополнением к регулированию банков второго уровня, выдающих потребительские кредиты.

Для дальнейшего развития данного исследования предполагается несколько направлений. В первую очередь, необходимо дополнить имеющиеся наборы данных социально-демографическими характеристиками заёмщиков, которые могут положительно повлиять на производительность алгоритмов Grier (2012).

В части моделирования, предполагается несколько подходов, которые могут повысить эффективность моделей:

а. Применение гибридных методов машинного обучения

б. Построение системы выборочного комбинированного прогнозирования

с. Включение дополнительных параметров в сетку параметров и стратегии сэмплинга.

Список литературы

Addo, P., Guegan, D., & Hassani, B. (2018). Credit Risk Analysis Using Machine and Deep Learning Models. *Risks*, *6*(2), 38. doi: 10.3390/risks6020038

Alencar, R. (2017). Resampling strategies for imbalanced datasets. Retrieved from https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

Baesens, B., Van Gestel, T., Viaene, S., Stepanova, M., Suykens, J., & Vanthienen, J. (2003). Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of The Operational Research Society*, *54*(6), 627-635. doi: 10.1057/palgrave.jors.2601545

Brown, I., & Mues, C. (2012). An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, *39*(3), 3446-3453. doi: 10.1016/j.eswa.2011.09.033

Brownlee, J. (2016). A Gentle Introduction to XGBoost for Applied Machine Learning. Retrieved from https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

Brownlee, J. (2016). Metrics to Evaluate Machine Learning Algorithms in Python. Retrieved from https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/

Brownlee, J. (2020). SMOTE for Imbalanced Classification with Python. Retrieved from https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16, 321-357. Doi: 10.1613/jair.953

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of The 22Nd ACM SIGKDD International Conference On Knowledge Discovery and Data Mining*, 785–794. doi: 10.1145/2939672.2939785

du Boisberranger, J., Van den Bossche, J., Esteve, L., Fan, T., Gramfort, A., & Grisel, O. et al. User guide: contents – scikit-learn 0.23.2 documentation. Retrieved from https://scikit-learn.org/stable/user_guide.html

Fuchs, M. (2019). Introduction to SGD Classifier - Michael Fuchs Python. Retrieved from https://michael-fuchs-python.netlify.app/2019/11/11/introduction-to-sgd-classifier/

Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). Sebastopol, California: O'Reilly Media, Incorporated.

Grier, W. (2012). *Credit analysis of financial institutions* (2nd ed., pp. 294-296). London: Euromoney Institutional Investor PLC.

Hansen, C. (2019). Neural Networks: Feedforward and Backpropagation Explained. Retrieved from https://mlfromscratch.com/neural-networks-explained/#overview

Henley, W., & Hand, D. (1996). A k-Nearest-Neighbour Classifier for Assessing Consumer Credit Risk. *The Statistician*, *45*(1), 77. doi: 10.2307/2348414

Labatut, V., & Cherifi, H. (2011). Evaluation of Performance Measures for Classifiers Comparison. *Ubiquitous Computing and Communication Journal*, *6*, 21-34. Retrieved from https://arxiv.org/abs/1112.4133

Lemaitre, G., Nogueira, F., Oliveira, D., & Aridas, C. 2. Over-sampling – imbalanced-learn 0.5.0 documentation. Retrieved from https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html#smote-adasyn

Lessmann, S., Baesens, B., Seow, H., & Thomas, L. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research*, *247*(1), 124-136. doi: 10.1016/j.ejor.2015.05.030

Mueller, A., & Guido, S. (2017). *Introduction to Machine Learning with Python* (1st ed.). Sebastopol, California: O'Reilly Media.

Munkhdalai, L., Munkhdalai, T., Namsrai, O., Lee, J., & Ryu, K. (2019). An Empirical Comparison of Machine-Learning Methods on Bank Client Credit Assessments. *Sustainability*, *11*(3), 699. doi: 10.3390/su11030699

Schmarzo, B. (2018). Understanding Type 1 and Type 2 Errors [Blog]. Retrieved from https://www.datasciencecentral.com/profiles/blogs/understanding-type-i-and-type-ii-errors

Tsai, C., & Chen, M. (2010). Credit rating by hybrid machine learning techniques. *Applied Soft Computing*, *10*(2), 374-380. doi: 10.1016/j.asoc.2009.08.003

Приложение

```
# Важные библиотеки
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import os
path = 'E:\...'   # выбор пути к документу, в котором находится файл
os.chdir(path)
df_ml = pd.read_csv('filename.csv')
X_df = df_ml.iloc[:,:-1].values
y_df = df_ml.iloc[:,-1].values

# Случайное удаление мажоритарного класса с библиотеки imblearn
from imblearn.under_sampling import RandomUnderSampler as rus
us = rus(random_state=42)
X, y = us.fit_resample(X_df, y_df)

# Стратегия SMOTE-NC с библиотеки imblearn
from imblearn.over_sampling import SMOTENC
sm = SMOTENC(random_state=42, categorical_features=[0,1,2,3,4,5])  # в исходных данных имеются категориальные переменные
X, y = sm.fit(X_df, y_df)

# Кодирование категориальных переменных
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# первый столбец категориальных переменных содержал 18 различных переменных (три города и региона (также бывшее название для одного региона)), таким образом создавалось 18 строк
ct0 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[0])], remainder='passthrough')
X=np.array(ct0.fit_transform(X))

# второй столбец категориальных переменных содержал 4 разных переменных (четыре типа валюты, в которой был выдан заем), таким образом создавая 4 строки
ct1 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
X=np.array(ct1.fit_transform(X))
```

```
# третий столбец категориальных переменных содержал 2 разные переменные (кредитная карта или заем), таким образом создавая 2 строки
ct2 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
X=np.array(ct2.fit_transform(X))

# четвертый столбец категориальных переменных содержал 2 разные переменные (потребительское или автокредитование), таким образом создавая 2 строки
ct3 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
X=np.array(ct3.fit_transform(X))

# пятый столбец категориальных переменных содержал 2 разные переменные (мужской или женский), таким образом создавая 2 строки
ct4 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
X=np.array(ct4.fit_transform(X))

# шестой столбец категориальных переменных содержал 2 разные переменные (резидент или нерезидент), таким образом создавая 2 строки
ct5 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
X=np.array(ct5.fit_transform(X))

# Label Encoder переменной y
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

# разделение данных на обучающий и тестовый набор данных
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=1)

# шкалирование
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:,-3:] = sc.fit_transform(X_train[:,-3:])
X_test[:,-3:] = sc.transform(X_test[:,-3:])

# Метод главных компонентов (Principal component analysis (PCA))
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=a)    # a is quantity where cumulative explained variance ratio > 95%
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

# Логистическая регрессия
from sklearn.linear_model import LogisticRegression
log = LogisticRegression().fit(X_train,y_train)
y_tr_log_pred = log.predict(X_train)
y_ts_log_pred = log.predict(X_test)

# Стохастческая градиентный спуск
from sklearn.linear_model import SGDClassifier
sgd=SGDClassifier().fit(X_train,y_train)
y_tr_sgd_pred = sgd.predict(X_train)
y_ts_sgd_pred = sgd.predict(X_test)

# Гауссовский наивный Байесовский классификатор
from sklearn.naive_bayes import GaussianNB
nbc = GaussianNB().fit(X_train,y_train)
y_tr_nb_pred = nbc.predict(X_train)
y_ts_nb_pred = nbc.predict(X_test)

# k-ближайшие соседи
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier().fit(X_train,y_train)
y_tr_knn_pred = knn.predict(X_train)
y_ts_knn_pred = knn.predict(X_test)

# Дерево решений
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier(max_depth=14,
criterion='entropy').fit(X_train,y_train)
y_tr_dt_pred = dtc.predict(X_train)
y_ts_dt_pred = dtc.predict(X_test)

# Случайный лес
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier().fit(X_train,y_train)
y_tr_rf_pred = rfc.predict(X_train)
y_ts_rf_pred = rfc.predict(X_test)

# Многослойный персептрон
from sklearn.neural_network import MLPClassifier
```

```
nnc = MLPClassifier().fit(X_train,y_train)
y_tr_nnc_pred = nnc.predict(X_train)
y_ts_nnc_pred = nnc.predict(X_test)

# XGBoost
from xgboost import XGBClassifier
xgb = XGBClassifier().fit(X_train,y_train)
y_tr_xgb_pred = xgb.predict(X_train)
y_ts_xgb_pred = xgb.predict(X_test)

# Кросс-валидация
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = xgb, X = X_train, y = y_train, cv = 5)
print('Accuracy: {:.2f} %'.format(accuracies.mean()*100))
print('Standard deviation: {:.2f} %'.format(accuracies.std()*100))

### Примечание: мы можем поставить другие модели вместо XGB для анализа результата перекрестной проверки.

# Перебор по сетке для логистической регрессии
from sklearn.model_selection import GridSearchCV
parameters = [{'penalty': ['none'], 'solver':['newton-cg', 'sag', 'saga', 'lbfgs']},
              {'penalty': ['elasticnet'], 'C': [0.01, 0.1, 0.25, 0.5, 0.75, 1, 5, 10], 'solver':['saga']},
              {'penalty': ['l2'], 'C': [0.01, 0.1, 0.25, 0.5, 0.75, 1, 5, 10], 'solver':['newton-cg', 'sag', 'saga', 'lbfgs']}]
grid_search = GridSearchCV(estimator = log,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Перебор по сетке для стохастического градиента спуска
from sklearn.model_selection import GridSearchCV
parameters = [{"loss" : ["hinge", "log", "squared_hinge", "modified_huber"],
               "alpha" : [0.0001, 0.001, 0.01, 0.1], "penalty" : ["l2", "l1", "none"]}]
grid_search = GridSearchCV(estimator = sgd,
                           param_grid = parameters,
                           scoring = 'accuracy',
```

```
                    cv = 5,
                    n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Перебор по сетке для Гауссовскому наивному Байесовскому классификатору
from sklearn.model_selection import GridSearchCV
parameters = [{'var_smoothing': [1e-12,1e-10,1e-7,1e-4,1e-3,1e-2,1e-1,1,10]}]
grid_search = GridSearchCV(estimator = nbc,
                    param_grid = parameters,
                    scoring = 'accuracy',
                    cv = 5,
                    n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Перебор по сетке для k-ближайших соседей
from sklearn.model_selection import GridSearchCV
parameters = [{'n_neighbors': list(range(1, 81))}]
grid_search = GridSearchCV(estimator = knn,
                    param_grid = parameters,
                    scoring = 'accuracy',
                    cv = 5,
                    n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Перебор по сетке для дерева решений
from sklearn.model_selection import GridSearchCV
parameters = [{'criterion':['gini', 'entropy'],'max_depth': list(range(1,21))}]
grid_search = GridSearchCV(estimator = dtc,
                    param_grid = parameters,
                    scoring = 'accuracy',
                    cv = 5,
                    n_jobs = -1)
```

```
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Перебор по сетке для случайного леса
from sklearn.model_selection import GridSearchCV
parameters = [{'n_estimators': list(range(1,21)), 'max_features': ['auto', 'sqrt', 'log2'],
            'max_depth' : ['None',8], 'criterion' :['gini', 'entropy']}]
grid_search = GridSearchCV(estimator = rfc,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Перебор по сетке для многослойного персептрона
from sklearn.model_selection import GridSearchCV
parameters = [{'hidden_layer_sizes':[100,200,300,[200,50],[100,100],[200,100]],
            'activation':['identity','logistic','tanh','relu'],
            'solver': ['adam'],
            'learning_rate':['constant','invscaling','adaptive'],
            'max_iter': [1000,1500,2000 ]}]
grid_search = GridSearchCV(estimator = nnc,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Перебор по сетке для XGBoost
from sklearn.model_selection import GridSearchCV
parameters = [{'n_estimators': [1000], #number of trees, change it to 1000 for better
results
            'max_depth': [6,7,8],
```

```
            'learning_rate': [0.05], #so called `eta` value
            'objective':['binary:logistic'],
            'tree_method':['exact'],
            'min_child_weight': [11],
            'subsample': [0.8],
            'colsample_bytree': [0.7]}]
grid_search = GridSearchCV(estimator = xgb,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)
```

После перебора по сетке должны быть применены все лучшие параметры, чтобы быть уверенным, что модель с наилучшей точностью дает более высокую точность, и после этого следует проанализировать другие метрики.

```
# Метрики
from sklearn.metrics import confusion_matrix, accuracy_score
knn_cm_tr = confusion_matrix(y_train,y_tr_knn_pred)
print(knn_cm_tr)
accuracy_score(y_train, y_tr_knn_pred)

knn_cm_ts = confusion_matrix(y_test,y_ts_knn_pred)
print(knn_cm_ts)
accuracy_score(y_test, y_ts_knn_pred)

from sklearn.metrics import roc_auc_score, jaccard_score, f1_score,
precision_score, recall_score
print(roc_auc_score(y_test,y_ts_knn_pred))
print(jaccard_score(y_test,y_ts_knn_pred))
print(f1_score(y_test,y_ts_knn_pred))
print(precision_score(y_test,y_ts_knn_pred))
print(recall_score(y_test,y_ts_knn_pred))
```

Здесь метрики использовались для анализа производительности классификатора k-ближайших соседей, переменные должны быть изменены для достижения результата других моделей.