Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»

Департамент анализа данных и машинного обучения

Пояснительная записка к курсовой работе

по дисциплине “Технологии анализа данных и машинное обучение”

на тему:

«Машинное обучение в задачах кредитного скоринга»

Выполнила:

студентка группы ПИ19-2 факультета
информационных технологий и анализа
больших данных

Волкова Татьяна Алексеевна

_____ _____
(Подпись)

Научный руководитель:

старший преподаватель

Добрина Мария Валерьевна

_____
(Подпись)

Москва, 2022

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Финансовые учреждения все больше полагаются на машинное обучение (ML) для поддержки принятия решений. В данной курсовой работе рассматривается применение ML на рынке розничного кредитования, который является крупным и экономически важным сегментом кредитной индустрии.

Актуальность темы обусловлена необходимостью прогнозирования надежности клиента с точки зрения его платежеспособности, ведь, например, только в 2020 году общая сумма задолженности по розничным кредитам в США превысила 4161 млрд долларов. Скоринговые модели на основе ML, также называемые скоринговыми картами, играют важную роль в утверждении соответствующих кредитов.

В задачах предсказания решений о выдаче кредита с помощью машинного обучения есть одна главная проблема – распределение классов (платежеспособный и неплатежеспособный клиент) не равномерное. Они могут делиться соответственно в отношении 70 на 30, а чаще всего (в открытых датасетах) распределение доходит до 90 на 10. Это очень сильно затрудняет процесс обучения моделей и сильно ухудшает предсказательные способности модели.

В рамках данной курсовой будет произведена обзорная работа по текущим решениям в сфере кредитного скоринга, сравнение этих решений и выбор наилучшего.

Практическая часть работы будет выполняться на языке программирования Python версии 3.10 в среде разработки Python (Jupyter Notebook) и Google Colaboratory.

1. ПОСТАНОВКА ЗАДАЧИ

В курсовом проекте в соответствии с выбранной темой требуется:

1. Выбрать набор данных для анализа, провести его очистку, предварительную обработку и описательный анализ.
2. Разделить набор данных на обучающую и тестовую выборки.
3. Обучить несколько моделей для решения выбранной задачи - не менее 7 различных алгоритмов.
4. Оценить качество моделирования с использованием метрик точности, достоверности, собственных и т. д.
5. Сравнить модели и выбрать наиболее перспективную для решения поставленной задачи.
6. Провести Grid Search и найти оптимальные гиперпараметры.
7. Сделать выводы.

Работа должна содержать

- замеры времени с использованием стандартной библиотеки Python;
- текстовые замечания, поясняющие каждый шаг работы;
- визуализацию моделирования в наглядном виде на этапах описательного анализа и анализа обучаемости модели (графики, таблицы сравнения моделей, таблицы классификации, и другие).

Результат должен быть представлен в виде программного ноутбука Python Jupyter.

Помимо практических навыков, важно умение оформлять документацию, в том числе пояснительную записку, отвечающую таким требованиям, как полнота, техническая корректность, лаконичность, и т.д. Техническая документация - неотъемлемая часть программного продукта, она является посредником между разработчиком и конечным пользователем. Поэтому грамотное оформление документации так же важно для разработки программы, как, собственно, и разработка.

2. МЕТОДОЛОГИЯ

В данной работе произведена работа в несколько этапов (см. Рисунок 1):

1. Очистка набора данных из открытого источника;
2. Разделение набора на обучающую и тестовую выборку;
3. Обучение каждой комбинации метода перебалансировки классов и классификационной модели;
4. Выбор наилучшей комбинации с помощью метрик классификации;
5. Нахождение гиперпараметров для наилучшей комбинации.

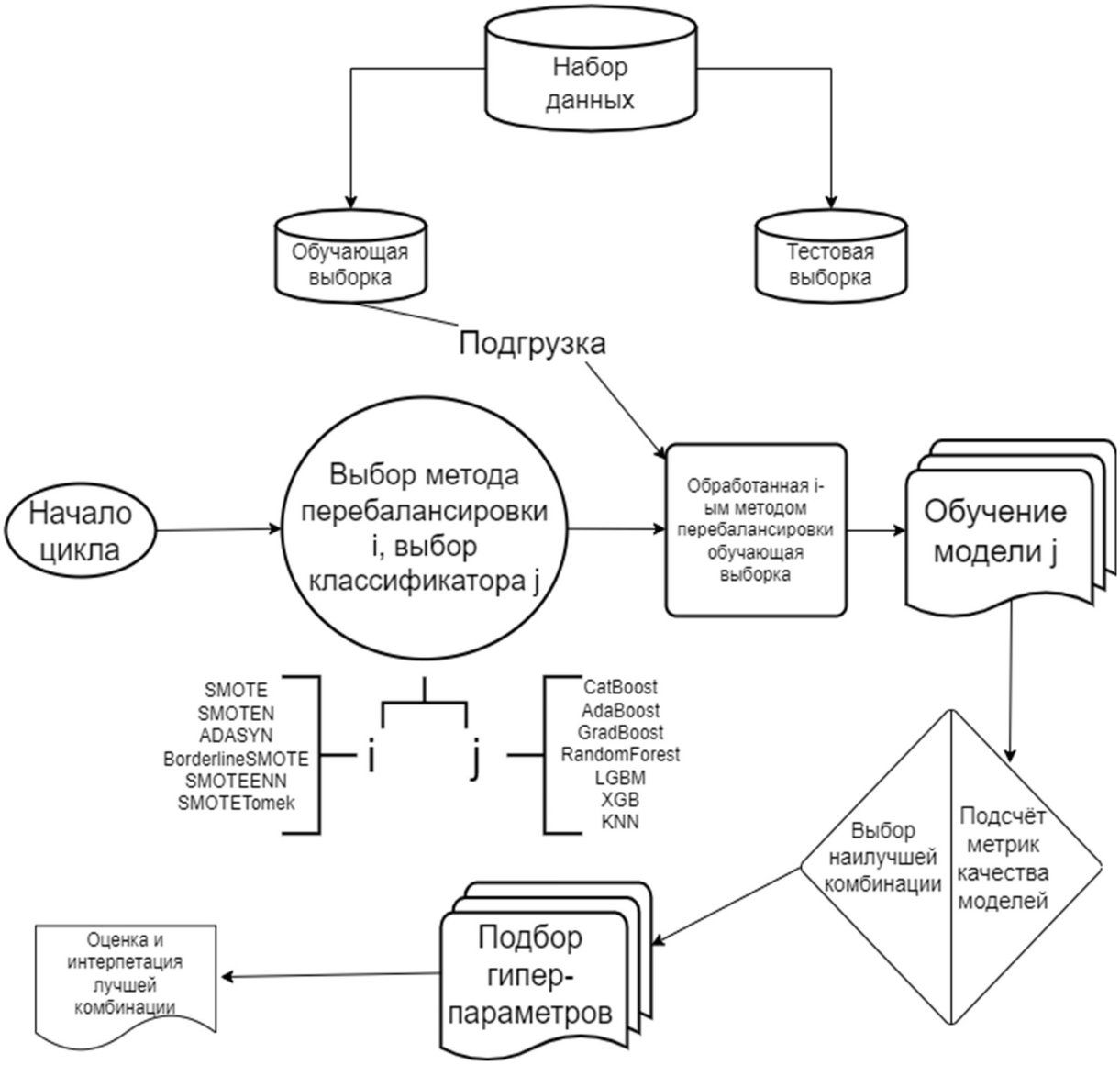

Рисунок 1. Общая методология проведения работы

В конце хода исследования планируется получить наилучшую модель с точки зрения метрик качества. Будет введена собственная метрика оценки модели в разрезе бизнеса, чтобы проще интерпретировать результат и наиболее достоверно выбрать модель.

3. ТЕОРЕТИЧЕСКАЯ СПРАВКА

3. 1 Модели машинного обучения

В этом разделе описаны выбранные алгоритмы классификации для дальнейших экспериментов. В основном все эти модели – виды бустингов, потому что цель данной работы – дать актуальный обзор на задачу кредитного скоринга, а алгоритмы бустинга считаются передовыми state-of-the-art решениями в классическом машинном обучении.

CatBoost[1]

CatBoost (от «Categorical Boosting») — открытая программная библиотека градиентного бустинга, разработанная компанией Яндекс. Это мощный метод машинного обучения, который позволяет достичь самых высоких результатов в различных практических задачах. На протяжении многих лет он остается основным методом для обучения задач с разнородными характеристиками, зашумленными данными и сложными зависимостями: веб-поиск, рекомендательные системы, прогнозирование погоды и многие другие.

Градиентный бустинг — это, по сути, процесс построения ансамблевого предиктора путем выполнения градиентного спуска в функциональном пространстве. Он опирается на надежные теоретические результаты, которые объясняют, как сильные предикторы могут быть построены путем итеративного объединения слабых моделей (базовых предикторов).

Все существующие реализации градиентного бустинга сталкиваются со следующей статистической проблемой. Модель предсказания F, полученная после нескольких шагов бустинга, полагается на цели всех обучающих примеров. Это приводит к смещению распределения F(xk) | xk для обучающего примера xk от распределения F(x) | x для тестового примера x. Это в итоге приводит к смещению предсказания выученной модели. Аналогичная проблема существует в стандартных алгоритмах предварительной обработки категориальных

признаков. Одним из наиболее эффективных способов их использования в градиентном бустинге является преобразование категорий в их целевые статистики. Целевая статистика сама по себе является простой статистической моделью, и она также может вызвать утечку цели и смещение предсказания.

Данный алгоритм предлагает принцип упорядочивания для решения обеих проблем. Опираясь на него, он вводит упорядоченный бустинг, модификацию стандартного алгоритма градиентного бустинга, которая позволяет избежать утечки цели. Их комбинация и реализована в виде библиотеки CatBoost, которая превосходит существующие современные реализации градиентного бустинга.

AdaBoost[2]

AdaBoost (сокращение от «Adaptive Boosting») — это алгоритм статистической классификации. Для увеличения производительность может применяться в сочетании со другими алгоритмами обучения. Результаты их работы объединяются во взвешенную сумму, которая и представляет еонечный результат классификатора с усилением. AdaBoost является адаптивным: последующие слабые обучающие алгоритмы настраиваются в пользу экземпляров, которые были неправильно классифицированы предыдущими классификаторами.

В некоторых задачах AdaBoost менее восприимчив к переобучению, чем другие методы классификации. Несмотря на то, что AdaBoost обычно используется для объединения слабых базовых обучаемых (таких как деревья решений), он также может эффективно объединять сильные базовые обучаемые (такие как глубокие деревья решений), создавая еще более точную модель.

AdaBoost с деревьями решений в качестве слабых обучаемых часто называют лучшим классификатором. Собранная на каждом последующем этапе информация об относительной «трудности» обучающей выборки, поступает в алгоритм выращивания деревьев, поэтому более поздние деревья фокусируются на более трудных для классификации примерах.

GradBoost[3]

GradBoost — это метод машинного обучения, используемый, в частности, в задачах регрессии и классификации. Он дает модель предсказания в виде ансамбля слабых моделей предсказания, которые обычно являются деревьями решений. Когда дерево решений является слабо-обучаемым, результирующий алгоритм называется gradient-boosted trees. Он обычно превосходит random forest. Модель gradient-boosted trees строится поэтапно, как и в других методах бустинга.

RandomForest[4]

Случайные леса решений — это алгоритм ансамблевого обучения для решения многих задач, например регрессии, классификации и некоторых других. Алгоритм его работы заключается в том, что он строит множество деревьев решений во время обучения

Для задач классификации результатом случайного леса является класс, выбранный большинством деревьев; для задач регрессии - средний или усредненный прогноз отдельных деревьев.

Случайные леса решений исправляют привычку деревьев решений чрезмерно подстраиваться под обучающий набор. Случайные леса в целом превосходят деревья решений, однако точность их ниже, чем у деревьев градиентным усилением. Впрочем, на их эффективность могут влиять характеристики данных.

Случайные леса часто используются в бизнесе, ведь не требуют особой настройки и при этом они генерируют разумные прогнозы по широкому спектру данных.

LGBM[5]

LightGBM (от «Light Gradient Boosted Machine») — это еще одна реализация градиентного бустинга. В этом алгоритме содержатся две новые

техники: Exclusive Feature Bundling и Gradient-based One-Side Sampling для работы с большим количеством экземпляров данных и большим количеством признаков соответственно. С помощью LightGBM можно значительно превзойти XGBoost и SGB по скорости вычислений и потреблению памяти.

XGB[6]

XGBoost (от «eXtreme Gradient Boosting») — одна из самых популярных и эффективных реализаций алгоритма градиентного бустинга. В этом алгоритме имплементирована масштабируемая сквозная система бустинга деревьев, которая широко используется специалистами по обработке данных для достижения современных результатов при решении многих задач машинного обучения. Это особенный алгоритм с точки зрения разреженности данных, имеющий взвешенный квантильный эскиз для приближенного обучения деревьев. Более того, он предоставляет информацию о моделях доступа к кэшу, сжатии и разделении данных для создания масштабируемой системы древовидного бустинга. Объединив эти идеи, XGBoost масштабируется до миллиардов примеров, используя гораздо меньше ресурсов, чем существующие системы.

KNN[7]

KNN (от «k Nearest Neighbor») — один из самых примитивных алгоритмов классификации, также иногда применяемый в задачах регрессии. Для прогнозирования новых значений KNN использует сходство признаков, т. е. новой точке данных будет определено значение на основании того, насколько близко он находится к точкам в обучающей выборке. Близость чаще всего измеряют евклидовом расстоянием.

3.2 Методы предобработки данных для решения разбалансировки классов

В этом разделе описаны подходы к построению классификаторов на основе несбалансированных наборов данных. Набор данных является несбалансированным, если классификационные категории представлены в нем неравномерно. Зачастую реальные наборы данных состоят преимущественно из нормальных примеров и лишь малого количества интересных или аномальных примеров. Помимо этого, стоимость ложной классификации аномального или интересного примера как нормального зачастую гораздо выше, чем стоимость обратной ошибки.

Так же присутствует следующая проблема – распределение классов платежеспособный и неплатежеспособный клиент не равномерно. Они могут делиться соответственно в отношении 70 на 30, а чаще всего (в открытых датасетах) распределение доходит до 90 на 10. Это очень сильно затрудняет процесс обучения моделей и сильно ухудшает предсказательные способности модели.

SMOTE[1]

SMOTE – в переводе, способ передискретизации синтезированных меньшинств. Алгоритм находит класс с меньшим количеством элементов и в близи от уже существующих элементов из этого класса создаёт новые (Рисунок 2).

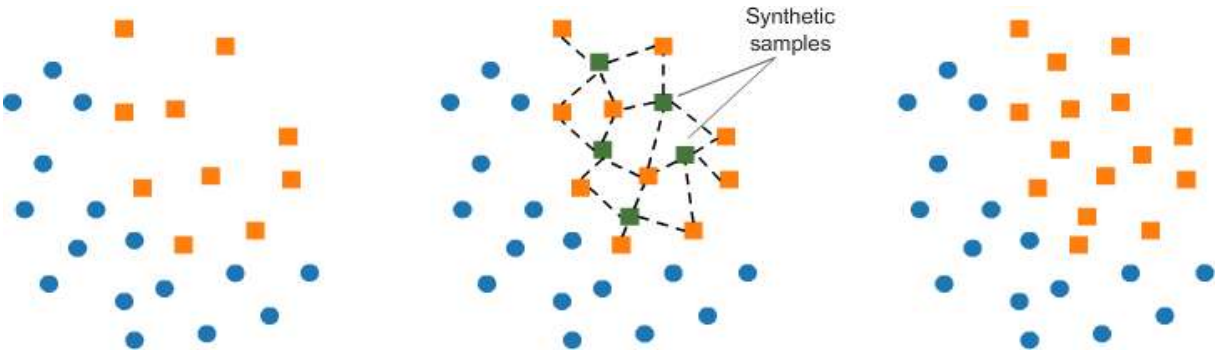

Рисунок 2 Визуализация алгоритма SMOTE

Дальнейшие использованные алгоритмы перебалансировки представляют из себя альтернативные версии алгоритма SMOTE, поэтому для лучшего понимания, представлен алгоритм на псевдо-языке:

Вход:

T – количество наблюдений в классе − меньшинстве

N − сколько % делать с помощью SMOTE

k – количество ближайших соседей

Выход:

N*T - дополнительных искусственных наблюдений в класс минор

```
numattrs = количество признаков
Sample[][]: массив для изначальных наблюдений из минорного класса
newindex: для отслеживания счётчика новых элементов, изначально 0
Synthetic[][]: массив для новых искусственных элементов

для i=1 до T
  Заполнить(N, i, nnarray)
конецдля

функция Заполнить(N, i, nnarray): вычислить k ближайших соседей и заполнить
ими
пока N != 0
   nn = случмежду(1,k)
   для attr = 1 до numattrs
      dif = Sample[nnarray[nn]][attr] - Sample[i][attr]
      gap = случмежду(0,1)
      Synthetic[newindex][attr] = Sample[i][attr] + gap * dif
   конецдля
   newindex += 1
   N = N-1
конецпока
конецфункции
```

SMOTEN[2]

Алгоритм аналогичен SMOTE, однако отличается способом расчёта расстояния между векторами признаков наблюдений для отделения k ближайших соседей.

ADASYN[3]

ADASYN – в переводе адаптивный подход синтетического отбора проб. Также пытается перераспределить класс меньшинств.

Алгоритм представлен на рисунке 3. и заключается в следующих шагах:

1. Подсчитать $d = ml/ms$ – отношение размеров меньшего класса и большего класса.
2. Определить B – гиперпараметр, для вычисления количества новых искусственных наблюдений для генераций $G = (ml - ms) * B$.
3. Для каждого наблюдения из меньшего класса подсчитать плотность присутствия наблюдений из большего класса $R_i = ml/k$.
4. Нормализовать все R_i, чтобы в сумме они давали 1.
5. Посчитать $G_i = G * R_i$ – количество наблюдений для генерации вокруг i-ого элемента.
6. Создать G_i наблюдений для каждого элемента по формуле $S_i = X_i + (X_{zi} - X_i) *$ случмежду(0 ,1).

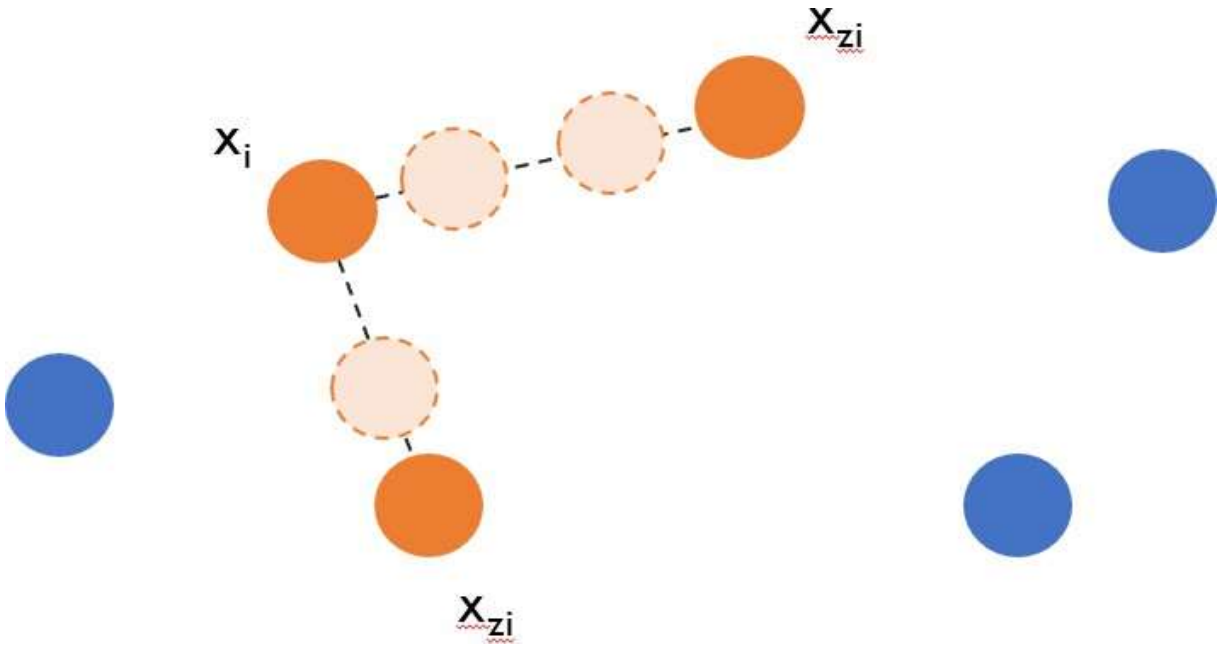

Рисунок 3 Визуализация алгоритма ADASYN

BorderlineSMOTE[4]

Работает так же, как и обычный SMOTE, но при каждом прохождении по наблюдению из минорного класса пропускает его, если у этого элемента все соседи из мажорного класса

SVMSMOTE[5]

SVM SMOTE (от «Support Vectors SMOTE») — также является вариантом алгоритма SMOTE. Однако для выявления элементов, в близи с которыми будут создаваться новые элементы. использует алгоритм SVM – или по-другому метод опорных векторов

SMOTEENN[6]

Опять же аналог SMOTE, но при проходе по всем наблюдениям удаляет i-ое наблюдение и его k-соседей, если класс наблюдения не совпадает с превосходствующим классом среди соседей.

SMOTETomek[7]

SMOTE + Tomek links. Этот алгоритм рассчитывает сколько элементов из мажорного класса необходимо удалить для достижения требуемого уровня соотношения различных классов. Удаляет он элементы, входящие в связи Томека. Этот способ хорошо очищает зашумленные данные.

В ходе эксперимента будет проведено сравнение вышеописанных алгоритмов классификации, а также алгоритмов перебалансировки данных и будет выделен лучшие из них, а также лучшая их комбинация.

4. ЭКСПЕРИМЕНТ

4.1 Описание набора данных

Для исследования упомянутых архитектур был выбран набор данных с соревнования Give Me Some Credit :: 2011 Competition Data | Kaggle.

Данный набор представляет из себя таблицу с информацией о клиентах заёмщиках со следующими признаками:

- RevolvingUtilizationOfUnsecuredLines - Общий баланс по кредитным картам и личным кредитным линиям, за исключением недвижимости и отсутствия задолженности по рассрочке платежа;
- Age – Возраст;
- DebtRation - Ежемесячные выплаты по долгам, алименты, расходы на проживание;
- MonthlyIncome – Ежемесячный доход;
- NumberOfDependents - Количество иждивенцев в семье, не считая себя (супруг, дети и тд).
- NumberOfOpenCreditLinesAndLoans - Количество открытых кредитов (например, рассрочка, автокредит или ипотека) и кредитных линий (например, кредитные карты);
- NumberRealEstateLoansOrLines - Количество ипотечных кредитов и кредитов на недвижимость, включая кредитные линии на покупку жилья;
- NumberOfTime30-59DaysPastDueNotWorse - Количество случаев, когда заемщик допускал просрочку в 30-59 дней (но не хуже) за последние 2 года;
- NumberOfTime60-89DaysPastDueNotWorse - Количество случаев, когда заемщик допускал просрочку на 60-89 дней (но не хуже) за последние 2 года;
- NumberOfTimes90DaysLate - Количество случаев, когда заемщик просрочил платеж на 90 дней и более;

Итого 11 признаков по которым определяется платёжеспособен ли клиент.

Столбец SeriousDlqin2yrs содержит значения 1 или 0 – выдан или нет кредит соответственно. Эти значения (классы) нам и необходимо предсказать.

Размер всего набора – 150000 тысяч наблюдений.

Загрузим датасет в Jupiter Notebook (Рисунок 4.)

```
data = pd.read_csv('cs-training.csv', index_col = 'Unnamed: 0')
data.head()
```

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOf
1	1	0.766127	45	2	0.802982	9120.0	
2	0	0.957151	40	0	0.121876	2600.0	
3	0	0.658180	38	1	0.085113	3042.0	
4	0	0.233810	30	0	0.036050	3300.0	
5	0	0.907239	49	1	0.024926	63588.0	

Рисунок 4. Загрузка датасета

Типы данных представлены в Таблице 2. Как мы видим все признаки относятся к числовым, следовательно преобразовывать нам их необходимости нет.

SeriousDlqin2yrs	int64
RevolvingUtilizationOfUnsecuredLines	float64
age	int64
NumberOfTime30-59DaysPastDueNotWorse	int64
DebtRatio	float64
MonthlyIncome	float64
NumberOfOpenCreditLinesAndLoans	int64
NumberOfTimes90DaysLate	int64
NumberRealEstateLoansOrLines	int64
NumberOfTime60-89DaysPastDueNotWorse	int64
NumberOfDependents	float64

Таблица 2. Типы данных

Статистика пропусков представлена в Таблице 3. Пропуски наблюдаются в столбцах ежемесячный доход и количество иждивенцев. На мой взгляд, восстановить их нет возможности (среднее или медианное значение могут быть сильно ошибочным), а также в их восстановлении нет смысла – объем данных и так большой, было принято решение полностью избавиться от пропусков – полностью удалить строки.

SeriousDlqin2yrs	0
RevolvingUtilizationOfUnsecuredLines	0
age	0
NumberOfTime30-59DaysPastDueNotWorse	0
DebtRatio	0
MonthlyIncome	29731
NumberOfOpenCreditLinesAndLoans	0
NumberOfTimes90DaysLate	0

```
NumberRealEstateLoansOrLines                  0
NumberOfTime60-89DaysPastDueNotWorse          0
NumberOfDependents                         3924
```

Таблица 3. Статистика пропусков

Также произведено вычисление описательной статистики и распределений признаков. Статистика представлена в Таблице 4. А распределение на рисунке 5. Судя по распределению выбросов, которые сильно бы помешали дальнейшему анализу, не наблюдается.

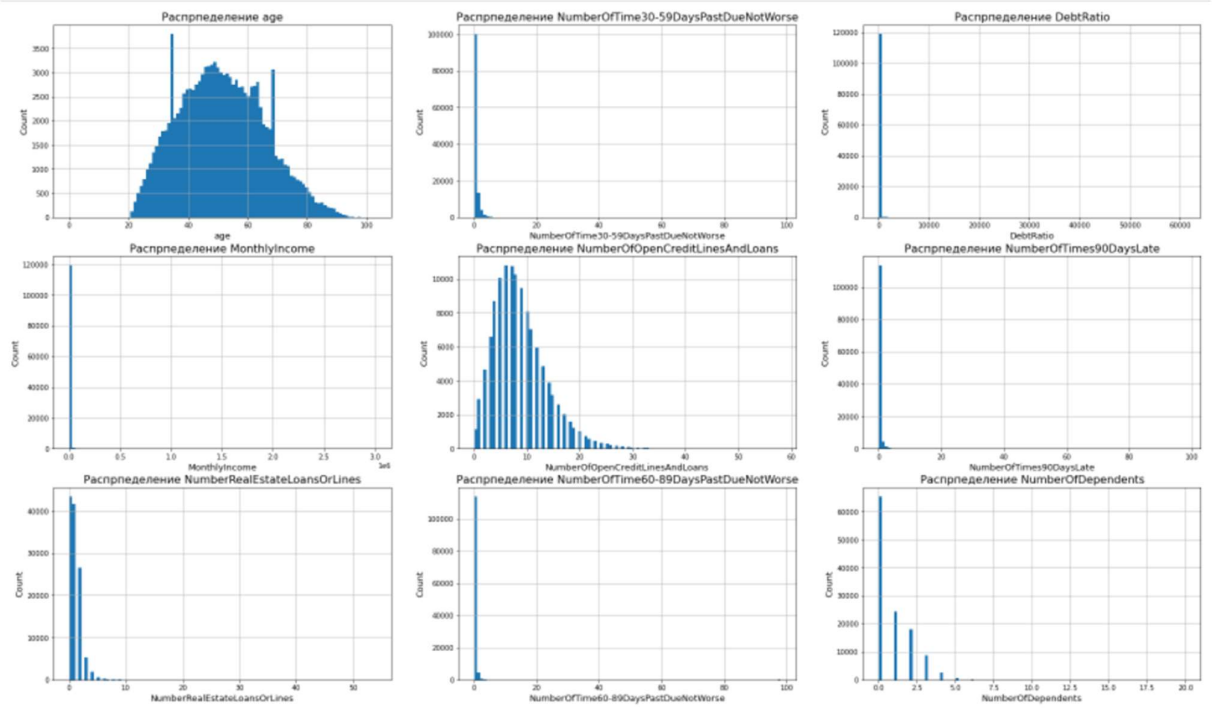

Рисунок 5. Распределение признаков

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome
count	120269.000000	120269.000000	120269.000000	120269.000000	120269.000000	1.202690e+05
mean	0.069486	5.899873	51.289792	0.381769	26.598777	6.670221e+03
std	0.254280	257.040685	14.426684	3.499234	424.446457	1.438467e+04
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00
25%	0.000000	0.035084	40.000000	0.000000	0.143388	3.400000e+03
50%	0.000000	0.177282	51.000000	0.000000	0.296023	5.400000e+03
75%	0.000000	0.579428	61.000000	0.000000	0.482559	8.249000e+03
max	1.000000	50708.000000	103.000000	98.000000	61106.500000	3.008750e+06

NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	NumberRealEstateLoansOrLines	NumberOfTime60-89DaysPastDueNotWorse	NumberOfDependents
120269.000000	120269.000000	120269.000000	120269.000000	120269.000000
8.758475	0.211925	1.054519	0.187829	0.851832
5.172835	3.465276	1.149273	3.447901	1.148391
0.000000	0.000000	0.000000	0.000000	0.000000
5.000000	0.000000	0.000000	0.000000	0.000000
8.000000	0.000000	1.000000	0.000000	0.000000
11.000000	0.000000	2.000000	0.000000	2.000000
58.000000	98.000000	54.000000	98.000000	20.000000

Таблица 4. Описательная статистика

Посмотрим коррелируют ли признаки между собой и узнаем нужно ли нам удалить некоторые из них. А также изучим какие признаки лучше всего влияют на принятие решения о выдаче кредита. Результат представлен на Рисунке 6. Так как значения корреляции в целом невысокие, поэтому оставим все признаки.

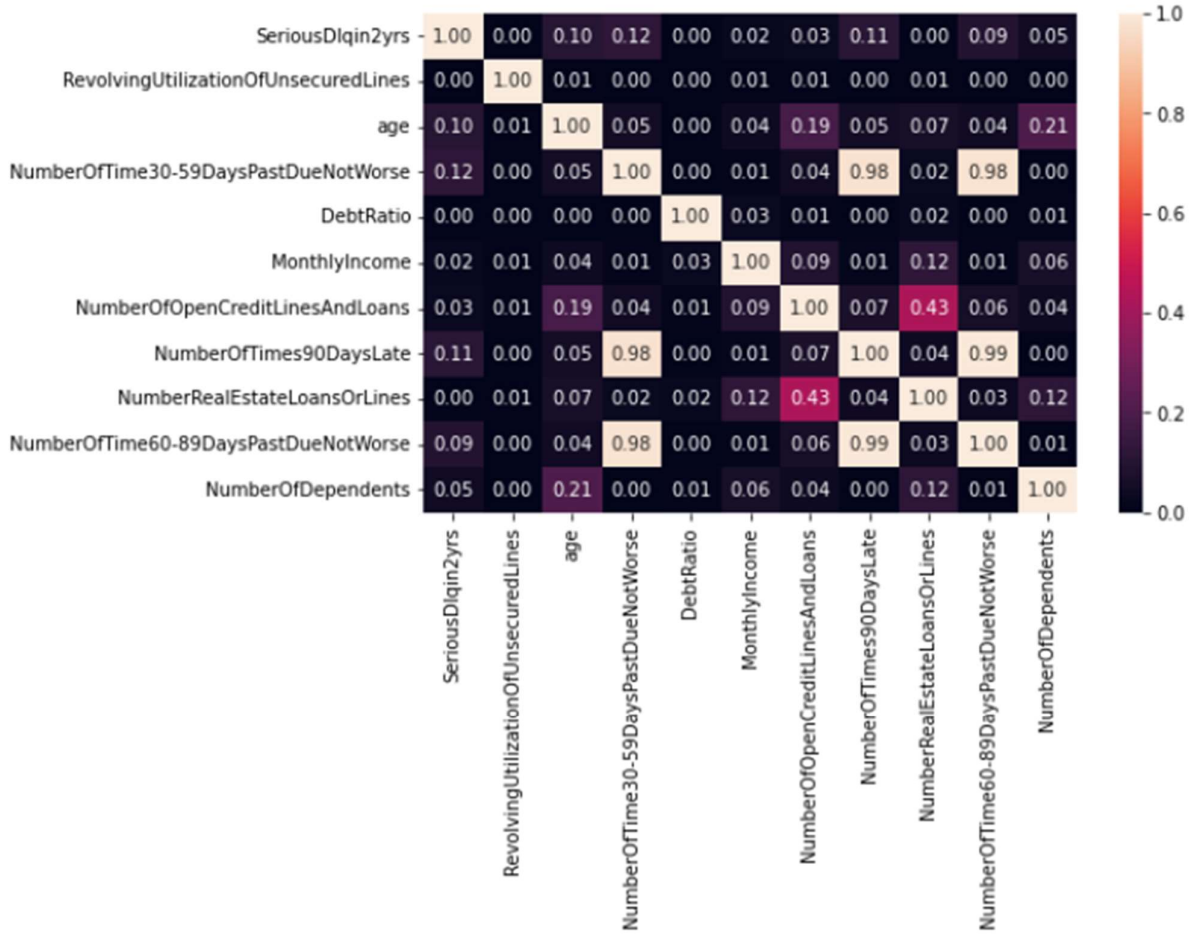

Рисунок 6. Корреляция признаков

4.2 Построение и обучение моделей

Ранее я уже описала какие классификаторы и способы перебалансировки классов будут использоваться. В этом разделе только перечислю их.

Классификаторы:

- Catboost
- AdaBoost
- GradBoost
- RandomForest
- LGBM
- XGB
- KNN

Способы перебалансировки:

- SMOTE
- SMOTEN
- ADASYN
- BorderlineSMOTE
- SVMSMOTE
- SMOTEENN
- SMOTETomek

Пройдемся в двух циклах по всем классификаторам и методам перебалансировке (в итоге будут составлены все возможные пары комбинаций) и будем создавать, обучать и тестировать модели с целью найти модель с лучшими метриками.

Кратко опишу процесс обучения и тестирования моделей:

1. Делим наш датасет на обучающую и тестовые выборки.
2. Применяем метод балансировки к данным
3. Обучаем модель на тренировочной (обучающей) выборке с помощью метода fit
4. С помощью метода predict тестируем (предсказываем значения) модель на тестовой выборке

5. Метрики accuracy, precision, recall, f1, матрица ошибок и др. позволяют нам определить, насколько хорошо созданная модель предсказывает класс. Обо всех метриках я расскажу подробнее далее.

   Все вышеописанные действия происходят в коде на Рисунке 6.

```
X = data.drop('SeriousDlqin2yrs', axis=1)
y = data['SeriousDlqin2yrs']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify = y)

for i in tqdm(oversamplers.keys()):
    for j in tqdm(classifiers.keys()):

        pipeline = Pipeline(
            [
             ('oversamplingmethod', oversamplers[i]),
             ('model', classifiers[j])
            ],
            verbose = False
        )

        start = time.time()
        pipeline.fit(X_train, y_train)
        end = time.time()

        y_pred = pipeline.predict(X_test)

        As.loc[i, j] = accuracy_score(y_test, y_pred)
        Ps.loc[i, j] = precision_score(y_test, y_pred)
        Rs.loc[i, j] = recall_score(y_test, y_pred)
        F1s.loc[i, j] = f1_score(y_test, y_pred)
        times.loc[i, j] = round(end - start, 2)

        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        results.loc[i, j] = fp * 1 + fn * 5

        matrices.append(confusion_matrix(y_test, y_pred))
```

Рисунок 6. Обучение и тестирование модели.

Весь пайплайн обучения занял 30 минут

4.3 Результаты обучения

Матрицы ошибок

Эффективность алгоритмов машинного обучения обычно оценивается с помощью матрицы ошибок, как показано в Таблице 5 (для задачи с двумя классами).

	Предсказано отрицательным	Предсказано положительным
Действительно отрицательные	TN True Negative	FP False Positive
Действительно положительные	FN False Negative	TP True Positive

Таблица 5. Матрица ошибок классификации

Строки - фактический класс , а столбцы — это прогнозируемый класс.

TN - число отрицательных примеров, предсказанных правильно.

FP - число отрицательных примеров, неправильно классифицированных как положительные.

FN - число положительных примеров, неправильно классифицированных как отрицательные.

TP - число положительных примеров, предсказанных правильно.

На Рисунке 6 представлен пример матрицы. Данный результат был получен при помощи Catboost и SMOTE

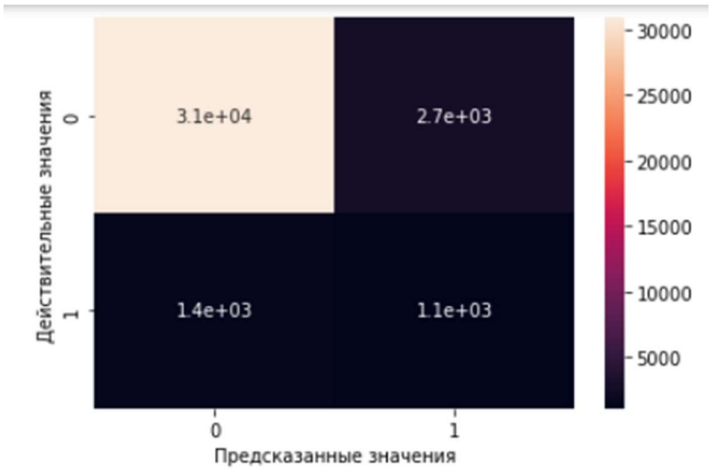

Рисунок 6. Матрица ошибок

Делать выводы по матрицам ошибок достаточно сложно, ведь количество моделей и объем данных слишком велик, чтобы визуально все сопоставить. Гораздо логичнее будет сравнить метрики.

Метрика достоверности предсказания

Accuracy = (TN + TP)/(TN + FP + TP + FN)

	Catboost	AdaBoost	GradBoost	RandomForest	LGBM	XGB	KNN
SMOTE	0.886755	0.878163	0.879133	0.893351	0.885646	0.885729	0.733793
SMOTEN	0.891272	0.837976	0.847149	0.927192	0.871206	0.894543	0.867853
ADASYN	0.884066	0.877193	0.879771	0.891272	0.880574	0.884759	0.726975
BorderlineSMOTE	0.893490	0.872953	0.876805	0.902275	0.885563	0.891938	0.794213
SVMSMOTE	0.912392	0.899365	0.912780	0.917907	0.905795	0.910479	0.828552
SMOTEENN	0.856656	0.824257	0.828054	0.863612	0.848230	0.853275	0.666556
SMOTETomek	0.888390	0.881378	0.880630	0.894016	0.884621	0.884648	0.734597

Таблица 5. Метрики достоверности предсказания.

Метрика точности

Precision = TP / (TP + FP)

	Catboost	AdaBoost	GradBoost	RandomForest	LGBM	XGB	KNN
SMOTE	0.288508	0.292189	0.290934	0.305257	0.301739	0.285563	0.100877
SMOTEN	0.256701	0.193726	0.190152	0.444444	0.227319	0.262792	0.126775
ADASYN	0.281884	0.285841	0.291315	0.299433	0.290075	0.285194	0.100696
BorderlineSMOTE	0.306601	0.279465	0.288613	0.329883	0.299654	0.301370	0.112756
SVMSMOTE	0.366748	0.336912	0.385796	0.396261	0.352318	0.359721	0.121736
SMOTEENN	0.255729	0.229543	0.233223	0.266718	0.250294	0.250403	0.096987
SMOTETomek	0.294038	0.298843	0.294239	0.307737	0.300674	0.284897	0.102284

Таблица 6. Метрики точности

Метрика полноты

Recall = TP / (TP + FN)

	Catboost	AdaBoost	GradBoost	RandomForest	LGBM	XGB	KNN
SMOTE	0.429597	0.529717	0.514559	0.419226	0.491424	0.429198	0.357798
SMOTEN	0.297966	0.421221	0.368169	0.191464	0.355804	0.286797	0.153171
ADASYN	0.431990	0.512166	0.509773	0.421619	0.496609	0.437176	0.369366
BorderlineSMOTE	0.422417	0.524930	0.527722	0.394097	0.483845	0.421221	0.285600
SVMSMOTE	0.358995	0.463103	0.431193	0.346629	0.424412	0.369765	0.236139
SMOTEENN	0.556442	0.648983	0.644595	0.550459	0.593538	0.557639	0.457120
SMOTETomek	0.432788	0.525329	0.513363	0.420423	0.498205	0.437176	0.362585

Таблица 7. Метрики полноты

Метрика F1

Среднее гармоническое между Precision и Recall

	Catboost	AdaBoost	GradBoost	RandomForest	LGBM	XGB	KNN
SMOTE	0.345192	0.376631	0.371704	0.353277	0.373900	0.342948	0.157382
SMOTEN	0.275798	0.265393	0.250781	0.267633	0.277406	0.274270	0.138728
ADASYN	0.341156	0.366910	0.370757	0.350174	0.366230	0.345197	0.158250
BorderlineSMOTE	0.355310	0.364745	0.373149	0.359142	0.370099	0.351356	0.161680
SVMSMOTE	0.362830	0.390055	0.407233	0.369787	0.385019	0.364673	0.160651
SMOTEENN	0.350414	0.339135	0.342518	0.359328	0.352106	0.345612	0.160022
SMOTETomek	0.350169	0.380966	0.374074	0.355361	0.375019	0.344980	0.159558

Таблица 8. Метрики F1

Метрика Accuracy высока у всех моделей, однако лучше всего себя показал классификатор RandomForest и балансировщик SVMSMOTE, а лучшая комбинация – SMOTEN + RandomForest. Метрики точности, полноты и F1 низки у всех моделей. Среди классификаторов можно выделить вновь RandomForest (по точности), GradBoost (по полноте и F1); среди балансировщиков - SVMSMOTE (по точности и F1), SMOTEENN (по полноте). Однако о выборе модели говорить еще рано.

Метрика штрафов

Для легкости интерпретации мною придумана и введена новая метрика штрафов – $F = 5 * FP + 1 * FN$

Смысл её состоит в том, что для бизнеса неплатёжеспособный клиент, предсказанный обратным, приносит в разы больше потерь, чем в обратном случае. Поэтому общая оценка модели будет заключаться в суммарных штрафных баллах посчитанной на основе матрицы ошибок классификации. И именно на эту метрика я буду смотреть в основном при выборе модели. Результат расчета этой метрики можно увидеть в Таблице 9.

	Catboost	AdaBoost	GradBoost	RandomForest	LGBM	XGB	KNN
SMOTE	9668	9293	9069	9680	9372	9674	16362
SMOTEN	11149	11852	11778	10785	11280	11091	13093
ADASYN	9764	9402	9307	9672	9470	9829	16357
BorderlineSMOTE	9687	9508	9105	9610	9247	9742	14755
SVMSMOTE	9716	9165	8885	9473	9155	9642	13970
SMOTEENN	9585	9928	9535	9416	9428	9858	17732
SMOTETomek	9577	9083	9130	9652	9272	9762	16160

Таблица 9. Метрики штрафов

Как видно из результатов:

- наилучшая комбинация – классический градиентный бустинг (GradBoost) после применения перебалансировки с помощью SVMSMOTE
- наилучшая модель - LGBM
- наилучший балансировщик - SVMSMOTE
- наихудшая модель – KNN
- наихудший балансировщик – SMOTEN

Бустинги оказались наиболее эффективными. Общая архитектура бустинговых моделей заключается в последовательном построении

классических моделей машинного обучения, где каждая последующая модель предсказывает разницу предсказания и реального значения предыдущей модели.

$$F_1(X) = y_1, \qquad F_2(X) = y - \widehat{y_1}, \dots, F_i(X) = e_{n-1}$$

Где e_n – ошибка полученная n – ной моделью.

4.4 Подбор параметров и финальный результат

Теперь, когда мы нашли наилучшую комбинацию, нам осталось подобрать гиперпараметры для этой модели. Для этого мною был использован такой инструмент как GridSearchCV. Он находит наилучшие параметры, путем обычного перебора - создает модель для каждой возможной комбинации параметров. Перебор параметров может занять достаточно много времени (у меня занял 30 мин), однако он очень важен, ведь их изменение может принципиально повлиять на ее качество.

К сожалению, в нашей модели после подбора параметров все метрики кроме полноты стали хуже (Рисунок 7, Рисунок 8). Особенно сильно это повлияло на метрику штрафов. Подбор параметров не требовался.

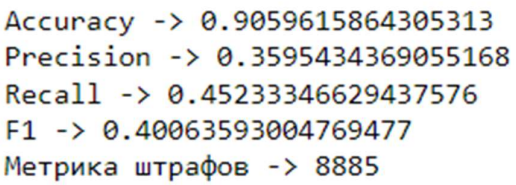

Рисунок 7. Метрики до подбора параметров

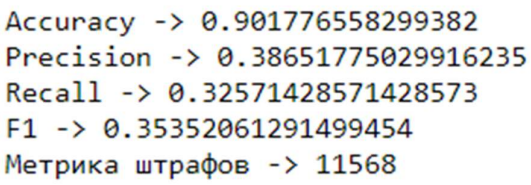

Рисунок 8. Метрики после подбора параметров

Таким образом, наилучшее сочетание - градиентный бустинг (GradBoost) после применения перебалансировки с помощью SVMSMOTE без подбора параметров (GridSearchCV). Полученная модель имеет оценку 8885 у.е. стоимости издержек для компании. По сравнению с другими комбинациями, разрыв большой, потому что каждая денежная единица на счету.

Минусом модели является лишь то , что относительно других моделей она достаточно долго обучается – 88 секунд (Сравнение времени обучения можно увидеть в Таблице 10). И в принципе перебалансировщик SVMSMOTE является самым долгим. Однако, на мой взгляд, качество модели гораздо важнее времени обучения.

	Catboost	AdaBoost	GradBoost	RandomForest	LGBM	XGB	KNN
SMOTE	24	15	31	29	0	7	1
SMOTEN	62	49	58	62	39	41	37
ADASYN	25	15	30	29	1	8	1
BorderlineSMOTE	24	15	30	29	1	8	1
SVMSMOTE	84	75	88	107	95	100	90
SMOTEENN	46	37	52	51	22	28	21
SMOTETomek	45	38	59	59	17	25	18

Таблица 10. Время обучения моделей

ЗАКЛЮЧЕНИЕ

В ходе написания данной курсовой работы, были изучены и представлены к обзору популярные подходы машинного обучения в задачах кредитного скоринга. Были успешно выполнены все поставленные задачи, в результате чего, сделан вывод: лучше всего использовать градиентный бустинг и SVMSMOTE. Балансировщик SVMSMOTE уверенно лидирует среди своих аналогов. В случае же классификатора – вопрос надлежит дальнейшему исследованию – сравнению работы на разных наборах данных. В работе не были использованы архитектуры из глубинного обучения, что по мнению автора – лучший исход. Как повод для дальнейших исследований – эксперименты со слоями балансировки.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. М.В. Коротеев. Учебное пособие по дисциплине «Анализ данных и машинное обучение» - 2018.
2. Liudmila Prokhorenkova. «CatBoost: unbiased boosting with categorical features» - 2019
3. Abraham J. Wyner. «Explaining the Success of AdaBoost and Random Forests as Interpolating Classifiers» - 2018
4. J. Elith. «A working guide to boosted regression trees» - 2008 - Journal of Animal Ecology - Wiley Online Library
5. Tim Kam Ho. «Random Decision Forests»
6. Guolin Ke. «LightGBM: A Highly Efficient Gradient Boosting Decision Tree (neuripscc)»
7. H Han, W Wen-Yuan, M Bing-Huan. «Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning» Advances in intelligent computing - 2005
8. A. Geron. Hand on Machine Learning with scikit-learn and Tensorflow - 2017 (564p)
9. C. Albon. Machine learning with Python Handbook - 2018 (427p)
10. L.P. Coelho, W. Richert. Building machine learning systems with Python - 2015 (326p)

ПРИЛОЖЕНИЕ

Приложение 1. Технические показатели

AMD Ryzen 5 3500U 2.1 ГГц

RAM 8,00 ГБ

Приложение. 2 Код

```
!pip install catboost
!pip install lightgbm --install-option=--gpu
!pip install -U xgboost

!wget https://s3-us-west-2.amazonaws.com/xgboost-wheels/xgboost-0.81-py2.py3-none-manylinux1_x86_64.whl

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from sklearn.model_selection import *
from sklearn.cross_decomposition import *
from sklearn.ensemble import *
from sklearn.svm import *
from sklearn.linear_model import *
from sklearn.gaussian_process import *
from sklearn.isotonic import *
from sklearn.kernel_ridge import *
from sklearn.pipeline import *
from sklearn.preprocessing import *
from sklearn.neighbors import *
```

```
from sklearn.neural_network import *
from sklearn.tree import *

import imblearn
from imblearn.combine import *
from imblearn.over_sampling import *
from imblearn.pipeline import Pipeline

from tqdm.auto import tqdm
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
import xgboost as xgb

from tqdm.auto import tqdm

import time

data = pd.read_csv('cs-training.csv', index_col = 'Unnamed: 0')
data.head()

data.shape

data.dtypes

data.isna().sum()

data.dropna(inplace=True)
data.shape

data.isna().sum()
```

```
plt.rcParams['figure.figsize'] = (30, 30)
for i, col in enumerate(data.columns[2:]):
    plt.subplot(5, 3, i+1)
    h = data[col].hist(bins=100)
    h.set_title("Распрпеделение " + col, size = 16)
    h.set_xlabel(col, size = 13)
    h.set_ylabel('Count', size = 13)
    h.get_figure()

plt.figure(figsize=(8, 5))
sns.heatmap(data.corr().abs(), vmin=0, vmax=1, annot=True, fmt=".2f");

data.describe()

classifiers = {
            'Catboost':CatBoostClassifier(verbose=False),
            'AdaBoost' : AdaBoostClassifier(n_estimators=100),
            'GradBoost' : GradientBoostingClassifier(),
            'RandomForest' : RandomForestClassifier(),
            'LGBM' : LGBMClassifier(),
            'XGB' : xgb.XGBClassifier(),
            'KNN' : KNeighborsClassifier()
            }

oversamplers = {
             'SMOTE' : SMOTE(),
             'SMOTEN' : SMOTEN(),
             'ADASYN' : ADASYN(),
             'BorderlineSMOTE' : BorderlineSMOTE(),
```

```
            'SVMSMOTE' : SVMSMOTE(),
            'SMOTEENN':SMOTEENN(),
            'SMOTETomek':SMOTETomek()
            }

X = data.drop('SeriousDlqin2yrs', axis=1)
y = data['SeriousDlqin2yrs']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify = y)

As = pd.DataFrame(index = oversamplers.keys(),
                  columns = classifiers.keys())
Ps = pd.DataFrame(index = oversamplers.keys(),
                  columns = classifiers.keys())
Rs = pd.DataFrame(index = oversamplers.keys(),
                  columns = classifiers.keys())
F1s = pd.DataFrame(index = oversamplers.keys(),
                  columns = classifiers.keys())
times = pd.DataFrame(index = oversamplers.keys(),
                  columns = classifiers.keys())
results = pd.DataFrame(index = oversamplers.keys(),
                  columns = classifiers.keys())
matrices = []

for i in tqdm(oversamplers.keys()):
   for j in tqdm(classifiers.keys()):

      pipeline = Pipeline(
         [
          ('oversamplingmethod', oversamplers[i]),
          ('model', classifiers[j])
```

```
            ],
            verbose = False
        )

        start = time.time()
        pipeline.fit(X_train, y_train)
        end = time.time()

        y_pred = pipeline.predict(X_test)

        As.loc[i, j] = accuracy_score(y_test, y_pred)
        Ps.loc[i, j] = precision_score(y_test, y_pred)
        Rs.loc[i, j] = recall_score(y_test, y_pred)
        F1s.loc[i, j] = f1_score(y_test, y_pred)
        times.loc[i, j] = round(end - start, 2)

        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        results.loc[i, j] = fp * 1 + fn * 5

        matrices.append(confusion_matrix(y_test, y_pred))

%matplotlib inline
for matrix in matrices:
    sns.heatmap(pd.DataFrame(
    matrix),
    annot=True)
    plt.ylabel('Действительные значения')
    plt.xlabel('Предсказанные значения')
    plt.show()
```

```
As.style.background_gradient().set_properties(**{'font-size': '15px'})

Ps.style.background_gradient().set_properties(**{'font-size': '15px'})

Rs.style.background_gradient().set_properties(**{'font-size': '15px'})

F1s.style.background_gradient().set_properties(**{'font-size': '15px'})

times.style.background_gradient().set_properties(**{'font-size': '15px'})

results = results.astype(int)
results.style.background_gradient().set_properties(**{'font-size': '15px'})

def custom_loss(y_pred, y_test):
    tn, fp, fn, tp = confusion_matrix(y_pred, y_test).ravel()
    return (fp * 1 + fn * 5)

print(f"Accuracy -> {As['GradBoost']['SVMSMOTE']}")
print(f"Precision -> {Ps['GradBoost']['SVMSMOTE']}")
print(f"Recall -> {Rs['GradBoost']['SVMSMOTE']}")
print(f"F1 -> {F1s['GradBoost']['SVMSMOTE']}")
print(f"Метрика штрафов -> {results['GradBoost']['SVMSMOTE']}")

X_sm, y_sm = SVMSMOTE().fit_resample(X_train, y_train)

GB_model = GradientBoostingClassifier()

parametrs = {'loss': ['deviance', 'exponential'],
             "learning_rate": [0.1, 0.25, 0.5, 1],
             'n_estimators': [25, 50, 100, 150],
```

```
    'subsample':[0.5, 1]
}

grid = GridSearchCV(GB_model, parametrs, n_jobs = -1, scoring = 'f1')
grid.fit(X_sm, y_sm)
grid.best_params_

GB_model = GradientBoostingClassifier(**grid.best_params_)
GB_model.fit(X_sm, y_sm)
y_pred = GB_model.predict(X_test)

print(f'Accuracy -> {accuracy_score(y_pred, y_test)}')
print(f'Precision -> {precision_score(y_pred, y_test)}')
print(f'Recall -> {recall_score(y_pred, y_test)}')
print(f'F1 -> {f1_score(y_pred, y_test)}')
print(f'Метрика штрафов -> {custom_loss(y_pred, y_test)}')
```