федеральное государственное бюджетное образовательное учреждение высшего образования
«Вологодский государственный университет»

Институт математики, естественных и компьютерных наук
(наименование института)
Кафедра математики
(наименование кафедры)

КУРСОВОЙ ПРОЕКТ/РАБОТА

Дисциплина: «Компьютерные технологии»
Наименование темы: «Разработка тестового стенда для анализа алгоритмов, основанных на SMOTE»

01.03.02.	43.03	8	1	2025
код направления подготовки/ специальности	код выпускающей кафедры	регистрационный номер по журналу	код формы обучения	год

Руководитель — доцент, Волкова Светлана Сергеевна
(уч. степень, звание, должность. Ф.И.О)

Выполнил (а) студент — Кожин Арсений Романович
(Ф.И.О)

Группа, курс — 4Б01 ПМ-41

Дата сдачи — 10.12.2025

Дата защиты _____

Оценка по защите _____
(подпись преподавателя)

Вологда
2025г.

Содержание

1 Введение **2**
2 Постановка задачи **2**
3 Требования к тестовому стенду **3**
4 Постановка и методика эксперимента **4**
5 Реализация **11**
6 Заключение **24**
7 Список использованных источников **25**
8 Приложения **28**

1 Введение

Одной из основных проблем задач машинного обучения является проблема дисбаланса данных в обучающих выборках, когда количество представителей одного класса мало, относительно количества представителей других классов. Это приводит к снижению качества прогнозирования для класса меньшинства и смещению метрик классификатора. Эту проблему решают алгоритмы синтетической генерации данных – SMOTE [1], имеющий множество расширений. Количества вариаций алгоритма растет и нужно как-то исследовать какой алгоритм работает лучше, на каком наборе данных возникают трудности, воспроизводимость результатов статей и оценку влияния параметров на качество синтетических примеров. Эту задачу решает мой курсовой проект - Тестовый стенд алгоритмов балансировки данных, основанных на SMOTE.

2 Постановка задачи

Разработка и реализация архитектуры, позволяющей проводить исследования алгоритмов, основанных на SMOTE. Архитектура должна контролироваться конфигурационными файлами, позволять проводить эксперименты алгоритмов и структурировано приводить результаты этих экспериментов, должна быть возможность сравнения экспериментов между собой, контролировать воспроизводимость экспериментов и удобный интерфейс для сравнения результатов.

3 Требования к тестовому стенду

1. **Архитектурные требования:**

1.1. модульность - проект должен иметь четкое разделение на компоненты

1.2. воспроизводимость экспериментов

1.3. масштабируемость - архитектура должна позволять легко добавлять новые алгоритмы, наборы данных, метрики.

2. **Системные требования:**

2.1. Версионирование наборов данных

2.2. Реализации алгоритмов

2.3. Система экспериментов - конфигурационные файлы для настройки параметров, запуск одиночный и пакетных тестирований.

3. **Логирование отчетов по эксперименту:**

3.1. Характеристики наборов данных

3.2. Конфигурации экспериментов

3.3. Таблица результатов работы алгоритма

3.4. Таблицы агрегированных результатов

3.5. Визуализация графиков

4 Постановка и методика эксперимента

Постановка эксперимента сводится к созданию конфигурационного файла, в котором указывается вся необходимая информация для проведения эксперимента: какие методы балансировки данных участвуют в эксперименте, на каких наборах данных, обработанных данных или сырых, параметры эксперимента, чтобы сохранить одинаковые условия для каждого эксперимента из этого пакета: cv_folds - количество фолдов в кросс-валидации, test_size - размер тестовой выборки, random_state - seed для реализации рандома, priority_metrics - какие метрики отслеживаются в эксперименте, selected_classifiers - на каких моделях классификатора проверяется работоспособность метода. Заполнение в файл этих параметров достаточно для создания эксперимента, однако, возможен запуск конфигурационного файла, ограниче нного названием метода и датасета, а остальные параметры будут выбраны по умолчанию.

Для запуска эксперимента используется файл main.py. Он считывает названия конфигурационного файла с консоли и дальше все выполняется программно, от пользователя больше ничего не требуется, последующие данные будут автоматически формироваться в ClearML [2] и вся аналитика продолжается там.

Пример запуска: *python main.py experiment.yaml*

При запуске эксперимента в сервисе ClearML будет создана задача с названием: Dataset name + Method name, с соответствующими тегами. В этой задаче будут сохранены файлы конфигурации, артефакты, информация по задаче, консольный вывод выполнения скрипта, метрики и графики.

На вход подается информация из конфига: названия набора данных, на котором проводится эксперимент, метод балансировки данных, параметры набора данных, параметры метода и конфигурационный файл эксперимента. В начале эксперимента происходит связь с ClearML,

настройка логирования и сохранение конфигов, затем загружаются данные из ClearML и разделяются на признаки и целевую переменные. Данные разделяются train_test_split(), используя стратификацию - разделение данных с сохранением баланса классов до разделения. После разделения данных используется кросс-валидация: сначала на данных без использования балансировки, а затем с методом, указанным в эксперименте. После кросс-валидации происходит тестирование работы алгоритма на тестовых данных. Получив все результаты, строится таблица метрик и создаются визуализации, и сохраняются результаты эксперименты в артефакты задачи в ClearML.

Кросс-валидация происходитс следующим образом, используя рекомендации из статьи Santos et al [3]: тренировочные данные разбиваются *StratifieldKFold() [4]* на K фолдов, где K контролируется конфигом эксперимента. Используется именно эта вариация кросс-валидации для сохранения баланса классов в каждом фолде. Затем для каждого классификатора, указанного в конфиге, строится результат кросс-валидации. По каждому разделению на фолды данные разделяются на тренировочную и валадитационную. Тренировочные данные балансируются алгоритмом, текущий классификатор обучается на этих данных и строятся предсказания модели, вычисляются выбранные в конфигурационном файле метрики и сохраняются результаты.

Metrics Summary - Safe_Level_SMOTE - Satimage Iteration 1

Classifier	Metric	Original	Safe_Level_SMOTE	Delta_Absolute	Delta_Percent
RandomForest	balanced_accuracy	0.7508	0.8582	0.1073	14.3
RandomForest	f1_weighted	0.937	0.9342	-0.0028	-0.29
RandomForest	g_mean	0.7118	0.8534	0.1416	19.89
RandomForest	roc_auc_weighted	0.963	0.9672	0.0042	0.43
RandomForest	precision_weighted	0.9392	0.9395	0.0003	0.03
RandomForest	recall_weighted	0.9433	0.9308	-0.0124	-1.32
kNN	balanced_accuracy	0.8165	0.8933	0.0768	9.41
kNN	f1_weighted	0.9387	0.8911	-0.0477	-5.08
kNN	g_mean	0.8021	0.8929	0.0908	11.32
kNN	roc_auc_weighted	0.9302	0.9448	0.0146	1.57
kNN	precision_weighted	0.9382	0.9354	-0.0028	-0.3
kNN	recall_weighted	0.9394	0.8718	-0.0676	-7.2
LogisticRegression	balanced_accuracy	0.4996	0.7109	0.2113	42.3
LogisticRegression	f1_weighted	0.8564	0.6855	-0.1709	-19.96
LogisticRegression	g_mean	0	0.6991	0.6991	0
LogisticRegression	roc_auc_weighted	0.7661	0.7787	0.0126	1.65
LogisticRegression	precision_weighted	0.8151	0.8942	0.0791	9.7
LogisticRegression	recall_weighted	0.9021	0.6068	-0.2953	-32.73

Рисунок 1 - Таблица улучшений метрик после использования алгоритма балансировки данных.

Итоговая оценка происходит похожим образом: классификатор обучается на исходных данных и такой же классификатор обучается на уже сбалансированных данных. Для этих классификаторов вычисляются метрики и считается улучшение метрик после использования алгоритма балансировки. Эти данные логируются в ClearML, строится таблица улучшений для каждого эксперимента, в соответствии с рисунком 1, которая состоит из 6 столбцов :

1) название классификатора, на котором измерялись метрики.

2) исследуемые метрики.

3) значение метрики на оригинальном наборе данных.

4) значение метрики на наборе данных, сбалансированном методом, который находится в названии столбца.

5) численное различие метрик

6) процентрое различие метрик.

по данным таблицы можно наглядно видеть было ли улучшение работы классификатора после балансировки данных; строятся графики, значений метрик на каждом из фолдов при кросс-валидации, в соответствии с рисунком 2: по оси x – номер фолда, по y - значение метрик. В легенде каждого графика находятся используемые метрики. Формируется ровно

столько графиков, сколько указано классификаторов в конфигурационном файле эксперимента.

Рисунок 2 - Пример логирования результатов кросс-валидации эксперимента.

После окончания эксперимента будет создана задача results, в которой будет создано N таблиц, которые будут отображаться, как на *Рисунок 3* в правом окне интерфейса, где N – количество наборов данных в конфигурационном файле эксперимента. Таким образом, при пакетном тестировании алгоритмов: например, 5 методов и 5 наборах данных, в задаче results будет создано 5 таблиц с названиями наборов данных. Каждая таблица, как показано в таблице 1, которая содержит в себе названия методов балансировки данных, по которым проводились эксперименты, классификаторы, по которым считались метрики, и сами значения метрик, указанных в конфигурационном файле эксперимента.

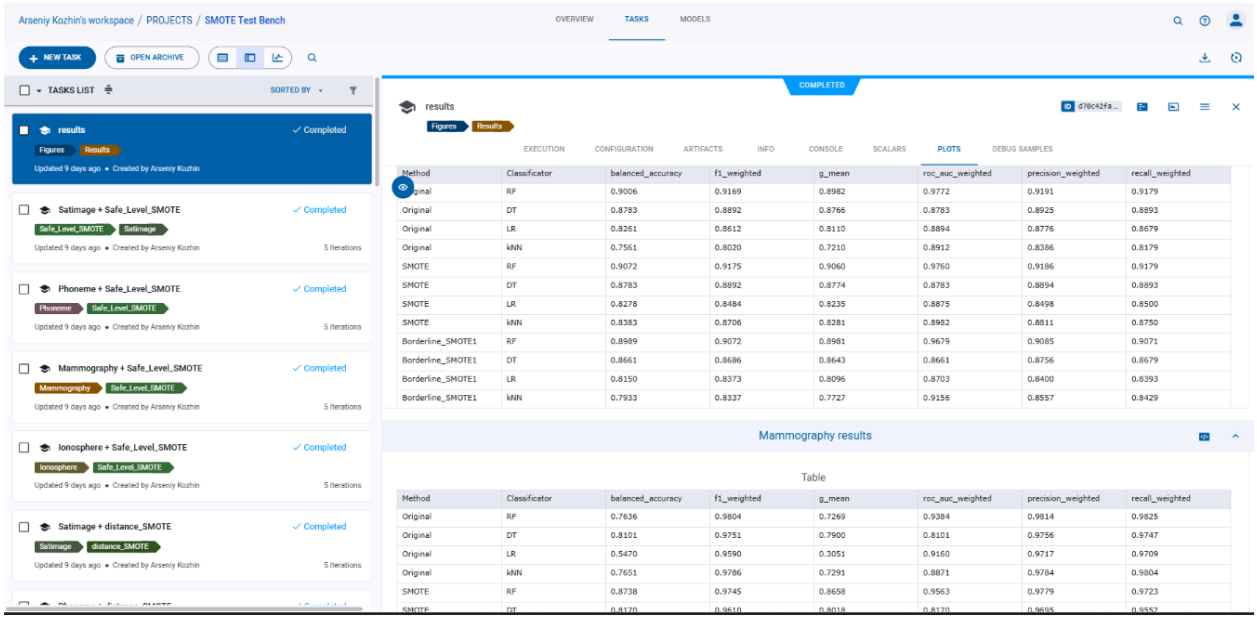

Рисунок 3 – Задача results в интерфейсе ClearML.

Method	Classificator	balanced_accuracy	f1_weighted	g_mean	roc_auc_weighted	p
Original	RF	0.9006	0.9169	0.8982	0.9772	0
	DT	0.8783	0.8892	0.8766	0.8783	0
	LR	0.8261	0.8612	0.8110	0.8894	0
	kNN	0.7561	0.8020	0.7210	0.8912	0
SMOTE	RF	0.9072	0.9175	0.9060	0.9760	0
	DT	0.8783	0.8892	0.8774	0.8783	0
	LR	0.8278	0.8484	0.8235	0.8875	0
	kNN	0.8383	0.8706	0.8281	0.8982	0
Borderline_SMOTE1 [5]	RF	0.8989	0.9072	0.8981	0.9679	0
	DT	0.8661	0.8686	0.8643	0.8661	0
	LR	0.8150	0.8373	0.8096	0.8703	0
	kNN	0.7933	0.8337	0.7727	0.9156	0
distance_SMOTE [6]	RF	0.9044	0.9142	0.9032	0.9704	0
	DT	0.8611	0.8653	0.8603	0.8611	0
	LR	0.8228	0.8446	0.8179	0.8858	0
	kNN	0.8361	0.8698	0.8241	0.9087	0
SafeLevelSMOTE [7]	RF	0.9067	0.9143	0.9058	0.9775	0
	DT	0.8667	0.8720	0.8656	0.8667	0
	LR	0.8322	0.8606	0.8240	0.8942	0
	kNN	0.8011	0.8413	0.7814	0.8981	0

Таблица 1 Пример таблицы результатов, полученных в задаче results.

В ходе проведения экспериментов было замечено, что в большинстве случаем балансировка данных не дает улучшений, в соответствии с таблицей 1, по большинству наблюдаемых метрик. Из-за этого возникли вопросы о корректности проведения эксперимента и исследования причин ухудшения метрик. Изучив модули постановки эксперимента, ошибок выявлено не было, поэтому

рассматривались данные, которые логировались в таблицах: в соответствии с рисунком 4, отображающем на скольких наборах данных после балансировки значение наблюдаемой метрики на классификаторе, улучшилось или ухудшилось, где зеленый цвет указывает на улучшение и сколько данных было улучшено, серый – улучшения не наблюдались, красное – ухудшение значения метрики, видно, что по метрике g_mean(отображает общую точность модели) результаты классификатора после оверсемплинга улучшались почти всегда, а метрики, по которым наблюдались ухудшения были подсчитаны в weighted формате, т.е. метрики учитывали дисбаланс классов в своих расчетах. Исходя из этой информации было выдвинута гипотеза о том, что вместо weighted усреднения корректнее будет использовать macro усреднение, именно при этом усреднении метрики могут сильно падать, если один класс предсказывается плохо, но без учета размеров (с одинаковым весом для всех классов).

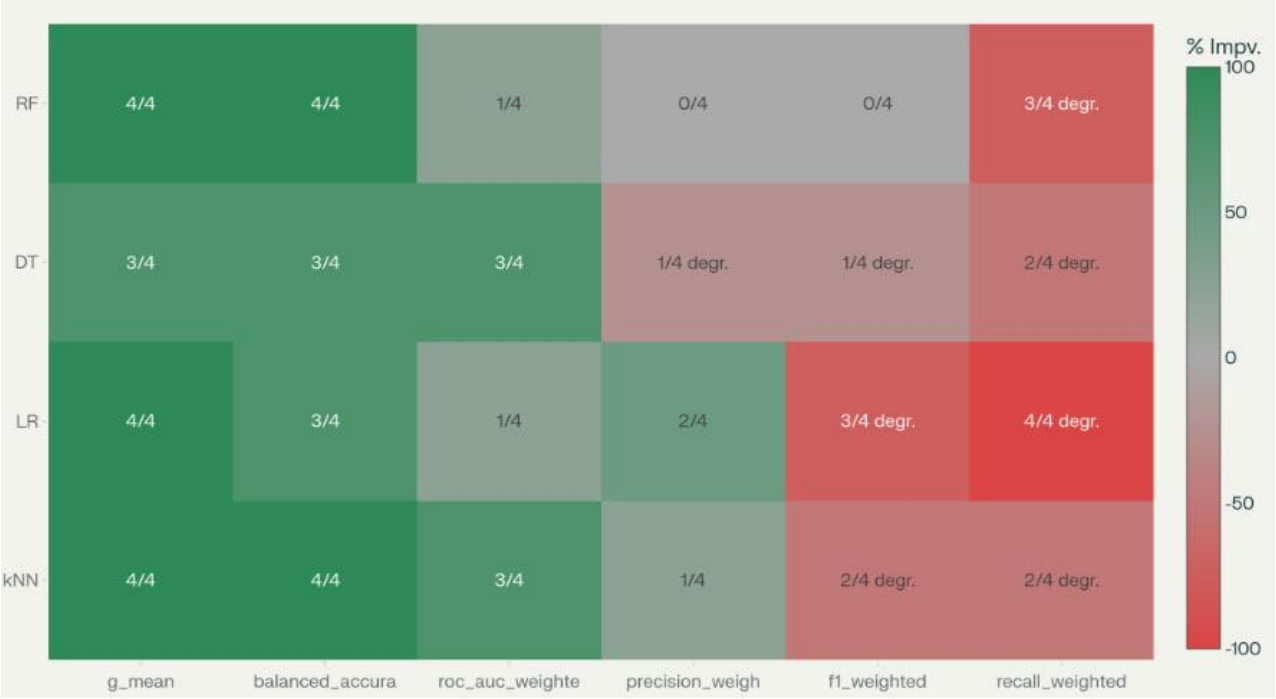

Рисунок 4 - Тепловая карта изменений показателей улучшений метрик на классификаторах.

Были проведены эксперименты, которые подсчитывают метрики с macro усреднением и получены данные, указанные на рисунке 5, на которых наблюдается, что происходит при использовании macro усреднения вместо weighted, наблюдается влияние алгоритмов балансировки данных на работу классификаторов: в большинстве случаем f1_macro падает после балансировки,

precision_macro падает, а recall_macro растет. Такое поведение объясняется тем, что SMOTE добавляют примеры в миноритарные классы, поэтому модель начинает чаще их предсказывать, значит recall по этим классам растёт, но при этом падает precision по другому классу, поэтому можем упасть по среднему f1. т.е. мы сделали модель более чувствительной, но менее точной. Исходя из этих экспериментов было выявлено, что реализация экспериментов является корректной, и при сравнении стоит ориентироваться на g_mean и balansed_acc, как более отражающие баланс.

Classifier	Metric	Original	SMOTE	Delta_Absolute	Delta_Percent
RandomForest	balanced_accuracy	0.7921	0.8931	0.1009	12.74
RandomForest	f1_macro	0.8404	0.8384	-0.002	-0.24
RandomForest	g_mean	0.7654	0.8886	0.1232	16.09
RandomForest	roc_auc_macro	0.9514	0.9627	0.0114	1.2
RandomForest	precision_macro	0.9098	0.7981	-0.1117	-12.27
RandomForest	recall_macro	0.7921	0.8931	0.1009	12.74
kNN	balanced_accuracy	0.7817	0.8799	0.0982	12.57
kNN	f1_macro	0.8246	0.7449	-0.0797	-9.66
kNN	g_mean	0.7521	0.8766	0.1245	16.56
kNN	roc_auc_macro	0.9122	0.9213	0.0091	1
kNN	precision_macro	0.8847	0.6864	-0.1983	-22.42
kNN	recall_macro	0.7817	0.8799	0.0982	12.57
LogisticRegression	balanced_accuracy	0.5879	0.8823	0.2944	50.08
LogisticRegression	f1_macro	0.6406	0.6534	0.0128	2
LogisticRegression	g_mean	0.4199	0.8821	0.4622	110.05
LogisticRegression	roc_auc_macro	0.9491	0.9452	-0.0039	-0.41

Рисунок 5 - Таблица улучшений метрик при исследовании macro усреднений метрик, в интерфейсе ClearML.

5 Реализация

5.1 Архитектура системы, основные модули

Главная идея реализации - интеграция логирования с сервисом ClearML. Он позволяет реализовать задачи, поставленные перед проектом, касаемые логирования и отчетов по экспериментам, проект реализован таким образом, что конечные результаты эксперимента наблюдаются интерфейсе сервиса. На рисунке 6 проиллюстрировано как происходит функционирование системы: модули обозначены черными большими

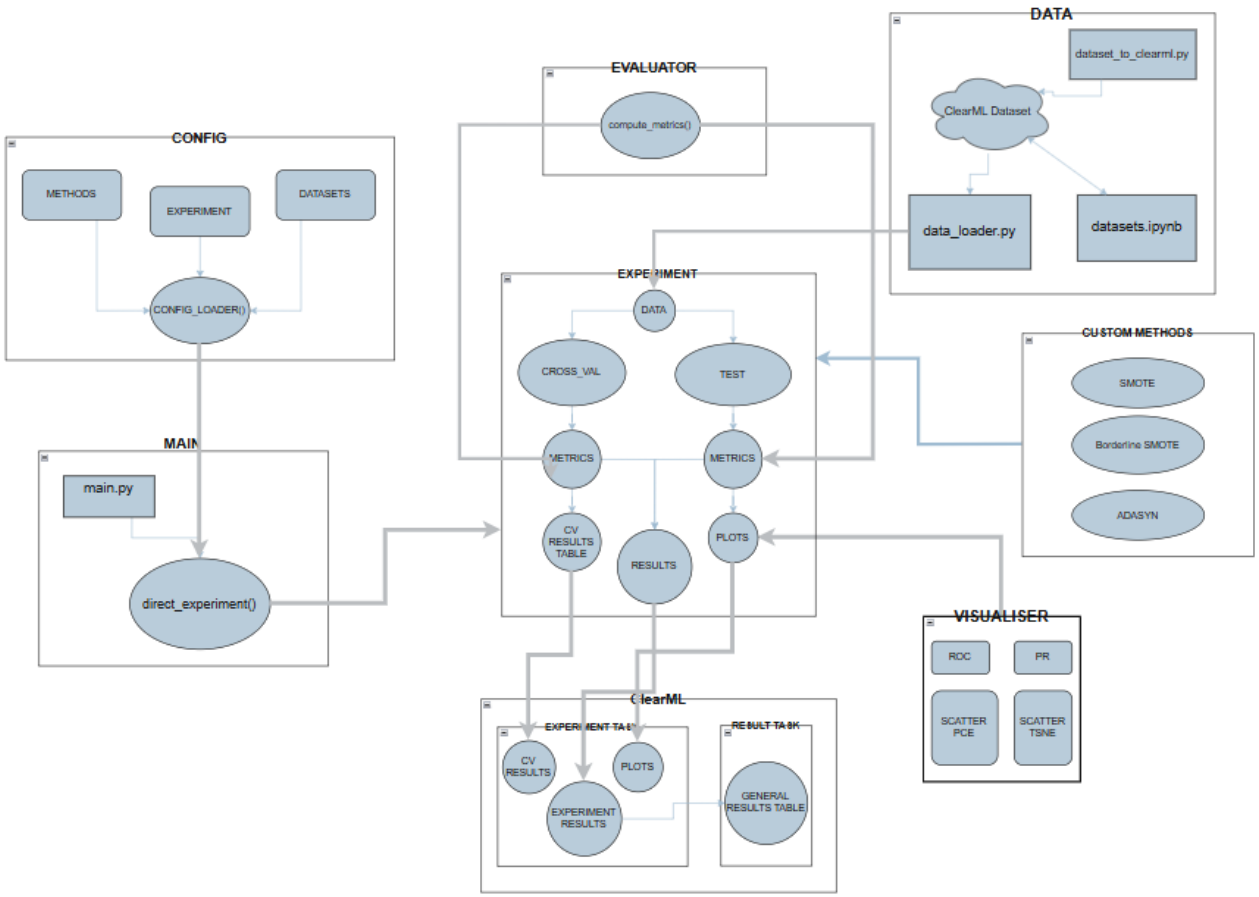

Рисунок 6 – Схема реализованных модулей и их взаимодействия.

прямоугольниками с названием на верхней грани фигуры;

прямоугольниками с закругленными углами обозначены конфигурационные файлы; овалы – реализованные функции в модулях; прямоугольные фигуры с закрашенным фоном – исполняемые .py файлы; ClearML Dataset – облачное хранение наборов данных; связи между модулями обозначены широкими стрелочками; связи внутри модулей – узкими стрелочками; кружками с названием внутри обозначены вызовы функций из других модулей. В конечном итоге все полученные на этапе эксперимента данные попадают в ClearML. Для того, чтобы реализовать такую систему в ClearML был создан свой Workspace и проект для работы с тестовым стендом.

На текущем этапе проекта реализованы следующие интеграции с ClearML:

1. Хранений и версионирование наборов данных в среде фреймворка, которые отображаются в соответствии с рисунком 7: у каждого набора данных описывается название, количество версий, размер. Это позволяет иметь удаленный доступ к наборам данных без потребности явного скачивания на других устройствах.

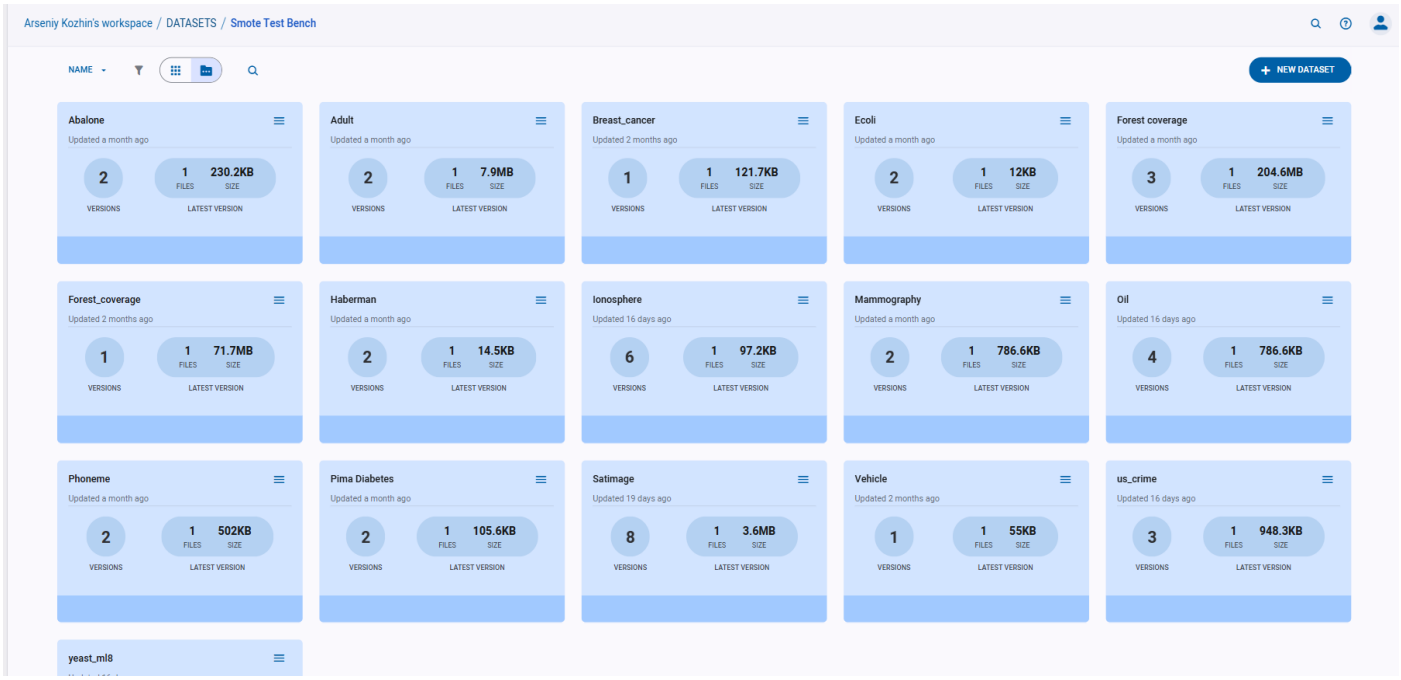

Рисунок 7 - Наборы данных, хранящиеся в ClearML.

2. Логирование экспериментов. В сервисе был создан проект: “SMOTE test bench”, в котором создаются задачи, в соответствии с рисунком 8, для каждого эксперимента создается задача, которая обозначена названием Dataset+Method и помечен тегами с названием набора данных и используемого метода в эксперименте. В каждой задаче предусмотрена возможность рассматривать детали эксперимента: конфигурационные файлы, артефакты, консольный вывод, скаляры, графики и дополнительные файлы, что очень полезно для поставленных перед проектом задач. Для навигации по задачам используются теги, в случае тестового стенда: название метода и набора данных.

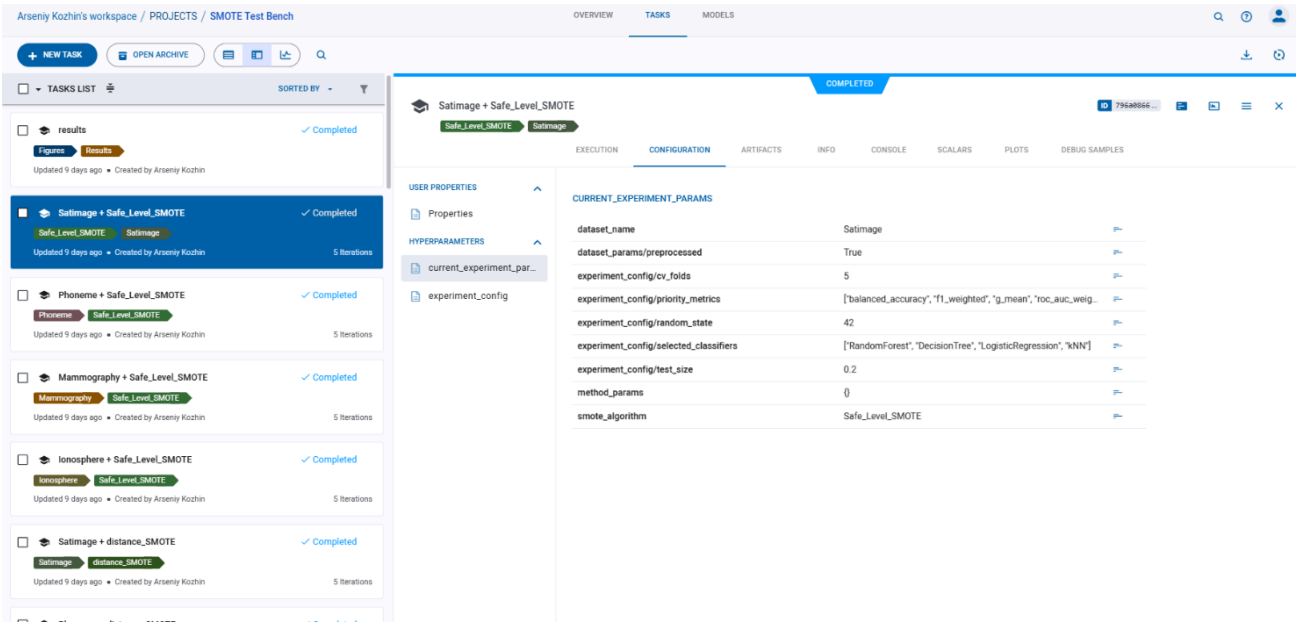

Рисунок 8 – Задачи в интерфейсе ClearML.

Эти возможности фреймворка позволили организовать централизованную систему логирования экспериментов и наборов данных для удобного использования и сравнения результатов.

Для работы с методами была выбрана библиотека smote_variants [8], где представлены 89 реализаций различных методов балансировки данных, основанных на SMOTE. Взятие готовых реализаций позволило

сконцентрироваться на реализации архитектуры и создания логики экспериментов.

5.2.1 Управление конфигурацией

Основой составляющей цепочки проведения эксперимента является задание параметров, по которым он будет проходить: над какими методами, над какими наборами данных, общие параметры, например, на сколько ячеек разбиваются данные в кросс-валидации. Такую общую информацию удобно создавать и хранить в конфигурационных файлах. Для работы с ними был выбрана библиотека yaml [9]. Это позволяет организовать работу тестового стенда без изменения .py файлов. Чтобы работать со стендом достаточно только изменить конфигурационный файл на нужные параметры и запустить проект. В проекте были реализованы следующие файлы:

configs/config_loader.py: модуль загрузки конфигурационных файлов для удобного использования. Он необходим, чтобы собирать конфигурационный файл эксперимента в удобный для языка программирования вид – словарь. Второе же применение – получение методов или наборов данных по названию, чтобы пользователи не нужно было вводить точное название вызываемого метода из библиотеки smote_variants. С интеграцией конфигурационных файлов эксперименты стали проводится быстрее и требовать только изменения параметров в одном файле.

methods.yaml: конфигурационный файл методов балансировки данных. Содержит в себе словарь методов и соответствующие им функции в библиотеке smote_variants.

templates/methods.yaml: шаблон для добавления методов в конфигурационный файл эксперимента. В этом файле содержатся названия методов; нужен он для того, чтобы можно было скопировать интересующие

для эксперимента методы и вставить в конфигурационный файл эксперимента.

data/datasets.yaml: конфигурационный файл наборов данных, содержащий id в ClearML Datasets. Содержит в себе как id сырых данных, так и обработанных, чтобы можно было подгружать наборы данных с ClearML с необходимыми для эксперимента условиями.

experiment/experiment.yaml: конфигурационный файл эксперимента, который составляется в соответствии с рисунком 9: список исследуемых методов, наборов данных, флаг обработки данных, параметры экспериментов: количество фолдов в кросс-валидации, размер тестовой выборки, зерно для генерации случайности, набор исследуемых метрик, набор классификаторов.

```
methods:

datasets:

datasets_params:
    preprocessed:

experiment_config:
    cv_folds:
    test_size:
    random_state:
    priority_metrics:
    selected_classifiers:
```

Рисунок 9 - Шаблон конфигурационного файла эксперимента.

data/dataset_to_clearML.py: модуль загрузки и версионирования набора данных в систему ClearML. Подгружает данные из ClearML Datasets по параметрам из конфигурационного файла эксперимента.

5.2.2 Модули системы

utils/data_loader.py: модуль загрузки данных из ClearML Datasets. Рассмотрев параметры конфигурационного файла, этот модуль загружает

нужный набор данных по id, который соответствует ему в конфигурационном файле наборов данных.

utils/visualise.py: модуль визуализации для логирования. В нем реализован класс визуализатора с функциями создания scatter графиков для визуального представления как распределяются данные, а также отображения, где метод генерирует синтетические образцы. Реализованы два вида построения таких графиков: основанный на PCE [10], по которому два признака для визуализации выбираются методом PCE и получается график, показанный на рисунке 10 - интерактивный график, который создается для каждого эксперимента; основанный на TSNE [11], в соответствии с рисунком 11- интерактивный график, который создается для каждого эксперимента. Обе реализации логируется в интерфейсе ClearML и имеют одинаковую легенду: синие точки отражают оригинальное распределение данных первого класса, зеленые отражают оригинальное распределение данных второго класса, розовые точки показывают, какие точки создал алгоритм балансировки данных. Модуль позволяет отрисовывать ROC - рисунок 12, и PR кривые – рисунок 13. Обе кривые создаются для каждого эксперимента и логируется в интерфейсе ClearML, снизу находится легенда карты классификаторов: у каждого классификатора уникальный цвет и на графике строится две линии для него: сплошная - до применения метода балансировки данных, пунктирная – после применения метода. Все графики являются интерактивными и могут увеличивать масштаб, приближать, рассматривать отдельные куски графиков, отключать наблюдения, например, в графиках распределений можно убрать отображения точек мажоритарного класса и рассмотреть только где алгоритм балансировки данных выбирает поставить новую точку, или посмотреть начальное распределение классов, убрав синтетические образцы.

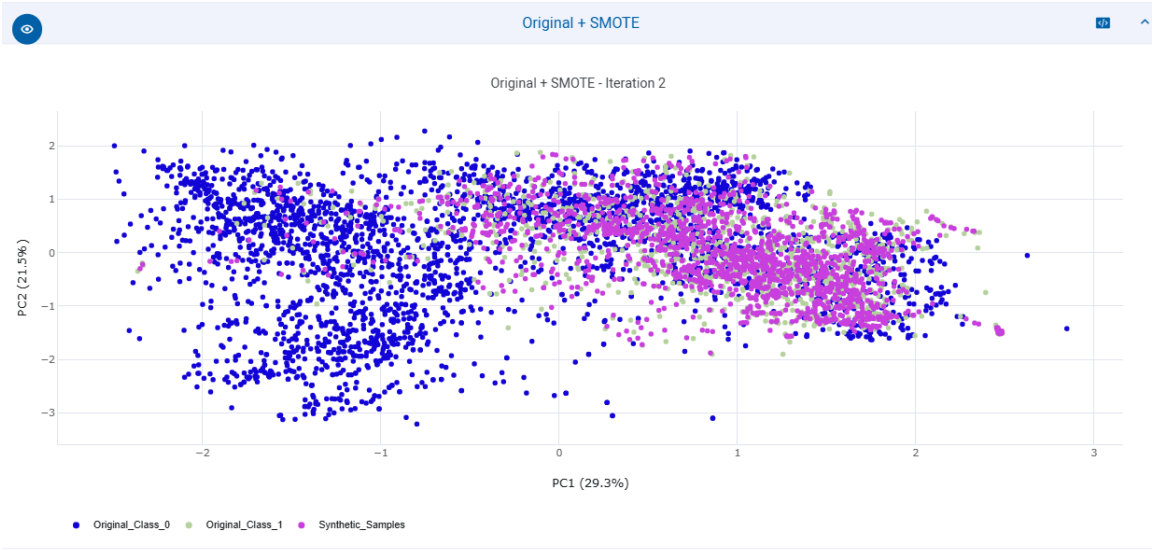

Рисунок 10 - Scatter plot, созданный с помощью PCE.

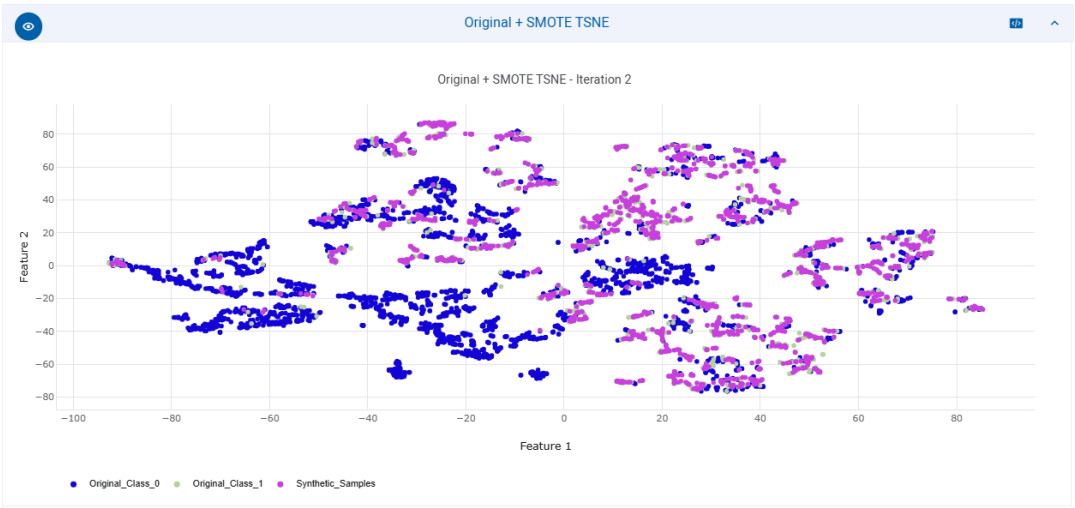

Рисунок 11 - Scatter plot, созданный с помощью TSNE.

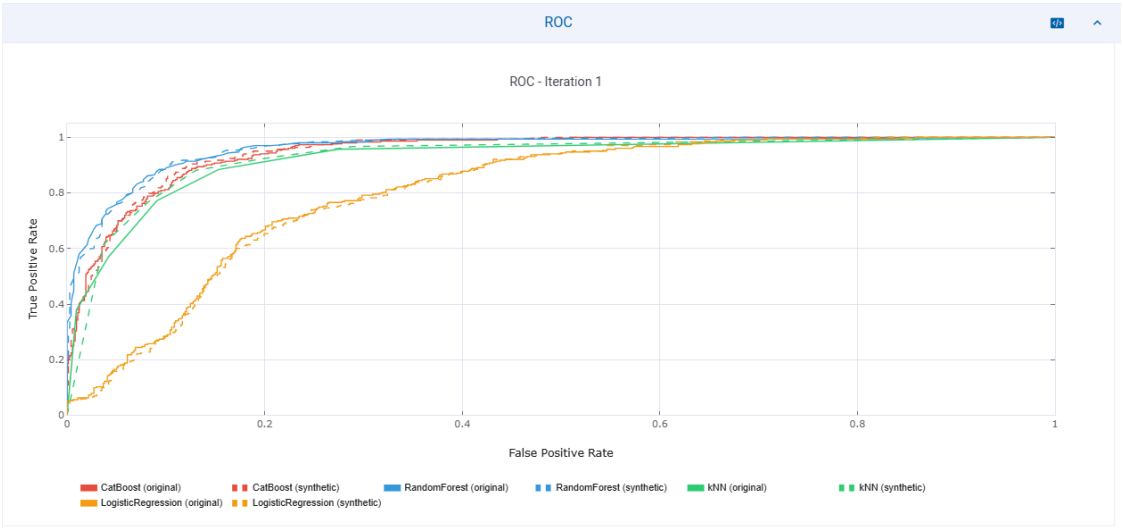

Рисунок 12 - ROC кривая эксперимента

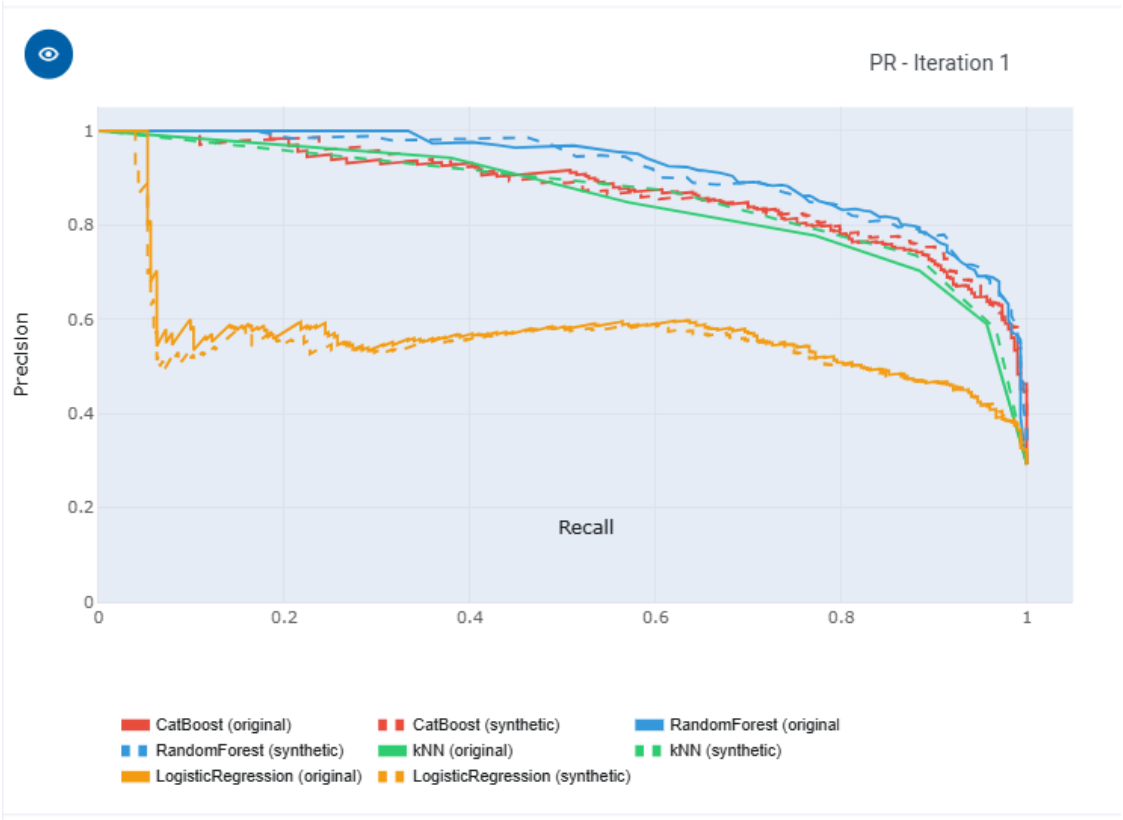

Рисунок 13 - Precision Recall кривая эксперимента.

src/base.py: реализован базовый класс для создания собственных алгоритмов балансировки данных. Класс вдохновлен библиотекой smote_variants и реализует слегка упрошенную структуру методов. На основе этого класса можно писать свои методы балансировки данных, которые можно интегрировать в библиотеку smote_variants.

classic/: Реализация собственных вариаций алгоритмов: SMOTE, Borderline_SMOTE , ADASYN [12].

evaluation/basic_evaluator.py: модуль всех вычисляемых метрик. Содержит в себе вычисление следующих метрик: метрики матрицы ошибок, precision, recall, specificity, F1 score, G mean, balanced accuracy, ROC AUC. А также вариации этих метрик с различными усреднениями: precision_macro, precision_weighted и тд. В конфигурационном файле эксперимента указываются метрики, которые необходимо наблюдать во время эксперимента, а модуль возвращает словарь со значениями

указанных метрик, которые затем в модуле эксперимента обрабатываются и логируются в ClearML удобным визуальным представлением.

5.2.3 Обработка данных

notebooks/datasets.ipynb: ноутбук, используемый для обработки наборов данных. Сырые данные подкачиваются из ClearML и обрабатываются по следующему сценарию:

1. Определение типов данных столбцов.
2. Сведение к бинарной классификации (Выбрано для упрощения и для возможности сравнивать различные методы, так как не все методы могут работать с мультиклассовостью)
3. Очистка данных: удаление дубликатов, обработка пропущенных значений: k-NN Imputer для численных, модальную для категориальных
4. Детекция выбросов Z-score
5. Кодирование категориальных переменных” one-hot для номинальных, label для порядковых
6. MiniMax scaling

Было обработано 13 наборов данных, с характеристиками в соответствии с таблицей 2: название, количество признаков, количество наблюдений в наборе данных, количество классов в изначальном наборе данных и дисбаланс классов, сведенных к бинарной классификации. Такой сценарий обработки был выбран для возможности оценивать различные методы на одинаковых наборах данных. Если необходимо точное воспроизведения экспериментов из статьи, то обработка может быть изменена и создана новая версия набора данных в среде ClearML.

5.2.4 Наборы данных:

1. *Adult [13]*: содержит данные переписи населения США 1994 года, используемые для классификации уровня дохода граждан (более или менее $50,000 в год). 14 признаков, 48842 записи, соотношение классов 3:1.

2. *Abalone [14]*: содержит физические измерения моллюсков абалон (Haliotis), используемые для предсказания возраста, 8 признаков, 4177 наблюдений, соотношение классов (класс 7 против остальных) 9.7:1.

3. *Forest coverage [15]*: содержит картографические данные участков леса размером 30×30 метров в Национальном лесу Рузвельта, северный Колорадо. Используется для классификации типов лесного покрова на основе картографических переменных без использования данных дистанционного зондирования, 54 признака, 581012 наблюдений, соотношение классов (Выбраны два класса из 7 с 35754 и 2747 наблюдениями соответственно) 13:1

4. *Haberman [16]:* содержит данные исследования выживаемости пациентов после операции по поводу рака молочной железы, проведенного в период с 1958 по 1970 год в больнице Биллингса Чикагского университета, 3 признака, 306 наблюдений, 2.8:1

5. *Ionosphere [17]*: содержит данные радарных сигналов. 34 признака, 351 наблюдений, 1.8:1

6. *Mammography [18]*: содержит данные о результатах анализа маммограмм для обнаружения кластеров микрокальцинатов - ранних признаков рака молочной железы. 6 признаков, 1183 наблюдений, 42:1

7. *Oil [19]:* содержит данные для классификации проб нефти на основе различных химических и физических характеристик., 49 признаков, 937 наблюдений, 3 класса, сведение к бинарной приводит к соотношению 22:1.

8. *Phoneme [20]*: содержит акустические признаки звуков речи для задачи различения назальных и ротовых звуков, 5 признаков, 5404 наблюдений, 2.4:1

9. *Pima diabetes [21]*: содержит медицинские данные индейцев племени Пима для предсказания наличия диабета, 8 признаков, 768 наблюдений, 1.9:1

10. *Satimage [22]*: содержит мультиспектральные значения пикселей в окрестностях 3×3 пикселя спутниковых изображений Landsat MSS. Каждый пиксель покрывает область около 80×80 метров, 36 признаков, 6435 наблюдений, рассматривается класс 4 с остальными, 9.3:1

11. *Ecoli [23]*: содержит данные для предсказания сайтов клеточной локализации белков E.coli на основе различных биохимических признаков, 7 признаков, 336 наблюдений, сведение к бинарной классификации: класс imU против остальных, 8.6:1

12. *US_crime [24]*: объединяет социально-экономические данные из переписи населения США 1990 года, данные правоохранительных органов из исследования LEMAS 1990 года и данные о преступности из отчетов ФБР UCR 1995 года. Используется для предсказания уровня насильственных преступлений в различных сообществах США., 100 призаков, 1994 наблюдений, 12:1

13. *Yeast_ml8 [25]*: содержит данные для предсказания клеточной локализации белков дрожжей, 103 признака, 2417 наблюдений, класс 8 против остальных, 13:1

Набор данных	Количество признаков	Количество наблюдений	Количество классов	Дисбаланс классов(бинарный)
Adult:	4	48842	2	3:1.
Abalone	8	4177	8	9.7:1
Forest coverage	54	581012	7	13:1
Haberman	3	306	2	2.8:1
Ionosphere	34	351	2	1.8:1
Mammography	6	1183	2	42:1
Oil	49	937	3	22:1
Phoneme	5	5404	2	2.4:1
Pima diabetes	8	768	2	1.9:1
Satimage	36	6435	4	9.3:1
Ecoli	7	336	7	8.6:1
US_crime	100	1994	2	12:1
Yeast_ml8	103	2417	8	13:1

Таблица 2 - Таблица характеристик наборов данных.

5.2.5 Модуль экспериментов

/experiments/experiment_runner.py *(Приложение):* главный модуль проведения экспериментов. Содержит в себе ClassifietPool - набор классификаторов, которые можно использовать в экспериментах; ExperimentConfig - класс для обработки и хранения параметров эксперимента и ExperimentRunner - главный класс, в котором реализуются сценарии эксперимента. Все раннее описанные модули используются для работы этого модуля: вызовы функций, загрузка конфигов и сохранение данных. В этом классе предусмотрены возможности: создание задачи в ClearML с необходимыми зависимостями, логированием конфига и добавления основных тэгов для навигации; Связь с модулем визуализации для вызова функций построения графиков распределений данных до и после применения метода, и для вызова функций для логирования графиков

ROC и PR кривых; Основная функция - *run_single_experiment*(): здесь реализована логика построения эксперимента, описанного в 4 главе.

Для возможности пакетного тестирования была создана функция *direct_experiment():* функция вызывается первой при начале эксперимента, собираются данные конфигов и в двух циклах: по методам балансировки и по наборам данных, собираются результаты каждого эксперимента, а затем данные агрегируются в таблицу результатов, по которой можно сравнивать методы.

6 Заключение

В результате выполнения курсовой работы разработан и реализован тестовый стенд для анализа алгоритмов балансировки данных, основанных на SMOTE. Выполнены все поставленные требования реализации тестового стенда, достигнуты следующие результаты:

Проведён анализ литературы по методам балансировки наборов данных и теоретическим основам SMOTE.

1. Реализована модульная архитектура, возможность воспроизводить эксперименты и настраивать конфигурационные файлы для повторения эксперимента.
2. Возможность масштабируемости: добавления новых алгоритмов, наборов данных и метрик.
3. Использование системы для версионирования наборов данных.
4. Реализованы базовый SMOTE и две модификации: Borderline-SMOTE и ADASYN
5. Вся настройка экспериментов происходит через конфигурационные файлы и не требует от пользователя менять .py код.
6. Интегрирована системы логирования экспериментов, возможностью сравнивать эксперименты, строятся таблицы, по которым можно анализировать работу алгоритмов и визуально наблюдать как ведут себя классификаторы на данных *(Рисунок 13)*.

Разработанный тестовый стенд может быть использован как для целей обучения и для практического исследования методов балансировки данных в реальных задачах машинного обучения.

7 Список используемых источников

[1] Chawla, N. V. SMOTE: Synthetic Minority Over-sampling Technique / N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer // Journal of Artificial Intelligence Research. – 2002. – Vol. 16. – P. 321–357.

[2] ClearML [Электронный ресурс]: платформа для MLOps и управления экспериментами. – Режим доступа: **https://clear.ml/** .

[3] Santos, M. S. Cross-Validation for Imbalanced Datasets: Avoiding Overoptimistic and Overfitting Approaches / M. S. Santos, J. P. Soares, P. H. Abreu, H. Araujo, J. Santos // IEEE Computational Intelligence Magazine. – 2018. – Vol. 13, no. 4. – P. 59–76.

[4] Pedregosa, F. Scikit-learn: Machine Learning in Python [Электронный ресурс] / F. Pedregosa, G. Varoquaux, A. Gramfort [et al.] // Journal of Machine Learning Research. – 2011. – Vol. 12. – P. 2825–2830. (StratifiedKFold implementation).

[5] Han, H. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning / H. Han, W.-Y. Wang, B.-H. Mao // Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science. – 2005. – Vol. 3644. – P. 878–887.

[6] De la Calleja, J. A distance-based over-sampling method for learning from imbalanced data sets / J. De la Calleja, O. Fuentes // Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2007). – Menlo Park: AAAI Press, 2007. – P. 634–635.

[7] Bunkhumpornpat, C. Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling Technique for Handling the Class Imbalanced Problem / C. Bunkhumpornpat, K. Sinapiromsaran, C. Lursinsap // Advances in Knowledge Discovery and Data Mining. PAKDD 2009. Lecture Notes in Computer Science. – 2009. – Vol. 5476. – P. 475–482.

[8] Kovács, G. smote-variants: A Python Implementation of 85 Minority Oversampling Techniques / G. Kovács // Neurocomputing. – 2019. – Vol. 366. – P. 352–354.

[9] Ben-Kiki, O. YAML Ain't Markup Language (YAML™) Version 1.2 [Электронный ресурс] / O. Ben-Kiki, C. Evans, I. d'Högt. – 2009. – Режим доступа: **https://yaml.org/spec/1.2/spec.html**

[10] Ghanem, R. G. Stochastic Finite Elements: A Spectral Approach / R. G. Ghanem, P. D. Spanos. – New York: Springer-Verlag, 1991. – 214 p. (Polynomial Chaos Expansion - PCE).

[11] Van der Maaten, L. Visualizing Data using t-SNE / L. Van der Maaten, G. Hinton // Journal of Machine Learning Research. – 2008. – Vol. 9. – P. 2579–2605.

[12] He, H. ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning / H. He, Y. Bai, E. A. Garcia, S. Li // Proceedings of the IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence). – Hong Kong: IEEE, 2008. – P. 1322–1328.

[13] Kohavi, R. Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid (Adult Dataset) / R. Kohavi // Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. – AAAI Press, 1996. – P. 202–207.

[14] Nash, W. J. The Population Biology of Abalone (Haliotis species) in Tasmania. I. Blacklip Abalone (H. rubra) from the North Coast and Islands of Bass Strait (Abalone Dataset) / W. J. Nash, T. L. Sellers, S. R. Talbot [et al.]. – Tasmania: Sea Fisheries Division, Technical Report No. 48, 1994.

[15] Blackard, J. A. Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis for Predicting Forest Cover Types from Cartographic Variables / J. A. Blackard, D. J. Dean // Computers and Electronics in Agriculture. – 1999. – Vol. 24, no. 3. – P. 131–151.

[16] Haberman, S. J. Generalized Residuals for Log-Linear Models (Haberman's Survival Data) / S. J. Haberman // Proceedings of the 9th International Biometrics Conference. – Boston, 1976. – P. 104–122.

[17] Sigillito, V. G. Classification of Radar Returns from the Ionosphere using Neural Networks / V. G. Sigillito, S. P. Wing, L. V.

Hutton, K. B. Baker // Johns Hopkins APL Technical Digest. – 1989. – Vol. 10, no. 3. – P. 262–266.

[18] Elter, M. The Prediction of Breast Cancer Biopsy Outcomes Using Two CAD Approaches that Both Emphasize an Intelligible Decision Process (Mammography Dataset) / M. Elter, R. Schulz-Wendtland, T. Wittenberg // Medical Physics. – 2007. – Vol. 34, no. 11. – P. 4164–4172.

[19] Kubat, M. Machine Learning for the Detection of Oil Spills in Satellite Radar Images / M. Kubat, R. C. Holte, S. Matwin // Machine Learning. – 1998. – Vol. 30. – P. 195–215.

[20] ELENA Project [Электронный ресурс]: Phoneme Dataset. – Режим доступа: **https://www.openml.org/d/1489**

[21] Smith, J. W. Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus / J. W. Smith, J. E. Everhart, W. C. Dickson [et al.] // Proceedings of the Symposium on Computer Applications in Medical Care. – IEEE Computer Society Press, 1988. – P. 261–265.

[22] King, R. D. Statlog: Comparison of Classification Algorithms on Large Real-World Problems (Satimage Dataset) / R. D. King, C. Feng, A. Sutherland. – Applied Artificial Intelligence. – 1995. – Vol. 9, no. 3. – P. 289–333.

[23] Horton, P. A Probabilistic Method for Predicting the Subcellular Localization of Proteins (Ecoli Dataset) / P. Horton, K. Nakai // Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology. – AAAI Press, 1996. – P. 109–115.

[24] Redmond, M. A Data-Driven Software Tool for Enabling Cooperative Crime Analysis (Communities and Crime Dataset) / M. Redmond, A. Baveja // Proceedings of the IEEE International Conference on Intelligence and Security Informatics. – Dallas: IEEE, 2002. – P. 1–12.

[25] Lemaître, G. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning (Yeast_ml8 Dataset usage) / G. Lemaître, F. Nogueira, C. K. Aridas // Journal of Machine Learning Research. – 2017. – Vol. 18, no. 17. – P. 1–5.

8 Приложение

Experiment_runner.py:

```
import logging
import time
import numpy as np
from typing import Any, Dict, Optional, List
from sklearn.model_selection import StratifiedKFold, train_test_split
import warnings
import json
import pandas as pd
import smote_variants as sv
from configs.config_loader import ConfigLoader

warnings.filterwarnings('ignore')
from clearml import Task
from src.utils.data_loader import fetch_dataset
from src.utils.preprocessing import SMOTEPreprocessor, PreprocessingConfig
from src.evaluation.basic_evaluator import all_smote_metrics
from src.utils.visualise import Visualiser


class ClassifierPool:
    def __init__(self, random_state: int = 42):
        self.random_state = random_state

    def get_classifiers(self) -> Dict[str, Any]:
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.svm import SVC
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.naive_bayes import GaussianNB
        from sklearn.tree import DecisionTreeClassifier
        from catboost import CatBoostClassifier
        return {
            'CatBoost': CatBoostClassifier(
                random_state=self.random_state,
                verbose=False,
            ),
            'RandomForest': RandomForestClassifier(
                n_estimators=100,
                random_state=self.random_state
            ),
            'SVM': SVC(
                kernel='rbf',
                probability=True,
                random_state=self.random_state
            ),
            'kNN': KNeighborsClassifier(n_neighbors=5),
            'LogisticRegression': LogisticRegression(
```

```
            random_state=self.random_state,
            max_iter=1000
        ),
        'DecisionTree': DecisionTreeClassifier(
            random_state=self.random_state
        ),
        'NaiveBayes': GaussianNB()
    }


class ExperimentConfig:
    def __init__(self, cfg=None):
        if cfg is None:
            cfg = {}

        self.cv_folds = cfg.get('cv_folds', 5)
        self.random_runs = cfg.get('random_runs', 3)
        self.test_size = cfg.get('test_size', 0.2)
        self.random_state = cfg.get('random_state', 42)

        self.priority_metrics = cfg.get('priority_metrics', [
            'balanced_accuracy', 'f1_weighted', 'g_mean',
            'roc_auc_weighted', 'precision_weighted', 'recall_weighted'
        ])

        self.selected_classifiers = cfg.get('selected_classifiers', [
            'RandomForest', 'SVM', 'kNN', 'LogisticRegression'
        ])

        self.clearml_project_name = "SMOTE Test Bench"
        self.clearml_task_name = None
        self.clearml_tags = None
        self.auto_log_artifacts = True
        self.log_model_params = True

        self.enable_scatter_plots = True
        self.enable_roc_curves = True
        self.enable_precision_recall_curves = True

    def get_config(self) -> Dict:
        config = {
            'cv_folds': self.cv_folds,
            'random_runs': self.random_runs,

            'test_size': self.test_size,
            'random_state': self.random_state,

            'priority_metrics': self.priority_metrics,
            'selected_classifiers': self.selected_classifiers,

            'clearml_project_name': self.clearml_project_name,
            'clearml_task_name': self.clearml_task_name,
```

```
            'clearml_tags': self.clearml_tags,
            'auto_log_artifacts': self.auto_log_artifacts,
            'log_model_params': self.log_model_params,

            'enable_scatter_plots': self.enable_scatter_plots,
            'enable_roc_curves': self.enable_roc_curves,
            'enable_precision_recall_curves': self.enable_precision_recall_curves
        }

        return config


class ExperimentRunner:
    def __init__(self,
                 config: Optional[ExperimentConfig] = None,
                 create_clearml_task: bool = True,
                 clearml_task: Optional[Task] = None
                 ):

        self.config = config or ExperimentConfig()
        self.create_clearml_task = create_clearml_task
        self.task = None

        if create_clearml_task and clearml_task is None:
            self._initialize_clearml_task()
            if self.task:
                self.logger = self.task.get_logger()
        else:
            self.task = clearml_task
            if self.task:
                self.logger = self.task.get_logger()

        self.visualiser = Visualiser()
        if self.task:
            self.visualiser.set_clearml_task(self.task)

        self.classifier_pool = ClassifierPool(random_state=self.config.random_state)
        self.results = {}
        self.experiment_metadata = {}
        self.visualisation_counter = 0

    def _initialize_clearml_task(self):
        task_name = self.config.clearml_task_name or f"SMOTE Experiment
{time.strftime('%Y%m%d_%H%M%S')}"

        Task.add_requirements('requirements.txt')
        self.task = Task.init(
            project_name=self.config.clearml_project_name,
            task_name=task_name,
            tags=self.config.clearml_tags
        )
```

```
    config_dict = self.config.get_config()
    self.task.connect(config_dict, name='experiment_config')

def _create_data_scatter_visualisation(self, X_train: np.ndarray, y_train: np.ndarray,
                                       X_train_smote: np.ndarray, y_train_smote: np.ndarray,
                                       synthetic_samples: np.ndarray
                                       ):
    if self.config.enable_scatter_plots and X_train.shape[1] >= 2:
        feature_names = [f'Feature {i + 1}' for i in range(X_train.shape[1])]

        X_train_np = X_train.values if hasattr(X_train, 'values') else X_train
        y_train_np = y_train.values if hasattr(y_train, 'values') else y_train

        self.visualiser.plot_data_scatter(
            X_original=X_train_np,
            y_original=y_train_np,
            X_smote=X_train_smote,
            y_smote=y_train_smote,
            synthetic_samples=synthetic_samples,
            feature_names=feature_names,
            log_to_clearml=True,
            iteration=2
        )


        self.visualiser.plot_data_scatter_tsne(
            X_original=X_train_np,
            y_original=y_train_np,
            X_smote=X_train_smote,
            y_smote=y_train_smote,
            synthetic_samples=synthetic_samples,
            feature_names=feature_names,
            log_to_clearml=True,
            iteration=2
        )
def _prepare_predictions_data(self, final_results: Dict) -> Dict:
    roc_predictions = {}

    for clf_name, clf_results in final_results.items():
        if 'original_data' in clf_results and 'smote_data' in clf_results:
            roc_predictions[clf_name] = {}
            roc_predictions[clf_name]['original'] = clf_results['original_data']['y_pred_proba']
            roc_predictions[clf_name]['smote'] = clf_results['smote_data']['y_pred_proba']

    roc_predictions = {k: v for k, v in roc_predictions.items() if v}

    return {'roc_predictions': roc_predictions}

def _create_results_visualisations(self, final_results: Dict,
                                   y_test: np.ndarray,
                                   dataset_name: str,
                                   smote_algorithm: Any
```

```
                            ):
    predictions_data = self._prepare_predictions_data(final_results)

    # ROC кривые
    if predictions_data['roc_predictions'] and self.config.enable_roc_curves:
        self.visualiser.plot_roc_curves(
            y_test=y_test,
            predictions=predictions_data['roc_predictions'],
            title=f"ROC",
            clearml_task=self.task,
            method_name=smote_algorithm.__class__.__name__,
            iteration=3
        )

    # Precision-Recall кривые
    if predictions_data['roc_predictions'] and self.config.enable_precision_recall_curves:
        self.visualiser.plot_precision_recall_curves(
            y_test=y_test,
            predictions=predictions_data['roc_predictions'],
            title=f"PR",
            clearml_task=self.task,
            method_name=smote_algorithm.__class__.__name__,
            iteration=3
        )

def _log_dataset_info(self, X, y):
    if not self.task:
        return

    logger = self.task.get_logger()
    class_dist = np.bincount(y)
    imbalance_ratio = max(class_dist) / min(class_dist) if min(class_dist) > 0 else float('inf')

    logger.report_scalar("Dataset Info", "Total Samples", len(X), iteration=0)
    logger.report_scalar("Dataset Info", "Features", X.shape[1], iteration=0)
    logger.report_scalar("Dataset Info", "Classes", len(class_dist), iteration=0)
    logger.report_scalar("Dataset Info", "Imbalance Ratio", imbalance_ratio, iteration=0)

def _cross_validation_with_smote(self,
                                 X_train: np.ndarray,
                                 y_train: np.ndarray,
                                 smote_algorithm: Any,
                                 classifiers: Dict[str, Any]
                                 ) -> Dict[str, Any]:
    logging.getLogger('smote_variants').setLevel(logging.WARNING)

    cv = StratifiedKFold(
        n_splits=self.config.cv_folds,
        shuffle=True,
        random_state=self.config.random_state
    )
```

```
cv_results = {}
current_iteration = 0
all_fold_results = []

for clf_name, classifier in classifiers.items():
    cv_scores = {metric: [] for metric in self.config.priority_metrics}

    for fold, (train_idx, val_idx) in enumerate(cv.split(X_train, y_train)):
        current_iteration += 1

        X_fold_train, X_fold_val = X_train.iloc[train_idx].values, X_train.iloc[val_idx].values
        y_fold_train, y_fold_val = y_train.iloc[train_idx].values, y_train.iloc[val_idx].values

        X_fold_train_smote, y_fold_train_smote = smote_algorithm.fit_resample(
            X_fold_train, y_fold_train
        )

        if fold == 0 and self.task:
            original_size = len(y_fold_train)
            smote_size = len(y_fold_train_smote)
            logger = self.task.get_logger()

        classifier.fit(X_fold_train_smote, y_fold_train_smote)
        y_pred = classifier.predict(X_fold_val)

        y_pred_proba = None
        if hasattr(classifier, 'predict_proba'):
            y_pred_proba = classifier.predict_proba(X_fold_val)[:, 1]

        fold_metrics = all_smote_metrics(
            y_fold_val, y_pred, y_pred_proba
        )

        for metric in self.config.priority_metrics:
            if metric in fold_metrics:
                cv_scores[metric].append(fold_metrics[metric])

        fold_result = {
            'Classifier': clf_name,
            'Fold': fold + 1,
            'Train_Samples': len(y_fold_train_smote),
            'Val_Samples': len(y_fold_val)
        }
        fold_result.update({
            metric: fold_metrics.get(metric, 0)
            for metric in self.config.priority_metrics
        })
        all_fold_results.append(fold_result)

        if self.task:
            logger = self.task.get_logger()
            for metric in self.config.priority_metrics:
```

```
                if metric in fold_metrics:
                    logger.report_scalar(
                        f"CV Fold Results - {clf_name}",
                        metric,
                        fold_metrics[metric],
                        iteration=fold + 1
                    )

        cv_results[clf_name] = {}
        for metric in self.config.priority_metrics:
            if cv_scores[metric]:
                mean_score = np.mean(cv_scores[metric])
                std_score = np.std(cv_scores[metric])
                cv_results[clf_name][f'{metric}_mean'] = mean_score
                cv_results[clf_name][f'{metric}_std'] = std_score

                if self.task:
                    logger = self.task.get_logger()
                    logger.report_scalar(
                        f"CV Summary - {metric}",
                        f"{clf_name}_mean",
                        mean_score,
                        iteration=1
                    )
                    logger.report_scalar(
                        f"CV Summary - {metric}",
                        f"{clf_name}_std",
                        std_score,
                        iteration=1
                    )

    return cv_results

def _final_evaluation(self,
                      X_train: np.ndarray, y_train: np.ndarray,
                      X_test: np.ndarray, y_test: np.ndarray,
                      smote_algorithm: Any,
                      classifiers: Dict[str, Any],
                      dataset_name: str
                      ) -> Dict[str, Any]:

    logging.getLogger('smote_variants').setLevel(logging.WARNING)
    X_train_smote, y_train_smote = smote_algorithm.fit_resample(X_train.values, y_train.values)

    final_results = {}

    for clf_name, classifier in classifiers.items():
        classifier_original = type(classifier)(**classifier.get_params())
        classifier_original.fit(X_train, y_train)
        y_pred_original = classifier_original.predict(X_test)

        classifier_smote = type(classifier)(**classifier.get_params())
```

```
classifier_smote.fit(X_train_smote, y_train_smote)
y_pred_smote = classifier_smote.predict(X_test)

y_pred_proba_original = None
y_pred_proba_smote = None

if hasattr(classifier_original, 'predict_proba') and hasattr(classifier_smote, 'predict_proba'):
    y_pred_proba_original = classifier_original.predict_proba(X_test)[:, 1]
    y_pred_proba_smote = classifier_smote.predict_proba(X_test)[:, 1]

metrics_original = all_smote_metrics(
    y_test, y_pred_original, y_pred_proba_original
)

metrics_smote = all_smote_metrics(
    y_test, y_pred_smote, y_pred_proba_smote
)

improvements = {
    metric: metrics_smote[metric] - metrics_original[metric]
    for metric in metrics_original.keys()
    if metric in metrics_smote
}

final_results[clf_name] = {
    'original_data': {
        **metrics_original,
        'y_pred': y_pred_original,
        'y_pred_proba': y_pred_proba_original,
    },
    'smote_data': {
        **metrics_smote,
        'y_pred': y_pred_smote,
        'y_pred_proba': y_pred_proba_smote,
    },
    'improvement': improvements,
}

if self.task:
    logger = self.task.get_logger()
    for metric in self.config.priority_metrics:
        if metric in metrics_original:
            logger.report_scalar(
                f"Final Test - Original",
                f"{clf_name}_{metric}",
                metrics_original[metric],
                iteration=1
            )

        if metric in metrics_smote:
            logger.report_scalar(
                f"Final Test - SMOTE",
```

```
                    f"{clf_name}_{metric}",
                    metrics_smote[metric],
                    iteration=1
                )

                logger.report_scalar(
                    f"Final Test - Improvement",
                    f"{clf_name}_{metric}",
                    improvements[metric],
                    iteration=1
                )

    n_original = len(X_train)
    synthetic_samples = X_train_smote[n_original:] if len(X_train_smote) > n_original else None

    self._create_data_scatter_visualisation(
        X_train, y_train, X_train_smote, y_train_smote, synthetic_samples
    )

    return final_results

def _create_metrics_summary_table(self, final_result: Dict[str, Any],
                                  dataset_name: str,
                                  smote_algorithm: Any,
                                  iteration: int = 1
                                  ) -> pd.DataFrame:
    table_data = []

    for clf_name, clf_results in final_result.items():
        original_data = clf_results.get('original_data', {})
        smote_data = clf_results.get('smote_data', {})
        improvements = clf_results.get('improvement', {})

        for metric in self.config.priority_metrics:
            if metric in original_data and metric in smote_data:
                orig_val = original_data[metric]
                smote_val = smote_data[metric]
                improv = improvements.get(metric, 0)
                improv_pct = (improv / orig_val * 100) if orig_val != 0 else 0

                table_data.append({
                    'Classifier': clf_name,
                    'Metric': metric,
                    'Original': round(orig_val, 4),
                    f'{smote_algorithm.__class__.__name__}': round(smote_val, 4),
                    'Delta_Absolute': round(improv, 4),
                    'Delta_Percent': round(improv_pct, 2)
                })

    df = pd.DataFrame(table_data)

    if self.task:
```

```
            logger = self.task.get_logger()
            logger.report_table(
                title=f"Metrics Summary - {smote_algorithm.__class__.__name__}",
                series=dataset_name,
                iteration=iteration,
                table_plot=df
            )

        return df

    def _save_experiment_artifacts(self,
                                   experiment_results,
                                   dataset_name,
                                   smote_algorithm
                                   ):
        results_filename =
f"results/experiment_results_{dataset_name}_{smote_algorithm.__class__.__name__}.json"

        with open(results_filename, 'w', encoding='utf-8') as f:
            json.dump(experiment_results, f, indent=2, ensure_ascii=False, default=str)

        self.task.upload_artifact('experiment_results', results_filename)

        self._save_results_csv(experiment_results, dataset_name, smote_algorithm)

    def _save_results_csv(self,
                          experiment_results,
                          dataset_name,
                          smote_algorithm
                          ):

        final_results = experiment_results.get('final_test_results', {})
        csv_data = []

        for clf_name, clf_results in final_results.items():
            original_data = clf_results.get('original_data', {})
            smote_data = clf_results.get('smote_data', {})
            improvements = clf_results.get('improvement', {})

            for metric in self.config.priority_metrics:
                if metric in original_data and metric in smote_data:
                    csv_data.append({
                        'Dataset': dataset_name,
                        'SMOTE_Algorithm': smote_algorithm.__class__.__name__,
                        'Classifier': clf_name,
                        'Metric': metric,
                        'Original_Score': original_data[metric],
                        'SMOTE_Score': smote_data[metric],
                        'Improvement': improvements.get(metric, 0),
                        'Improvement_Percentage': (
                              improvements.get(metric, 0) / original_data[metric] * 100
                        ) if original_data[metric] != 0 else 0
```

```
                })

        if csv_data:
            csv_df = pd.DataFrame(csv_data)
            csv_filename =
f"results/results_summary_{dataset_name}_{smote_algorithm.__class__.__name__}.csv"
            csv_df.to_csv(csv_filename, index=False, encoding='utf-8')

            self.task.upload_artifact('results_summary_csv', csv_filename)

    def close_task(self):
        if self.task:
            self.task.close()

    def run_single_experiment(self,
                              dataset_name: str,
                              smote_algorithm: Any,
                              dataset_params: Optional[Dict] = None,
                              method_params: Optional[Dict] = None,
                              parent_task_id: Optional[str] = None,
                              experiment_config: Optional[Dict] = None
                              ) -> Dict[str, Any]:

        experiment_name = f"{dataset_name} + {smote_algorithm.__class__.__name__}"

        self._initialize_clearml_task()
        self.visualiser.set_clearml_task(self.task)

        if self.task and not self.config.clearml_task_name:
            self.task.set_name(f"{experiment_name}")

        if self.task and parent_task_id:
            self.task.set_parent(parent_task_id)

        dataset_params = dataset_params or {}
        method_params = method_params or {}

        if self.task:
            experiment_params = {
                'dataset_name': dataset_name,
                'smote_algorithm': smote_algorithm.__class__.__name__,
                'experiment_config': experiment_config,
                'dataset_params': dataset_params,
                'method_params': method_params
            }
            self.task.connect(experiment_params, name='current_experiment_params')
            self.task.add_tags([dataset_name, smote_algorithm.__class__.__name__])

        preprocess = dataset_params.get('preprocessed', False)

        df, metadata = fetch_dataset(dataset_name, preprocess)
        target = df.columns.tolist()[-1]
```

```
X = df.drop([target], axis=1)
y = df.iloc[:, -1]

if self.task:
    self._log_dataset_info(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=self.config.test_size,
    stratify=y,
    random_state=self.config.random_state
)

classifiers = self.classifier_pool.get_classifiers()
selected_classifiers = {
    name: clf for name, clf in classifiers.items()
    if name in self.config.selected_classifiers
}

cv_imbalanced_results = self._cross_validation_with_smote(
    X_train, y_train, sv.NoSMOTE(), selected_classifiers
)

cv_results = self._cross_validation_with_smote(
    X_train, y_train, smote_algorithm, selected_classifiers
)

final_results = self._final_evaluation(
    X_train, y_train, X_test, y_test,
    smote_algorithm, selected_classifiers, dataset_name=dataset_name
)

self._create_metrics_summary_table(final_results, dataset_name, smote_algorithm)
self._create_results_visualisations(final_results, y_test, dataset_name, smote_algorithm)

experiment_results = {
    'metadata': {
        'dataset_name': dataset_name,
        'algorithm_name': smote_algorithm.__class__.__name__,
        'dataset_params': dataset_params,
        'timestamp': time.strftime('%Y-%m-%d %H:%M:%S'),
        'clearml_task_id': self.task.id if self.task else None
    },
    'dataset_info': {
        'total_samples': len(X),
        'features': X.shape[1],
        'train_samples': len(X_train),
        'test_samples': len(X_test),
        'original_class_distribution': np.bincount(y).tolist(),
        'train_class_distribution': np.bincount(y_train).tolist()
    },
    'cross_validation_imbalanced_results': cv_imbalanced_results,
```

```
            'cross_validation_results': cv_results,
            'final_test_results': final_results,
        }

        if self.task and self.config.auto_log_artifacts:
            self._save_experiment_artifacts(experiment_results, dataset_name, smote_algorithm)

        self.close_task()

        return experiment_results

    def get_row(self, results, cv, classifier, algorithm_name, priority_metrics):
        clf_rename = {
            'CatBoost': 'CB',
            'RandomForest': 'RF',
            'SVM': 'SVM',
            'kNN': 'kNN',
            'LogisticRegression': 'LR',
            'DecisionTree': 'DT',
            'NaiveBayes': 'NB'
        }
        row = [algorithm_name, clf_rename[classifier]]
        cv_results = results[cv][classifier]
        for metric in priority_metrics:
            str = f'{cv_results[f"{metric}_mean"]:.4f}'
            row.append(str)
        return row

    def make_tables(self, datasets, methods, results, experiment_config):
        logger = self.task.get_logger()
        priority_metrics = list(experiment_config['priority_metrics'])
        selected_classifiers = experiment_config['selected_classifiers']
        columns = ['Method', 'Classificator'] + priority_metrics
        for dataset in datasets:
            rows = []
            dataset_count = 0
            for result in results:
                if dataset == result['metadata']['dataset_name']:
                    dataset_count += 1
                    if dataset_count == 1:
                        for classifier in selected_classifiers:
                            rows.append(self.get_row(result, 'cross_validation_imbalanced_results', classifier,
'Original', priority_metrics))
                    algorithm_name = result['metadata']['algorithm_name']
                    for classifier in selected_classifiers:
                        rows.append(self.get_row(result, 'cross_validation_results', classifier,
algorithm_name, priority_metrics))
            df = pd.DataFrame(rows, columns=columns)
            logger.report_table(
                title=f'{dataset} results',
                series='Table',
                iteration=0,
```

```
            table_plot=df
        )

def aggregate_results(self, results, experiment_config):
    experiment_name = "results"

    self._initialize_clearml_task()
    self.visualiser.set_clearml_task(self.task)

    if self.task and not self.config.clearml_task_name:
        self.task.set_name(f"{experiment_name}")

    datasets = []
    methods = []

    for result in results:
        datasets.append(result['metadata']['dataset_name'])
        methods.append(result['metadata']['algorithm_name'])

    datasets = datasets
    methods = methods

    if self.task:
        experiment_params = {
            'datasets': datasets,
            'methods': methods
        }
        self.task.connect(experiment_params, name='Data')
        self.task.add_tags(['Results', 'Figures'])

    self.make_tables(datasets, methods, results, experiment_config)

    self.close_task()

def direct_experiments(self, config_name: str):

    logging.getLogger(sv.__name__).setLevel(logging.WARNING)

    loader = ConfigLoader(config_name)
    cfg = loader.load()

    experiment_config = cfg['experiment_config']

    datasets_name = list(cfg['datasets'])
    datasets_params = cfg['datasets_params']

    oversamplers_names = list(cfg['methods'])

    oversamplers_config_name = 'methods.yaml'
    oversamplers_loader = ConfigLoader(oversamplers_config_name)
    oversamplers_config = oversamplers_loader.load()
```

```
general_results = []

for oversampler_name in oversamplers_names:
    oversampler = eval(oversamplers_config[oversampler_name]['method'])
    for dataset_name in datasets_name:
        results = self.run_single_experiment(dataset_name, oversampler, datasets_params,
                                             experiment_config=experiment_config)
        general_results.append(results)

self.aggregate_results(general_results, experiment_config)
```