МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка веб-приложения для бинарной классификации обфусцированных команд PowerShell с использованием алгоритмов машинного обучения

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-315.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ К.Ю. Никольская

Автор работы,
студент группы КЭ-401
_____ М.И. Скоблюк

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра

студенту группы КЭ-401
Скоблюку Максиму Игоревичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка веб-приложения для бинарной классификации обфусцированных команд PowerShell с использованием алгоритмов машинного обучения.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. Detecting Obfuscated Scripts With Machine Learning Techniques. [Электронный ресурс] URL: https://core.ac.uk/download/pdf/288496043 (дата обращения: 12.01.2024 г.).
3.2. Введение в Data Science и машинное обучение. [Электронный ресурс] URL: https://stepik.org/course/4852/syllabus (дата обращения: 10.01.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Провести обзор научной литературы.
4.2. Подготовить обучающий набор данных.
4.3. Реализовать выбранные методы машинного обучения.

4.4. Разработать веб-приложение для бинарной классификации обфусцированных команд PowerShell.

4.5. Провести тестирование разработанного веб-приложения.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП К.Ю. Никольская

Задание принял к исполнению М.И. Скоблюк

ГЛОССАРИЙ

1. *Машинное обучение* – это подмножество искусственного интеллекта, которое изучает алгоритмические методы для анализа и обработки данных, позволяющие компьютеру обучаться и делать прогнозы на основе предоставленных ему данных [1].

2. *Программное обеспечение (ПО)* – это совокупность программ, процессов и данных, используемых для управления работой вычислительной системы [2].

3. *PowerShell* – это инструмент автоматизации администрирования, который позволяет пользователям управлять и обрабатывать процессы на компьютере или в сети, используя командную строку и скрипты [3].

4. *Обфускация* – это техника, которая заключается в изменении исходного кода программы, чтобы сделать его трудным для чтения и понимания [4].

5. *Обфусцированная команда PowerShell* – это команда, которая была изменена таким образом, чтобы обмануть систему антивирусной защиты или другие методы обнаружения вредоносных программ [5].

6. *Вредоносная команда PowerShell* – это любая команда или скрипт, который создан для нанесения вреда компьютеру или сети, такой как кража информации, нарушение нормальной работы или установка вредоносного ПО [6].

7. *Токенизация* – это процесс разделения текста на отдельные элементы, как правило, слова или фразы, которые затем могут быть анализированы отдельно [1].

8. *Abstract Syntax Tree (AST)* – это внутреннее представление программы в виде древовидной структуры, которая облегчает анализ и манипулирование программным кодом компилятором или интерпретатором [7].

9. *Unified Modeling Language (UML)* – это графический язык для краткого и точного определения требований к системе и визуализации и документирования поведения системы через методы диаграммирования [8].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном мире в связи с ростом объемов информационных технологий в то же время значительно увеличивается число вредоносного ПО. С каждым днем угрозы эволюционируют, и это требует эволюции средств защиты от них.

После того, как в 2016 году PowerShell 6.0 стал открытым программным обеспечением, значительно увеличилось число вредоносного ПО, использующего возможности PowerShell. Согласно отчету McAfee Labs, число вредоносного ПО, основанного на применении PowerShell, с 2016 по 2017 год выросло на 432% [9]. Компания Symantec упоминает рост числа компьютеров, где была заблокирована работа PowerShell, в период с середины 2017 года по 2018 год на 661% [10].

Статистика роста вредоносного ПО PowerShell за последние 2 квартала 2019 года и 2 первых квартала 2020 года согласно отчету McAfee представлена на рисунке 1.

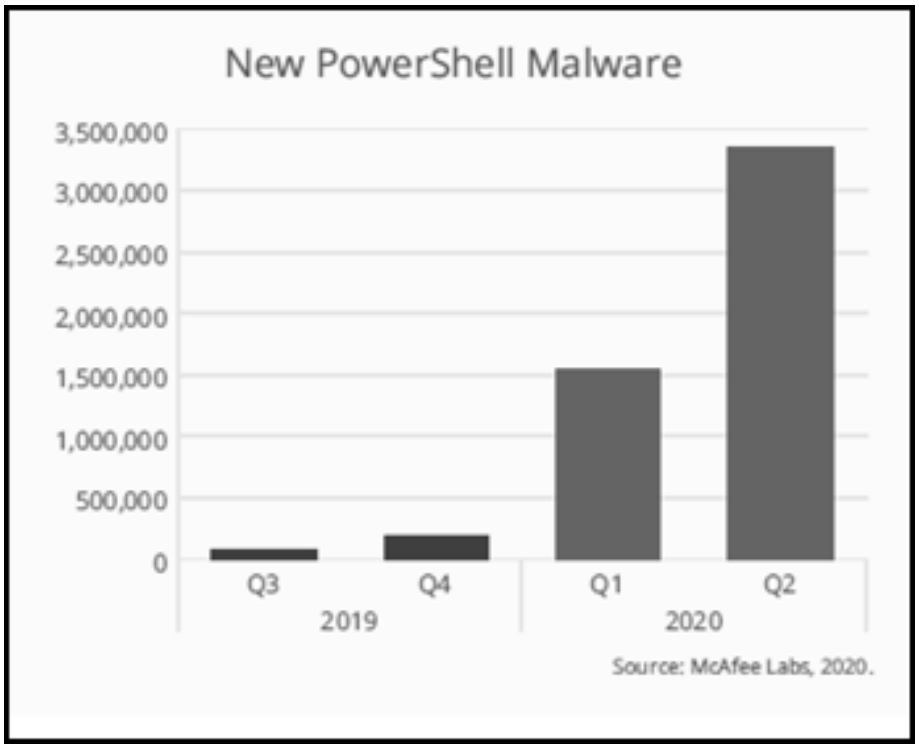


Рисунок 1 – Статистика роста вредоносного ПО PowerShell

В последнем квартале 2020 года количество угроз, использующих PowerShell, выросло на 208% согласно отчету компании McAfee [11].

Кроме того, использование PowerShell и обфускации относится к трем самым популярным техникам кибератак [12]. Известные команды киберпреступников используют обфускацию PowerShell для достижения поставленных вредоносных целей [13]. А также известные вредоносные программы прибегают к использованию обфускации [14].

Обфускация является отличным инструментом для злоумышленников в случае использования вредоносных PowerShell команд и скриптов. Методы обфускации позволяют скрыть нежелательное содержимое, ухудшить читаемость и замедлить время обнаружения вреда не только для человека, но и для программ, предназначенных для защиты от вредоносных программ [15].

Для этих целей злоумышленники применяют ряд техник обфускации, позволяющих значительно изменить внешний вид команды или скрипта PowerShell [15].

Постановка задачи

Целью выпускной квалификационной работы является разработка веб-приложения для бинарной классификации обфусцированных команд PowerShell с использованием алгоритмов машинного обучения. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор научной литературы;
2) подготовить обучающий набор данных;
3) реализовать выбранные методы машинного обучения;
4) разработать веб-приложение для бинарной классификации обфусцированных команд PowerShell;
5) провести тестирование разработанного приложения.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения, списка литературы и приложений. Объем работы составляет 101 страницу, объем списка литературы – 56 источников.

В первой главе описывается теоретическая часть, в которую входит следующее.

1. Обзор научной литературы – описание набора готовых решений для бинарной и многоклассовой классификации обфусцированных и вредоносных команд PowerShell, основанных на машинном обучении и нейронных сетях.
2. Описание задачи бинарной классификации обфусцированных команд PowerShell.
3. Теоретическая информация об используемых методах машинного обучения.

Вторая глава посвящена проектированию системы: описан процесс разметки набора данных, представлены функциональные и нефункциональные требования, UML диаграмма вариантов использования, схема архитектуры приложения, схема базы данных и макеты приложения.

В третьей главе описывается реализация приложения: представлено описание программных средств разработки, этапов предобработки набора данных, обучения алгоритмов машинного обучения, реализации серверной и клиентской части приложения и развертывания приложения.

В четвертой главе описано A/B тестирование алгоритмов машинного обучения, функциональное тестирование веб-приложения и тестирование Docker контейнеров.

В приложениях содержатся листинги кода обучения алгоритмов, описание роутов серверной части приложения и функциональное тестирование приложения.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Обзор научной литературы

PowerDP: De-Obfuscating and Profiling Malicious PowerShell Commands With Multi-Label Classifiers [16].

В данной работе реализована многоклассовая классификация вредоносных обфусцированных команд PowerShell.

Многоклассовость команд по степени обфусцированности заключается в следующем: имеется определенное количество техник обфускации, необходимо определить, какая комбинация этих техник применена к команде. Для каждой техники определяется: использовалась ли она для обфускации данной команды или нет. На выходе получается вектор, размерность которого равна количеству техник обфускации, состоящий из 0 и 1.

Данные о вредоносных необфусцированных командах были взяты из отчетов аналитики внутренней песочницы и из открытых источников. После обработки получилось 3 057 вредоносных необфусцированных команд. Источники, количество команд и время их загрузки показаны в таблице 1.

Таблица 1 – Вредоносные необфусцированные команды в наборе данных

Источник	Число команд	Период
Internal sandbox reports	495	с 2019 по 2021
Palo Alto’s research data	1 644	до 2017
Any.run	85	2021
FileScan.IO	175	2021
Hatching Triage	156	с 2020 по 2021
VirusShare	502	с 2020 по 2021

Обфусцированные данные были сгенерированы вручную с помощью утилиты Revoke-Obfuscation [17]. Конечный набор данных включал в себя 39 741 объектов, среди которых 3 057 необфусцированных и 36 684 обфусцированных.

Использовались и сравнивались следующие модели машинного обучения: Linear Support Vector Machine (SVM), Random Forest (RF), K-Nearest Neighbors (KNN) и Decision Tree (DT). Параметры для моделей представлены в таблице 2.

Таблица 2 – Параметры моделей машинного обучения

Алгоритм	Параметры
SVM	SVC, kernel=«linear», C=1, random_state=42
RF	n_estimators=300, max_depth=19, random_state=42
KNN	n_neighbors=3, p=1, manhattan_distance
DT	max_depth=15, random_state=42

Помимо моделей машинного обучения, была использована нейронная сеть Deep Neural Network (DNN). Архитектура нейронной сети представлена в таблице 3.

Таблица 3 – Архитектура нейронной сети (Learning rate: 0,0001; Batch size: 256; Optimizer: Adam)

Слой	Число нейронов	Функция активации
Dense	512	ReLU
Dense	256	ReLU
Dense	128	ReLU
Dense	64	ReLU
Dense	32	ReLU
Dense	15	Sigmoid

Для оценки качества результатов использовались следующие метрики, представленные ниже.

1. Hamming Loss (HL) – показывает сколько раз в среднем предсказанное значение записи к метке класса было предсказано неверно.
2. Accuracy Score (AS) – показывает долю записей, у которых все метки классифицированы корректно.
3. Precision Score (PS) – отношение правильно классифицированных положительных значений к общему количеству предсказанных положительных значений.
4. Recall Score (RS) – отношение правильно классифицированных положительных значений к их общему количеству.
5. F1 Score (F1) – среднее гармоническое PS и RS.

Вычисленные метрики для обученных моделей машинного обучения и нейронной сети представлены в таблице 4.

Таблица 4 – Сравнение метрик для используемых алгоритмов

Алгоритм	HL	AS	PS	RS	F1
SVM	0,94	87,26	95,76	93,65	94,56
RF	0,18	97,56	99,69	98,98	99,33
KNN	0,69	93,16	96,96	95,88	96,29
DT	0,43	95,91	98,22	98,13	98,17
DNN	0,18	99,82	99,44	99,53	99,48

В данном исследовании нейронная сеть почти по всем показателям превосходит алгоритмы машинного обучения и лучше справляется с многоклассовой классификацией обфусцированных команд PowerShell.

Detecting Malicious PowerShell Commands using Deep Neural Networks [18].

В данной работе реализована бинарная классификация обфусцированных вредоносных PowerShell команд с использованием различных архитектур нейронных сетей.

Источником данных являлась компания Microsoft. Набор данных содержал:

1) 60 098 безопасных необфусцированных PowerShell команд, которые были выполнены пользователями корпоративной сети Microsoft;

2) 5 819 вредоносных обфусцированных команд, которые были преднамеренно заражены различными видами вредоносных программ и выполнены на виртуальных машинах;

3) 471 вредоносную обфусцированную команду, каждая из которых была получена другими средствами.

Для обучения и кросс-валидации использовались 5 819 (20%) вредоносных и 48 094 безопасных команд (80%). Чтобы избежать дисбаланса тренировочных данных, число вредоносных команд было дублировано до числа безопасных команд. Для теста были применены оставшиеся 471 вредоносная команда и 12 004 безопасных команд.

В качестве алгоритмов были реализованы следующие детекторы, основанные на обработке естественного языка (NLP) и сверточных нейронных сетях (CNN), представленные ниже.

1. 9-слойная CNN (9-CNN), содержащая 6 сверточных слоев. Среди них 2 dropout слоя, необходимых для регуляризации и минимизации переобучения сети, 1 max pooling слой с шагом продвижения stride, равным 1, 2 полносвязных слоя с 256 входами, имеющих размеры 1 024 и 2 048 нейрона, и 1 выходной слой.

2. 4-слойная CNN (4-CNN) – «уменьшенная» версия архитектуры 9-CNN, содержащая только 1 сверточный слой с 128 ядрами размером 62x3 и шагом продвижения stride, равным 1, за которым следует max pooling слой размера 3. Далее следуют 2 полносвязных слоя, равновероятно следующих после dropout слоя, оба размером 1 024 нейрона, и выходной слой. Архитектура без учета dropout слоя представлена на рисунке 2.

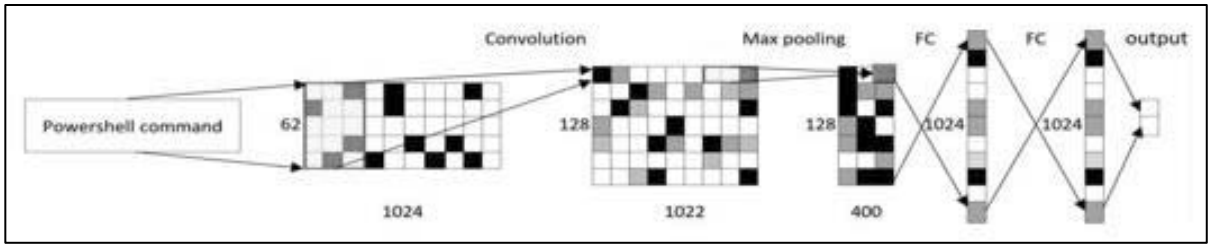


Рисунок 2 – Архитектура 4-CNN

3. 4-CNN* – вариант 4-слойной CNN, в котором не используется case bit, предназначенный для сохранения дополнительной информации о регистре букв.

4. Рекуррентная нейронная сеть, состоящая из LSTM блоков. Содержит embedding слой размера 32. Используемые LSTM блоки – двунаправленные ячейки LSTM с размерностью выхода 256 нейронов. Далее следуют 2 полносвязных слоя, каждый размером 256 нейронов, использующих dropout вероятность, равную 0,5.

5. Character level 3-gram, использующая метрику TF-IDF и классификатор логистической регрессии.

6. BoW (Bag of Words), использующая метрику TF и классификатор логистической регрессии.

Используемые методы оценки важности слов в документе (команде):

1) TF (Term Frequency) – показывает частоту появления определенного слова в документе;

2) IDF (Inverse Document Frequency) – показывает частоту документов, в которых встречается определенное слово;

3) TF-IDF – совокупный показатель, равный произведению TF и IDF.

Для глубокого обучения моделей использовался стохастический градиентный спуск (Minibatch SGD), для которого на каждой из 16 эпох данные для обучения делились на мини батчи размером 128 команд.

В качестве метрики оценки качества классификации использовалась AUC (Accuracy) – площадь под ROC (Receiver operating characteristic) кривой. Полученные значения метрики для каждого детектора после 2-фолдовой кросс-валидации на обучающей выборке представлены в таблице 5.

Таблица 5 – Значения AUC для детекторов на обучающей выборке

9-CNN	**4-CNN**	**4-CNN***	**LSTM**	**3-gram**	**BoW**
0,985	0,988	0,987	0,988	0,990	0,989

Все детекторы имеют высокие показатели метрики, самые лучшие результаты получились у NLP алгоритмов 3-gram и BoW.

Помимо AUC использовалась метрика Recall, для оценки которой использовались различные значения FPR (False Positive Rate). Результаты для 2-фолдовой кросс-валидации на обучающей выборке и тестовой выборки представлены в таблице 6.

Таблица 6 – Значения Recall на обучающей и тестовой выборке

FPR	**Кросс-валидация**			**Тестовая выборка**		
	10^{-2}	**10^{-3}**	**10^{-4}**	**10^{-2}**	**10^{-3}**	**10^{-4}**
9-CNN	0,95	0,89	0,73	0,72	0,52	0,24
4-CNN	0,98	0,96	0,89	0,89	0,76	0,65
4-CNN*	0,97	0,93	0,85	0,89	0,72	0,49
LSTM	0,98	0,95	0,85	0,88	0,81	0,64
3-gram	0,99	0,98	0,95	0,87	0,83	0,66
BoW	0,98	0,93	0,87	0,87	0,50	0,35

Далее были скомбинированы две лучшие модели разных типов архитектур – 4-CNN и 3-gram – в одну D/T Ensemble. Результаты сравнения на тестовых данных отдельно взятых моделей и их комбинации представлены на рисунке 3 в виде матриц ошибок (Confusion (Error) Matrices).

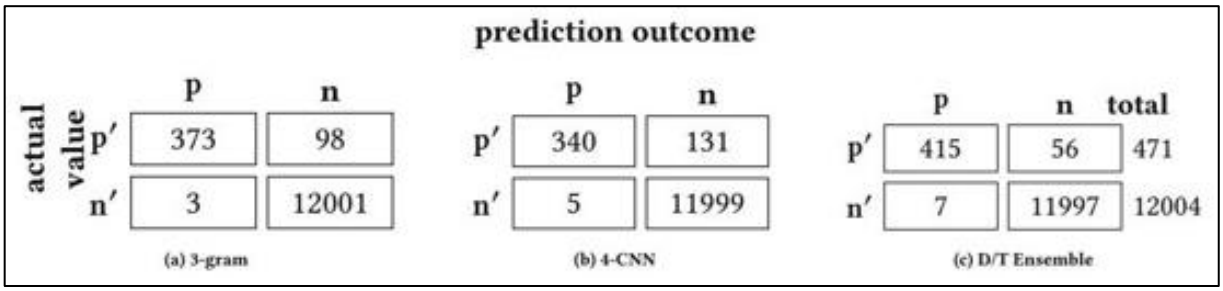


Рисунок 3 – Матрицы ошибок для моделей 3-gram, 4-CNN и их комбинации D/T Ensemble (FPR < 10^{-3})

Из сравнения можно заметить, что комбинированный вариант моделей имеет результат лучше, чем у отдельно взятых моделей.

AST-Based Deep Learning for Detecting Malicious PowerShell [19].

В данной работе реализована классификация обфусцированных вредоносных PowerShell команд с помощью глубокого обучения нейронных сетей и информации о командах из статического анализа данных, представленных в виде абстрактного синтаксического дерева (AST).

Данные об обфусцированных вредоносных PowerShell командах были взяты из открытых наборов данных Palo Alto Networks. Набор данных содержал 4 079 вредоносные команды.

Первым шагом данные о PowerShell командах были конвертированы к их AST представлениям, где содержалось 37 различных типов AST узлов.

Каждой команде была классифицирована своя метка среди типов семейств вредоносности. Для этого была использована модель Random Forest Classifier. В качестве признаков использовались глубина и число узлов Powershell AST. Настройка гиперпараметров показала, что оптимальным значением параметра максимальной глубины модели max_depth является 11.

Ввиду разреженности и несбалансированности классов данные были предобработаны и поделены на обучающие и тестовые в соотношении 70 к 30. Типы семейств вредоносности, насчитывающие менее 40 объектов (команд), были исключены.

В качестве метрики использовалась accuracy. После проведения 3-х фолдовой кросс-валидации отметка accuracy достигала 85%. Матрица ошибок на тестовой выборке представлена на рисунке 4.

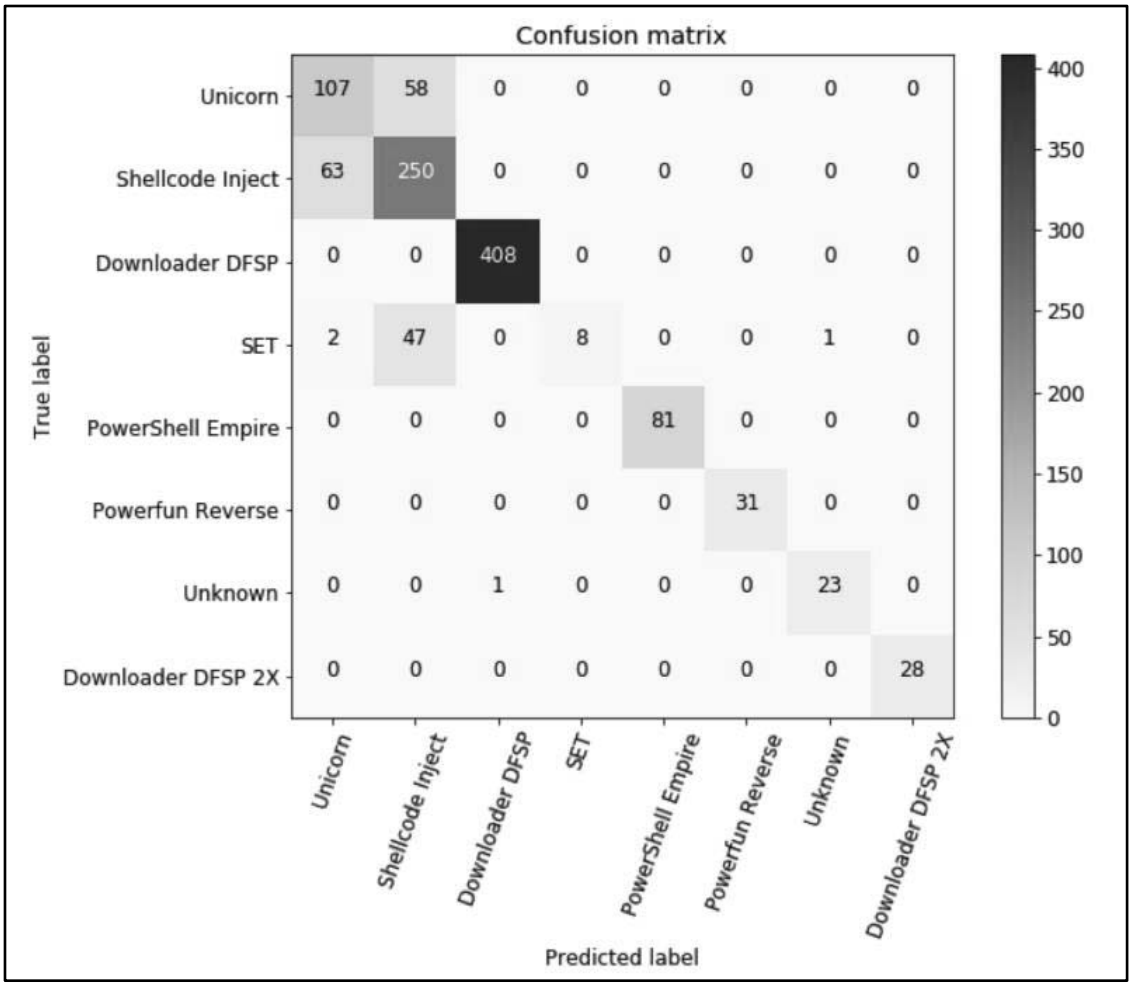


Рисунок 4 – Матрица ошибок на тестовой выборке

В данной работе двух признаков оказалось достаточно для достижения хорошей точности в классификации команд по типу обфусцированности и вредоносности, что говорит о высокой эффективности использования AST представления команд.

Evaluations of AI-based malicious PowerShell detection with feature optimizations [20].

В данной работе рассматривается использование метода оптимизации признаков для статического анализа PowerShell команд, позволяющего выделить оптимальную комбинацию признаков для определения наличия вредоносности и обфусцированности у команды с помощью применения технологий искусственного интеллекта.

Набор данных состоял из 26 475 PowerShell команд, среди которых 4 214 являлись вредоносными и 22 261 безопасными. Примерно 3 000 команд были закодированы с помощью кодировки Base64 и взяты из открытых источников, около 400 вредоносных команд содержали OLE (Object Linking and Embedding) файлы и были предоставлены компанией ESTsecurity. Оставшаяся часть данных состояла из общедоступных неразмеченных PowerShell команд, которые затем были поделены на вредоносные и безопасные с помощью ПО VirusTotal следующим образом: если анализатор фиксировал содержание хотя бы 5% вредоносности в команде, то она определялась как вредоносная, иначе как безопасная.

Для оценки используемого метода оптимизации признаков применялись 3 алгоритма машинного обучения и 3 модели глубокого обучения. Среди алгоритмов машинного обучения использовались Random Forest (RF), Support Vector Machines (SVM) и K-Nearest Neighbors (K-NN). В качестве моделей глубокого обучения были использованы Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) и их комбинация CNN-LSTM.

Архитектура модели глубокого обучения CNN представлена на рисунке 5.

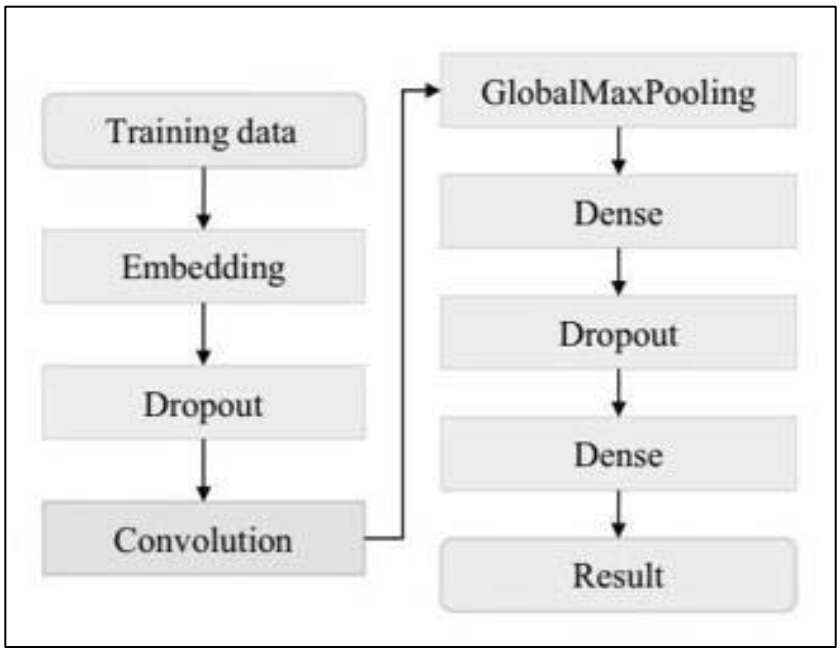


Рисунок 5 – Архитектура CNN

Архитектура модели глубокого обучения LSTM представлена на рисунке 6.

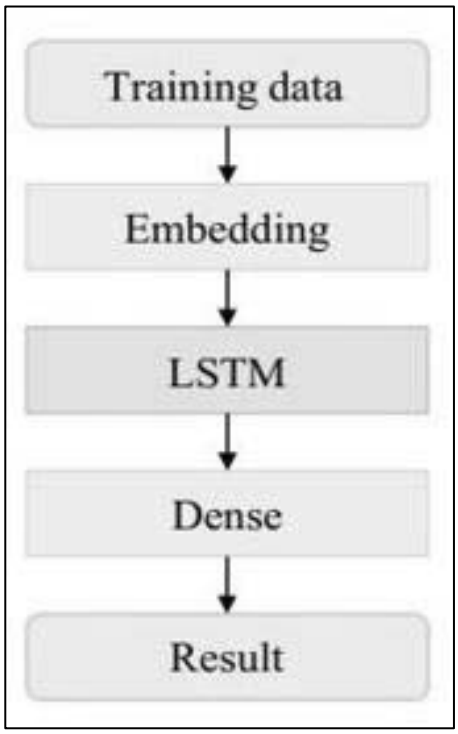


Рисунок 6 – Архитектура LSTM

Архитектура модели глубокого обучения CNN-LSTM представлена на рисунке 7.

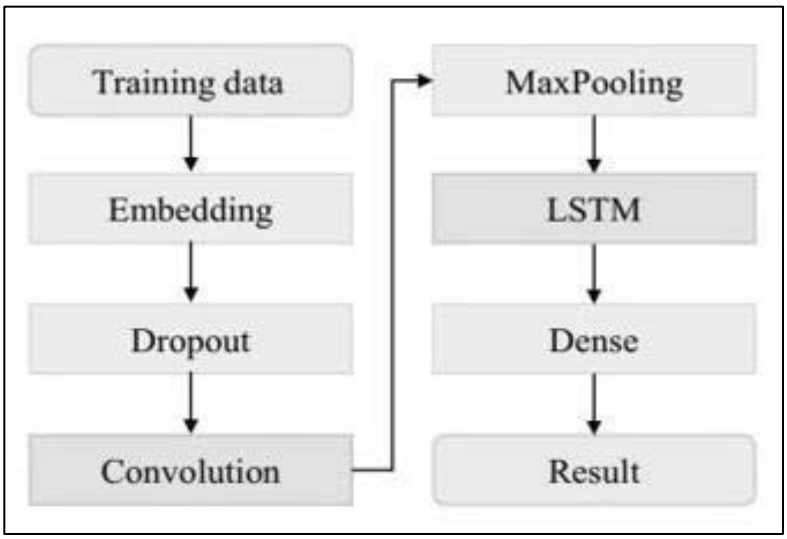


Рисунок 7 – Архитектура CNN-LSTM

Для оценки качества результатов использовались метрики Recall и False Positive Rate (FPR).

Метрика Recall вычисляется по формуле (1):

$$Recall = \frac{TP}{TP + FN}, \quad (1)$$

где TP $(True\ Positive)$ – количество правильно классифицированных обфусцированных команд;

FN $(False\ Negative)$ – количество обфусцированных команд, классифицированных как безопасные.

Метрика FPR вычисляется по формуле (2):

$$FPR = \frac{FP}{FP + TN}, \quad (2)$$

где FP $(False\ Positive)$ – количество безопасных команд, классифицированных как обфусцированные;

TN $(True\ Negative)$ – количество правильно классифицированных безопасных команд.

На рисунке 8 представлена матрица ошибок для бинарной классификации, иллюстрирующая упомянутые выше характеристики.

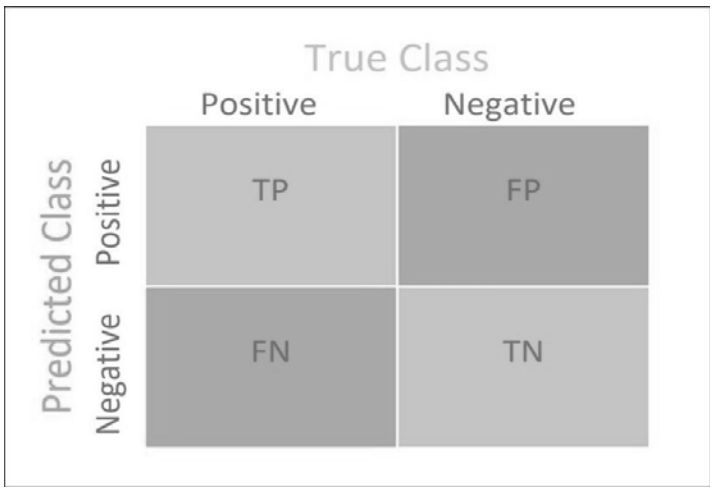


Рисунок 8 – Матрица ошибок для бинарной классификации

Экспериментальные результаты метрик после 5-фолдовой кросс-валидации на тестовых данных для различных моделей ML и DL с использованием различных методов предобработки данных (комбинаций признаков) представлены в таблицах 7 и 8.

Таблица 7 – Значения метрик на тестовых данных для ML моделей

Feature	**RF**		**SVM**		**K-NN**	
	Recall	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**
AST	88,50	0,10	84,80	0,05	87,40	1,00
AST 3-gram	89,50	0,10	86,00	0,05	87,70	0,60
AST frequency	94,20	0,20	86,50	0,60	91,10	0,50
5-token	95,60	0,30	86,30	0,10	89,70	0,60
5-token 3-gram	98,90	0,10	80,50	0,20	94,50	0,20
5-token frequency	96,10	2,20	70,00	0,20	85,10	1,20
Member	92,20	0,50	86,60	0,30	88,20	0,70
Member 3-gram	96,60	0,60	88,30	0,30	91,60	0,40
Member frequency	93,60	0,70	84,40	1,50	91,40	0,80

Таблица 8 – Значения метрик на тестовых данных для DL моделей

Feature	**CNN**		**LSTM**		**CNN-LSTM**	
	Recall	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**
AST	89,00	0,60	88,10	1,30	85,80	1,20
AST 3-gram	98,50	0,08	98,00	0,10	98,40	0,06
AST frequency	79,50	0,60	77,30	1,30	79,00	1,50
5-token	93,80	0,30	94,90	0,50	91,10	0,40
5-token 3-gram	75,10	2,80	78,20	3,90	75,50	3,60
5-token frequency	93,20	8,30	34,60	1,40	18,20	0,40
Member	92,00	0,50	94,80	0,50	90,90	0,50
Member 3-gram	92,40	0,40	91,10	0,40	91,80	0,50
Member frequency	44,00	0,80	7,90	0,00	22,10	3,20

Среди алгоритмов машинного обучения наилучших результатов достигла комбинация признаков 5-token 3-gram с применением модели Random Forest. Для моделей глубокого обучения комбинация признаков AST 3-gram заметно превосходит все остальные. Наилучший результат для метрики Recall достигается с применением CNN, а для FPR наименьшее значение получается в случае использования LSTM.

ROC кривые вместе со значением AUC для лучших моделей и комбинаций признаков представлены на рисунке 9.

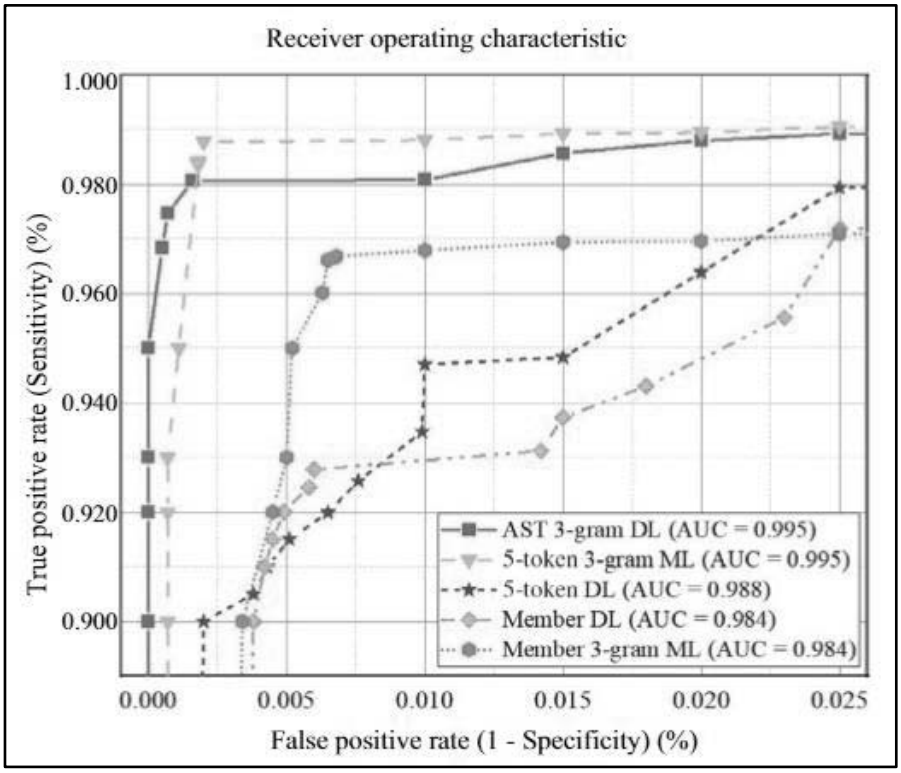


Рисунок 9 – ROC кривые для наилучших моделей

Таким образом, комбинации признаков AST 3-gram с использованием DL модели и 5-token 3-gram с использованием ML модели показали самые высокие результаты в классификации вредоносных обфусцированных PowerShell команд.

1.2. Задача бинарной классификации обфусцированных команд PowerShell

Для обфускации существуют следующие техники:

1) short – использование сокращений;
2) variables – манипуляции с переменными среды;
3) encoding – изменение кодировки;
4) string – манипуляции со строками;
5) symbol – использование игнорируемых символов.

Пример обфусцированной команды для каждой техники обфускации представлен в таблице 9.

Таблица 9 – Примеры обфусцированных команд

Техника обфускации	**Обфусцированная команда**	**Необфусцированная команда**
short	`powershell -NoExi "$os = Get-WmiObject win32_operatingsystem $uptime = (Get-Date) - $os.ConvertToDateTime($os.LastBootUpTime) Write-Output (\"Last boot: \" + $os.ConvertToDateTime($os.LastBootUpTime))"`	`$os = Get-WmiObject win32_operatingsystem $uptime = (Get-Date) - $os.ConvertToDateTime($os.LastBootUpTime) Write-Output ("Last boot: " + $os.ConvertToDateTime($os.LastBootUpTime))`
variables	`C:\WINDOWS\system32\cmd.exe/c "set fnqj=$os = Get-WmiObject win32_operatingsystem $uptime = (Get-Date) - $os.ConvertToDateTime($os.LastBootUpTime) Write-Output ("Last boot: " + $os.ConvertToDateTime($os.LastBootUpTime)) &&set bhpo=echo ${ExecutionContext}.InvokeCommand.InvokeScript((Item env:fnqj).Value) ^|powershell ${Input}^^^|Invoke-Expression&&C:\WINDOWS\system32\cmd.exe/c%bhpo%"`	`$os = Get-WmiObject win32_operatingsystem $uptime = (Get-Date) - $os.ConvertToDateTime($os.LastBootUpTime) Write-Output ("Last boot: " + $os.ConvertToDateTime($os.LastBootUpTime))`
encoding	`Invoke-Expression ( ('Add-ADG'+'r'+'oupMembe'+'r'+' -Ide'+'ntit'+'y group'+'-name -Me'+'mbers'+' S'+'se'+'r1, us'+'e'+'r2'))`	`Add-ADGroupMember -Identity group-name -Members Sser1, user2`
string	`$members = Import-CSV ("{0}{5}{4}{1}{2}{3}" -f'c:it-ad','o-gro','up','.csv','t','d-') | Select-Object -ExpandProperty ("{0}{1}{2}{3}" -f'sa-macc','ount','n','ame') ("{0}{3}{2}{1}"-f'A','r','ou-pMembe','dd-ADGr') -Identity ("{2}{1}{0}" -f 'e-rw','v','hr-n-dri') -Members $members`	`$members = Import-CSV c:it-add-to-group.csv | Select-Object -ExpandProperty samaccountname Add-ADGroup-Member -Identity hr-n-drive-rw -Members $members`
symbol	`Add-AD`Grou`pMe`mber -Identity group-name -Members Sser1, user2`	`Add-ADGroupMember -Identity group-name -Members Sser1, user2`

В общем виде задача бинарной классификации обфусцированной команды PowerShell определяется следующим образом: на вход подаются некоторые данные, характеризующие команду, а на выходе имеется один из двух классов, к которому данная команда принадлежит.

Для классификации команды необходимо определить основные характеристики. Для этого будет использоваться представление команды в виде абстрактного синтаксического дерева (AST), на основе которого используемая утилита Revoke-Obfuscation будет подсчитывать входящие в нее символы, их распределения и другую информацию.

В результате для команды будет определен класс: обфусцированная или необфусцированная, значения 1 и 0.

1.3. Методы машинного обучения

Обучение с учителем

Обучение с учителем – это один из методов обучения моделей, при котором модель обучается на основе набора данных, где каждый объект имеет соответствующий выходной результат или метку [21]. Иначе говоря, модель учится предсказывать правильный результат, полагаясь на метки входных данных.

Naïve Bayes [22]

Метод наивного Байеса – это статический алгоритм машинного обучения, основанный на теореме Байеса и используемый в основном для классификации текстовых данных. Алгоритм предполагает, что все признаки независимы друг от друга, что делает его «наивным».

Пусть имеется набор данных, содержащий M признаков, объекты которого могут принадлежать N различным классам. Существует конечное множество классов $\{c_1, \dots, c_N\}$. Необходимо определить такой класс c_k, чтобы вероятность принадлежности объекта $X = \left(x^{(1)}, \dots, x^{(M)}\right)$ данному классу была максимальна: $c_k = \underset{c_i}{\operatorname{argmax}}[P(c_i|X)]$.

Вероятность принадлежности объекта X классу c_k вычисляется по формуле Байеса (3):

$$P(c_k|X) = \frac{P(c_k) * P(X|c_k)}{P(X)}, \quad (3)$$

где $P(X|c_k)$– совокупная вероятность всех признаков для класса c_k.

Так как признаки независимы, то вероятности $P(X)$ и $P(X|c_k)$ вычисляются по следующим формулам (4) и (5):

$$P(X) = \prod_{i=1}^{M} P\left(x^{(i)}\right), \quad (4)$$

$$P(X|c_k) = \prod_{i=1}^{M} P\left(x^{(i)}|c_k\right). \quad (5)$$

В случае непрерывных случайных величин предполагается использовать вместо вероятности $P(X|c_k)$ плотность распределения $f(X|c_k)$.

Данный метод является достаточно быстрым и, несмотря на строгость независимости признаков, эффективным при анализе текста.

Ниже приведены два классических алгоритма машинного обучения, которые были использованы для решения задачи бинарной классификации команд PowerShell.

Multinomial Naive Bayes [23]

Multinomial Naive Bayes – это алгоритм машинного обучения, основанный на методе наивного Байеса. Применяется в случае полиномиального распределения признаков в используемых данных.

Алгоритм содержит важный гиперпараметр smoothing, который незаменим в случаях, когда значение какого-либо признака объекта равно 0. Полученное нулевое значение способствует нулевой вероятности признака для определенного класса, которая при перемножении аннулирует совокупную вероятность объекта для данного класса. Гиперпараметр «сглаживает» значение вероятности, не допуская аннулирования совокупной вероятности путем добавления указанного значения к значениям всех признаков всех объектов и пересчета вероятностей, учитывая обновленные значения.

Gaussian Naive Bayes [24]

Gaussian Naive Bayes – это алгоритм машинного обучения, основанный на методе наивного Байеса. Применяется в случае нормального распределения (распределения Гаусса) признаков в используемых данных.

Алгоритм вычисляет вероятность принадлежности объекта каждому классу, основываясь на среднем значении и стандартном отклонении признаков в каждом классе.

С помощью плотности нормального распределения вероятность $P\left(x^{(i)}|c_k\right)$ в данном случае вычисляется по формуле (6):

$$P\left(x^{(i)}|c_k\right) = \frac{1}{\sqrt{2\pi\sigma}} exp\left(-\frac{\left(x^{(i)} - \mu_{c_k}\right)^2}{2\sigma_{c_k}^2}\right). \quad (6)$$

Support Vector Machines [25]

Support Vector Machines (SVM) – это алгоритм машинного обучения, широко применяющийся для задач классификации и регрессии.

Основная идея алгоритма заключается в поиске оптимальной разделяющей гиперплоскости между классами данных, которая максимально удалена от ближайших к ней точек обучающего набора данных, называемых опорными векторами.

Для обработки нелинейных данных алгоритм использует ядровые функции, которые позволяют проецировать данные в пространство более высокой размерности, где они становятся линейно разделимыми.

SVM обычно включает регуляризацию для управления сложностью модели и предотвращения переобучения.

K-Nearest Neighbors [26]

K-Nearest Neighbors (K-NN) – это алгоритм машинного обучения, используемый для задач классификации и регрессии.

Алгоритм основывается на предположении, что близкие объекты в пространстве признаков имеют схожие значения целевой переменной.

Основная идея K-NN заключается в том, что объект классифицируется путем голосования: если большинство его K ближайших соседей принадлежат определенному классу, то объект также относится к этому классу.

Одним из ключевых параметров K-NN является выбор числа соседей. Значение K влияет на гладкость границы принятия решений: меньшие значения могут привести к более чувствительным моделям, тогда как большие значения могут привести к более сглаженным границам.

K-NN использует различные метрики для измерения расстояния между объектами в пространстве признаков, например, евклидово расстояние, манхэттенское расстояние, косинусное расстояние и другие.

Преимущества K-NN включают простоту реализации, способность обрабатывать многоклассовые задачи и способность работать без предположений о данных.

Но в то же время среди недостатков выделяется высокая вычислительная сложность при работе с большими объемами данных и чувствительность к выбору метрики расстояния и числу соседей.

Logistic Regression [27]

Логистическая регрессия – это алгоритм машинного обучения, очень часто применяемый для бинарной классификации.

Алгоритм основан на модели линейной регрессии, но вместо простой линейной функции для предсказания значения зависимой переменной применяется функция сигмоиды, которая ограничивает выходное значения в интервале от 0 до 1. Формула функции сигмоиды имеет следующий вид (7):

$$\sigma(x) = \frac{1}{1+e^{-x}}. \quad (7)$$

В случае M признаков в наборе данных входной пример представляет собой вектор из как минимум M элементов и равен, например, $X = \left(1, x^{(1)}, \dots, x^{(M)}\right)$. Формула модели имеет следующий вид (8):

$$f(X) = \sigma\left(\sum_{i=0}^{M} w^{(i)} * x^{(i)}\right). \quad (8)$$

Первый элемент входного вектора X, равный 1, в данном примере используется для компактного представления скалярного представления входного примера и вектора весов, включающего смещение bias.

Обучение модели логистической регрессии, как и любой другой модели, заключается в минимизации функции потерь и подбора комбинации весов, удовлетворяющей заданным требованиям.

Формула функции потерь для произвольного классификатора, когда размер набора данных для обучения состоит из N объектов, имеет следующий вид (9):

$$L(w) = -\sum_{i=1}^{N} ln\big(P(Y = y_i|x_i)\big). \quad (9)$$

В случае бинарной классификации функция потерь $L(w)$ определяется по формуле (10):

$$L(w) = -\sum_{i=1}^{N}\big[y_i * ln\big(f(X_i)\big) + (1 - y_i) * ln\big(1 - f(X_i)\big)\big]. \quad (10)$$

Для многоклассовой классификации вводятся две новые функции: logits и SoftMax, которые позволяют определить вероятность принадлежности входного примера каждому из имеющихся классов.

Decision Tree [28]

Decision Tree – это простой для понимания и интерпретации результатов алгоритм машинного обучения, основанный на создании дерева решения, которое представляет собой последовательность вопросов, на которые можно ответить «да» или «нет», и приводит к конечному значению целевой переменной.

Процесс построения дерева решений начинается с выбора признака, который лучше всего разделяет данные на две или более группы с разными значениями целевой переменной, в зависимости от задачи. Затем происходит разбиение данных на подгруппы в соответствии с выбранным признаком. Этот процесс повторяется для каждого из полученных подмножеств, пока все данные не будут классифицированы или пока не будет достигнут критерий остановки, например, заданная максимальная глубина дерева или минимальное количество объектов в листе.

Для выбора признака в соответствующем уровне дерева используются различные методы (Gini Impurity, Entropy и Information Gain), способные оценить, какой именно признак лучше разделяет данные на группы с разными значениями целевой переменной.

Алгоритм Decision Tree гибок к входным данным и может обрабатывать набор данных, содержащий разные по типу признаки (числовые, бинарные, категориальные).

В то же время данный алгоритм подвержен переобучению, особенно в случаях, когда глубина дерева решений не ограничена. В таких ситуациях дерево полностью подстраивается под обучающие данные и неправильно работает с новыми входными данными. Для решения этой проблемы существуют методы регуляризации, такие как ограничение глубины дерева и обрезка дерева.

Random Forest [29]

Random Forest – это алгоритм машинного обучения, который используется для классификации, регрессии и других задач. Его также называют «ансамблевым» алгоритмом, так как он основан на идее комбинирования нескольких деревьев решений для достижения более точных прогнозов.

Принцип работы алгоритма заключается в создании случайного набора деревьев, каждое из которых обучается на случайной подвыборке данных. При построении каждого дерева случайно выбираются признаки для разделения узлов. Каждое дерево строится независимо от других, а затем результаты всего набора деревьев комбинируются в конечный прогноз посредством голосования.

Алгоритм случайного леса преимущественно используется в задачах с большим количеством признаков, когда некоторые признаки могут быть неинформативными или коррелированными друг с другом. Он также способен обрабатывать данные с пропущенными значениями и выбросами, что делает его работоспособным к шуму и неполным данным.

Одним из ключевых преимуществ алгоритма является его способность к снижению переобучения. Поскольку каждое дерево обучается на случайной подвыборке данных и признаков, оно менее склонно к переобучению, чем отдельное решающее дерево. Кроме того, случайный выбор признаков позволяет избежать сильной корреляции между деревьями, что повышает их разнообразие.

Однако, в отличие от деревьев решений, алгоритм случайного леса не предоставляет простой интерпретации полученной модели и требует больших ресурсов для использования из-за своей реализации.

XGBoost [30]

XGBoost (eXtreme Gradient Boosting) – это эффективный алгоритм машинного обучения, который широко применяется в задачах классификации, регрессии и ранжирования.

Алгоритм основан на технике градиентного бустинга, которая строит модель предсказания в форме ансамбля слабых предсказывающих моделей, таких как деревья решений, последовательно объединяя их результаты.

XGBoost включает в себя встроенные методы регуляризации, такие как L1 и L2 регуляризации, что способствует уменьшению переобучения и сложности модели.

Алгоритм может обрабатывать различные типы признаков, например, числовые и категориальные, а также способен обрабатывать пропущенные значения в данных.

Кроме того, XGBoost обеспечивает эффективную оптимизацию функции потерь, что помогает достичь более точных результатов при минимальном использовании ресурсов.

CatBoost [31]

CatBoost (Categorical Boosting) – это эффективный алгоритм машинного обучения, основанный на технике градиентного бустинга и предназначенный для работы с категориальными признаками в наборе данных.

Алгоритм предлагает ряд преимуществ для задач классификации, регрессии и ранжирования, особенно в контексте категориальных данных, так как способен автоматически обрабатывать категориальные признаки без предварительной обработки.

CatBoost включает в себя встроенные механизмы регуляризации, что способствует уменьшению переобучения.

Кроме того, алгоритм способен обрабатывать пропущенные значения в данных, имеет встроенные методы выбора оптимальных гиперпараметров и может осуществлять параллельное обучение.

LightGBM [32]

LightGBM (Light Gradient Boosting Machine) – это высокопроизводительный алгоритм градиентного бустинга, использующийся для задач классификации, регрессии и ранжирования и отличающийся эффективной обработкой больших объемов данных и ускоренным обучение моделей в сравнении с другими алгоритмами градиентного бустинга.

LightGBM использует методы оптимизации, такие как Gradient-based One-Side Sampling (GOSS) и Exclusive Feature Bundling (EFB), что значительно ускоряет обучение моделей на больших объемах данных.

Алгоритм поддерживает обработку категориальных признаков, параллельное обучение моделей и включает в себя встроенные методы регуляризации.

Выводы по первой главе

После обзора теоретической части было решено для анализа обфусцированных команд PowerShell использовать их AST представление, а для реализации бинарной классификации применить следующие алгоритмы машинного обучения: Multinomial Naive Bayes, Gaussian Naive Bayes, Support Vector Machines, K-Nearest Neighbors, Logistic Regression, Decision Tree, Random Forest, XGBoost, CatBoost и LightGBM.

2. ПРОЕКТИРОВАНИЕ

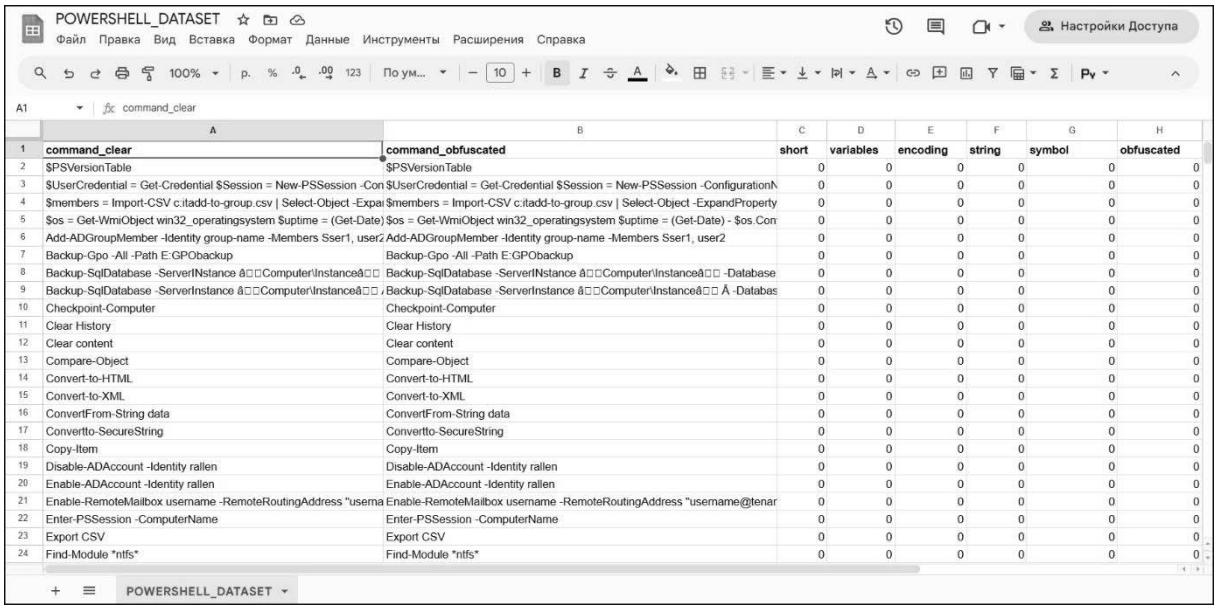
2.1. Разметка набора данных

Разметка набора данных проводилась вручную.

Для определения наличия обфусцированности у команд и примененных к ним техник обфускации использовались методические материалы и инструкции, предоставленные специалистами из компании R-Vision.

В результате было размечено 32 210 команд PowerShell: у каждой команды был указан класс наличия обфусцированности, а также примененные техники обфускации.

Пример размеченных данных представлен на рисунке 10.



	command_clear	command_obfuscated	short	variables	encoding	string	symbol	obfuscated
2	$PSVersionTable	$PSVersionTable	0	0	0	0	0	0
3	$UserCredential = Get-Credential $Session = New-PSSession -Con	$UserCredential = Get-Credential $Session = New-PSSession -ConfigurationN	0	0	0	0	0	0
4	$members = Import-CSV c:itadd-to-group.csv | Select-Object -Expa	$members = Import-CSV c:itadd-to-group.csv | Select-Object -ExpandProperty	0	0	0	0	0	0
5	$os = Get-WmiObject win32_operatingsystem $uptime = (Get-Date)	$os = Get-WmiObject win32_operatingsystem $uptime = (Get-Date) - $os.Con	0	0	0	0	0	0
6	Add-ADGroupMember -Identity group-name -Members Sser1, user2	Add-ADGroupMember -Identity group-name -Members Sser1, user2	0	0	0	0	0	0
7	Backup-Gpo -All -Path E:GPObackup	Backup-Gpo -All -Path E:GPObackup	0	0	0	0	0	0
8	Backup-SqlDatabase -ServerINstance â□□Computer\Instanceâ□□	Backup-SqlDatabase -ServerINstance â□□Computer\Instanceâ□□ -Database	0	0	0	0	0	0
9	Backup-SqlDatabase -ServerInstance â□□Computer\Instanceâ□□	Backup-SqlDatabase -ServerInstance â□□Computer\Instanceâ□□ Â -Databas	0	0	0	0	0	0
10	Checkpoint-Computer	Checkpoint-Computer	0	0	0	0	0	0
11	Clear History	Clear History	0	0	0	0	0	0
12	Clear content	Clear content	0	0	0	0	0	0
13	Compare-Object	Compare-Object	0	0	0	0	0	0
14	Convert-to-HTML	Convert-to-HTML	0	0	0	0	0	0
15	Convert-to-XML	Convert-to-XML	0	0	0	0	0	0
16	ConvertFrom-String data	ConvertFrom-String data	0	0	0	0	0	0
17	Convertto-SecureString	Convertto-SecureString	0	0	0	0	0	0
18	Copy-Item	Copy-Item	0	0	0	0	0	0
19	Disable-ADAccount -Identity rallen	Disable-ADAccount -Identity rallen	0	0	0	0	0	0
20	Enable-ADAccount -Identity rallen	Enable-ADAccount -Identity rallen	0	0	0	0	0	0
21	Enable-RemoteMailbox username -RemoteRoutingAddress "userna	Enable-RemoteMailbox username -RemoteRoutingAddress "username@tenar	0	0	0	0	0	0
22	Enter-PSSession -ComputerName	Enter-PSSession -ComputerName	0	0	0	0	0	0
23	Export CSV	Export CSV	0	0	0	0	0	0
24	Find-Module *ntfs*	Find-Module *ntfs*	0	0	0	0	0	0

POWERSHELL_DATASET

Рисунок 10 – Пример размеченных данных

Размеченные данные были проверены и верифицированы специалистами из компании R-Vision.

2.2. Требования к приложению

2.2.1. Функциональные требования

Функциональные требования – это описание функций, которые должно предоставлять программное обеспечение. Они определяют, каким

образом ПО должно взаимодействовать с пользователями, системами и другими сторонними компонентами [33].

Разрабатываемое приложение должно удовлетворять следующим функциональным требованиям:

1) приложение должно предоставлять пользователю возможность ознакомиться со справочной информацией о проекте, инструкцией по работе с ним и контактах;

2) приложение должно предоставлять пользователю возможность загрузки предобработанного набора данных;

3) приложение должно сохранять предобработанный набор данных во внешнее хранилище;

4) приложение должно выводить информацию о доступных предобработанных наборах данных, расположенных во внешнем хранилище;

5) приложение должно предоставлять пользователю возможность скачивать и удалять предобработанные наборы данных, расположенные во внешнем хранилище;

6) приложение должно предоставлять пользователю возможность ознакомиться с теоретической информацией об используемых алгоритмах машинного обучения;

7) приложение должно предоставлять пользователю возможность выбрать алгоритм для обучения;

8) приложение должно предоставлять пользователю возможность задавать параметры обучения модели;

9) приложение должно предоставлять пользователю возможность выбора предобработанного набора данных для обучения модели;

10) приложение должно сохранять обученные модели во внешнем хранилище;

11) приложение должно выводить информацию о доступных обученных моделях, расположенных во внешнем хранилище;

12) приложение должно предоставлять пользователю возможность скачивать и удалять обученные модели, расположенные во внешнем хранилище;

13) приложение должно предоставлять пользователю возможность загрузки предобработанных команд PowerShell для проверки у них наличия обфусцированности;

14) приложение должно предоставлять пользователю возможность выбора обученных моделей для определения наличия обфусцированности у загруженных предобработанных команд PowerShell;

15) приложение должно классифицировать входные данные на 2 класса: наличие или отсутствие обфусцированности;

16) приложение должно выводить информацию о результатах классификации команд PowerShell;

17) приложение должно предоставлять пользователю возможность выгрузить результаты классификации в файл;

18) приложение должно информировать пользователя о наличии ошибок на стороне клиента или сервера и уведомлять об успешных операциях.

2.2.2. Нефункциональные требования

Нефункциональные требования – это требования, которые описывают свойства и ограничения системы, не связанные непосредственно с ее функциональностью [34].

Разрабатываемое приложение должно удовлетворять следующим нефункциональным требованиям:

1) приложение должно быть запущено с помощью ПО контейнеризации Docker;

2) приложение должно иметь Web-интерфейс;

3) frontend приложения должен быть разработан с помощью языка программирования TypeScript и библиотеки React;

4) backend приложения должен быть разработан с помощью языка программирования Python и веб-фреймворка FastAPI;

5) backend приложения должен быть асинхронным и использовать Celery, Flower и Redis для выполнения ресурсоемких задач в фоне;

6) приложение должно использовать MinIO для хранения предобработанных наборов данных и обученных моделей машинного обучения;

7) приложение должно использовать MongoDB для хранения информации о файлах, моделях и результатах классификации;

8) в приложении должно быть реализовано 10 моделей машинного обучения.

2.3. Диаграмма вариантов использования

Для проектирования приложения был использован язык UML. Была разработана диаграмма вариантов использования, представленная на рисунке 11.

Главный и единственный актер – это пользователь. Пользователь может выполнять следующие действия:

1) ознакомиться с теоретической информацией об используемых алгоритмах машинного обучения;

2) посмотреть доступные предобработанные наборы данных – ознакомиться с существующими во внешнем хранилище предобработанными наборами данных;

3) загрузить/скачать предобработанный набор данных;

4) удалить предобработанный(-ые) набор(-ы) данных;

5) посмотреть доступные обученные модели – ознакомиться с существующими во внешнем хранилище обученными моделями;

6) обучить новую модель – заполнить форму настроек, выбрать алгоритм, предобработанный набор данных и объем обучающей выборки и запустить обучение;

7) скачать обученную модель, удалить обученную(-ые) модель(-и);

8) классифицировать команды PowerShell – заполнить форму, загрузить файл с предобработанными командами и выбрать обученные модели, и запустить классификацию;

9) сохранить результаты классификации в файл – сохранить информацию о каждой команде с ссылками на скачивание используемых моделей;

10) ознакомиться со справочной информацией о проекте и инструкцией по работе с ним – посмотреть цель разработки проекта и его документацию.

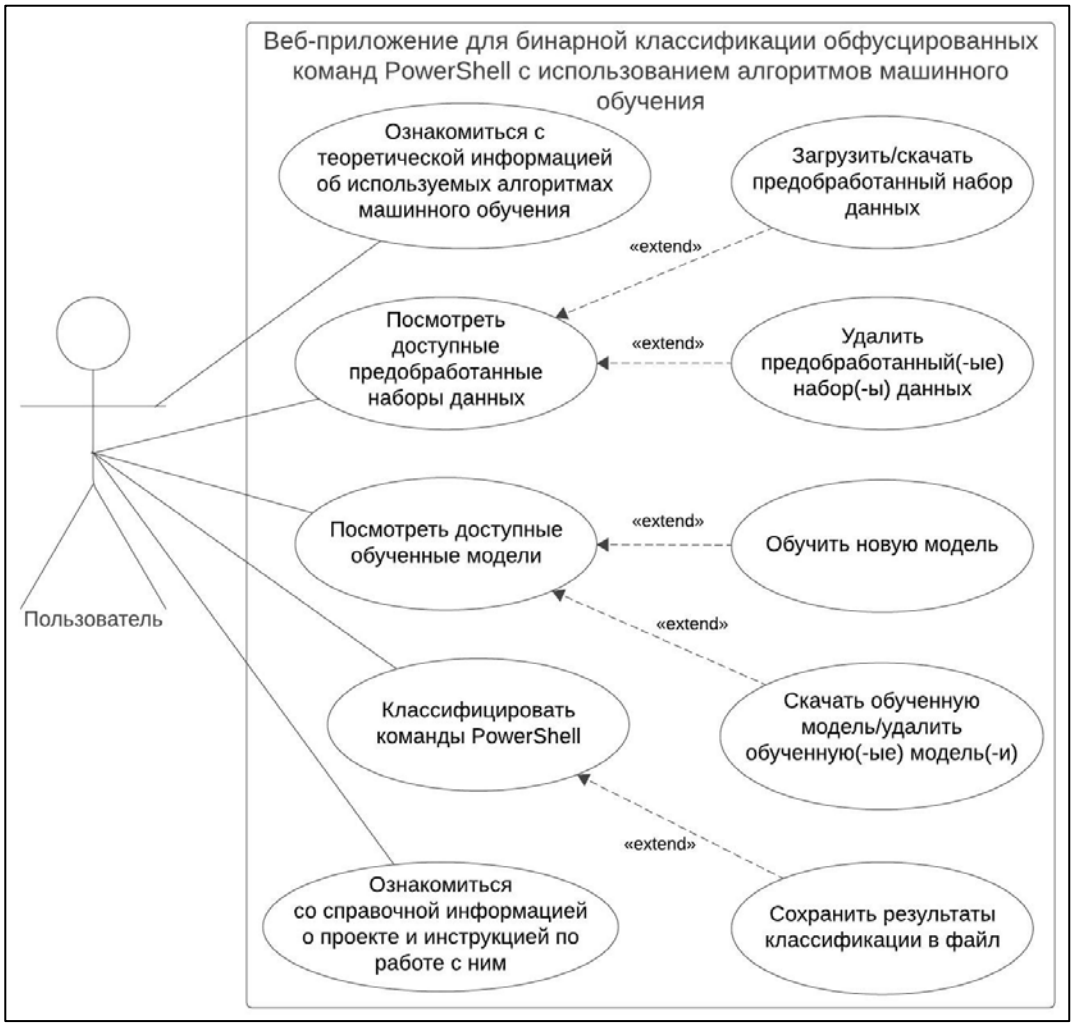


Рисунок 11 – Диаграмма вариантов использования

2.4. Архитектура приложения

На рисунке 12 представлена схема архитектуры приложения.

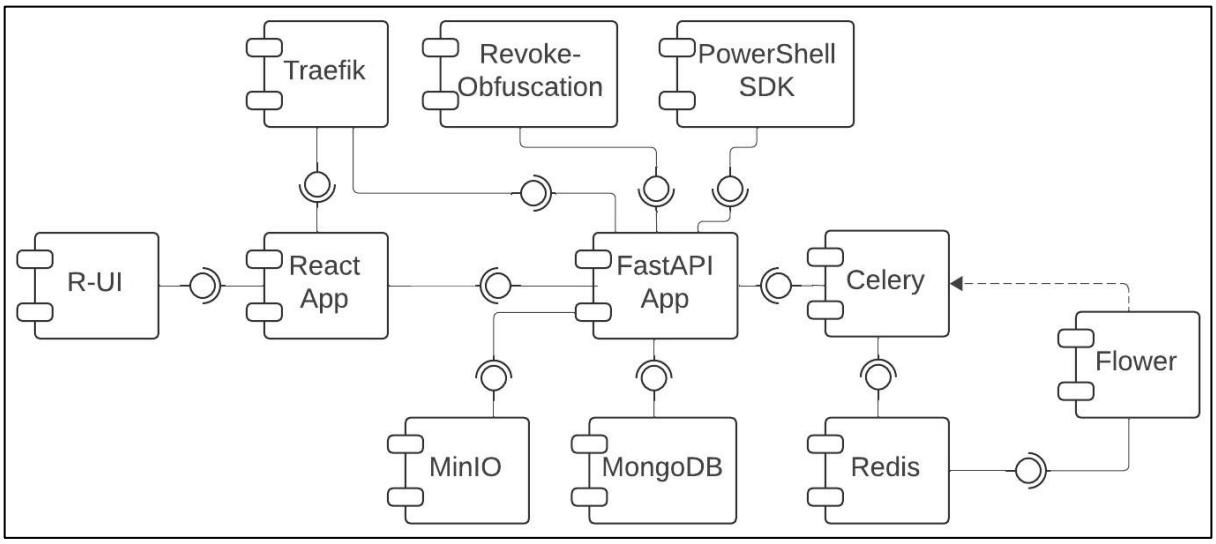


Рисунок 12 – Архитектура приложения

Архитектура приложения состоит из следующих компонентов.

1. React App – клиент веб-приложения, реализованный с помощью языка программирования TypeScript и библиотеки React.
2. R-UI – корпоративная библиотека готовых UI компонентов для React приложения.
3. FastAPI App – серверная часть веб-приложения, реализованная с помощью языка программирования Python и веб-фреймворка FastAPI.
4. Traefik – обратный прокси-сервер, необходимый для облегченного управления и маршрутизации трафика и запросов.
5. PowerShell SDK – набор инструментов PowerShell, содержащий необходимый для парсинга входных команд PowerShell и преобразования их в AST метод ParseInput.
6. Revoke-Obfuscation – утилита для подсчета метрик на основании AST представления команд PowerShell.
7. MinIO – объектное хранилище для предобработанных наборов данных и обученных моделей.

8. MongoDB – СУБД, работающая с документоориентированной моделью данных, которая необходима для хранения информации о предобработанных наборах данных, обученных моделях и результатах классификации.

9. Celery – распределенная очередь задач, необходимая для выполнения ресурсоемких задач.

10. Flower – веб-приложение для мониторинга и управления обработчиками Celery.

11. Redis – база данных типа ключ-значение с открытым исходным кодом, использующийся как broker и backend для Celery.

2.5. Проектирование базы данных

База данных приложения состоит из четырех коллекций – Datasets, Models, Classifications и CommonClassifications. Схема базы данных представлена на рисунке 13. За основу была взята графическая нотация моделирования документных баз данных [35].

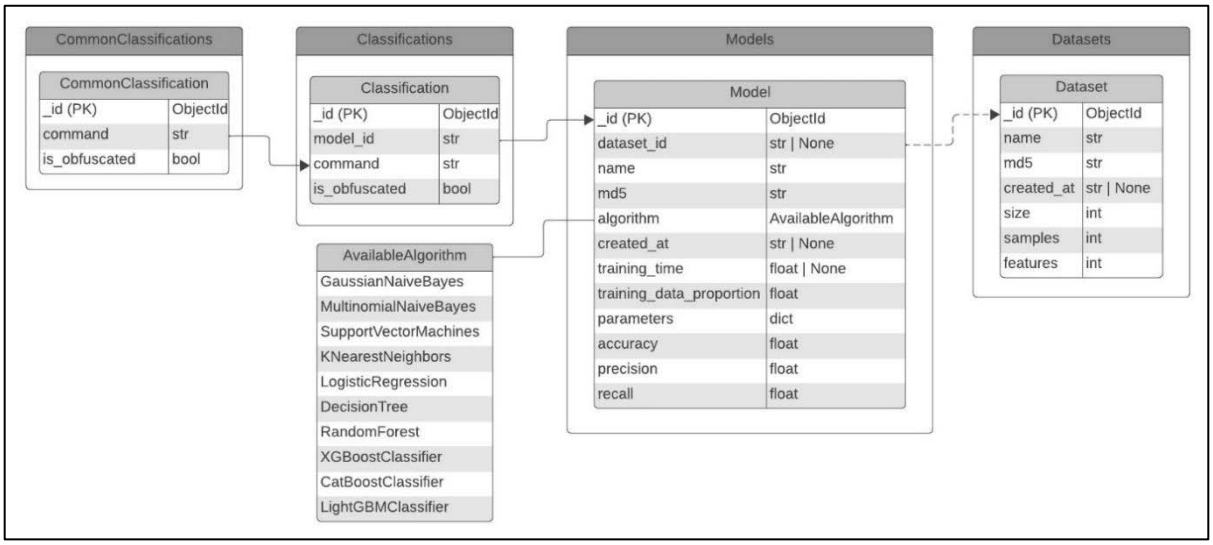


Рисунок 13 – Схема базы данных

Документ Dataset хранит информацию о предобработанных наборах данных и имеет следующие атрибуты:

1) _id – уникальный идентификатор набора данных;

2) name – имя предобработанного набора данных;

3) md5 – хэш-сумма предобработанного набора данных, которая необходима для исключения дублирования одних и тех же данных;

4) created_at – дата и время загрузки набора данных в формате «%Y-%m-%d %H:%M:%S»;

5) size – размер предобработанного набора данных в байтах;

6) samples – число объектов (команд PowerShell) в предобработанном наборе данных;

7) features – число признаков в предобработанном наборе данных, не включая целевой.

Документ Model хранит информацию об обученных моделях машинного обучения и имеет следующие атрибуты:

1) _id – уникальный идентификатор обученной модели;

2) dataset_id – id предобработанного набора данных, использованного для обучения;

3) name – имя файла обученной модели;

4) md5 – хэш-сумма файла обученной модели, необходимая для исключения дубликатов;

5) algorithm – используемый алгоритм машинного обучения;

6) created_at – дата и время начала обучения модели в формате «%Y-%m-%d %H:%M:%S»;

7) training_time – время, затраченное на обучение, в секундах;

8) training_data_proportion – объем обучающей выборки из предобработанного набора данных (значение от 0 до 1), оставшаяся часть данных используется для тестирования;

9) parameters – параметры обучения модели;

10) accuracy – метрика оценки качества, показывающая долю правильных ответов обученной модели;

11) precision – метрика оценки качества, показывающая какую долю обфусцированных команд обученная модель классифицировала правильно среди всех команд, определенных ей как обфусцированные;

12) recall – метрика оценки качества, показывающая какую долю обфусцированных команд среди всех обфусцированных команд в наборе данных обученная модель классифицировала правильно.

AvailableAlgorithm – это Enum, состоящий из строк, в котором хранятся доступные алгоритмы машинного обучения.

Документ Classification хранит информацию о результатах классификации команд PowerShell отдельно для каждой модели и имеет следующие атрибуты:

1) _id – уникальный идентификатор классификации;

2) model_id – id обученной модели, использованной для классификации;

3) command – команда PowerShell, для которой проводилась классификация;

4) is_obfuscated – результат классификации команды PowerShell для данной модели.

Документ CommonClassification хранит информацию о результатах классификации команд PowerShell в совокупности для всех моделей и имеет следующие атрибуты:

1) _id – уникальный идентификатор классификации;

2) command – команда PowerShell, для которой проводилась классификация;

3) is_obfuscated – совокупный для всех моделей результат классификации команды PowerShell.

2.6. Макеты приложения

UI приложения включает в себя 6 основных экранов:

1) главная страница – домашняя страница приложения;

2) используемые алгоритмы – набор страниц с теоретической информацией о каждом используемом алгоритме;

3) наборы данных – страница с доступными предобработанными наборами данных, где у пользователя есть возможность загрузить, скачать или удалить предобработанные наборы данных;

4) обучение – страница с доступными обученными моделями, где у пользователя есть возможность обучить новую модель на основе доступных алгоритмов, предобработанных наборов данных и заданных параметров обучения, а также скачать и удалить обученные модели;

5) классификация – страница, на которой у пользователя есть возможность классифицировать команды PowerShell, посмотреть результаты классификации для первых 100 команд и полностью выгрузить результаты классификации;

6) о проекте – набор страниц, на которых содержатся справочная информация, документация и контакты.

Макет экрана «Главная страница» представлен на рисунке 14.

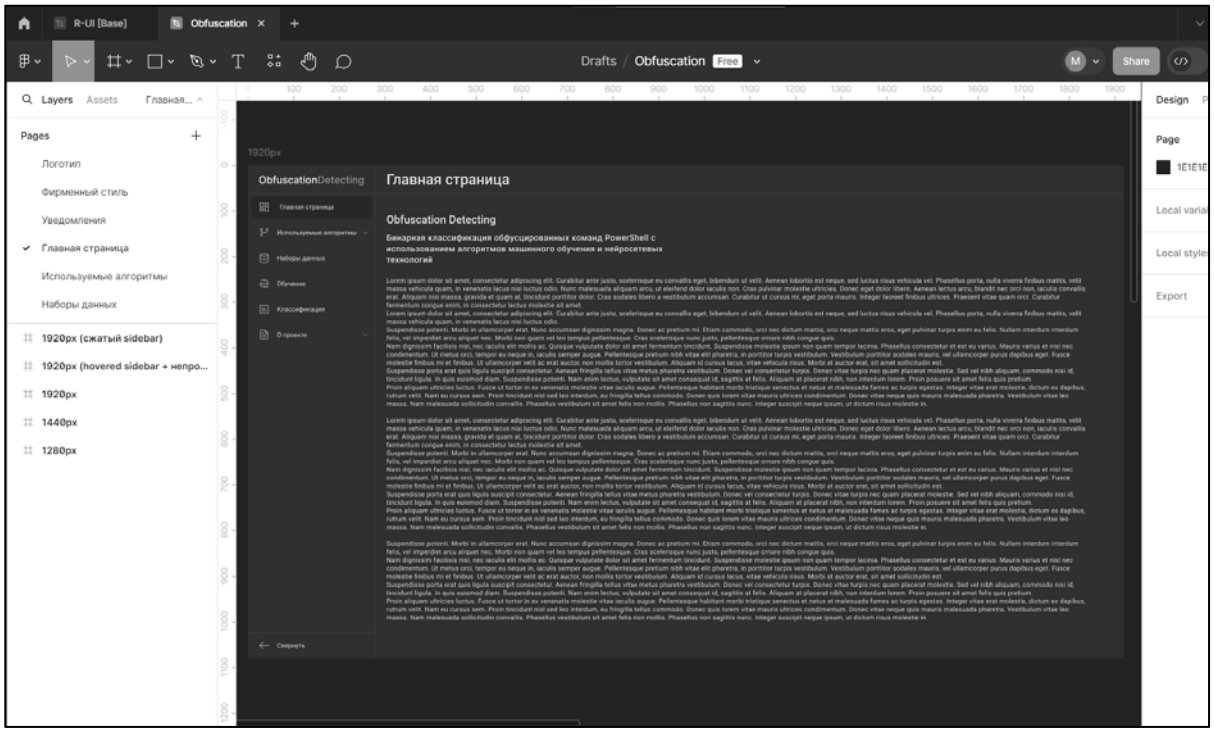


Рисунок 14 – Макет экрана «Главная страница»

Макет экрана «Используемые алгоритмы» представлен на рисунке 15.

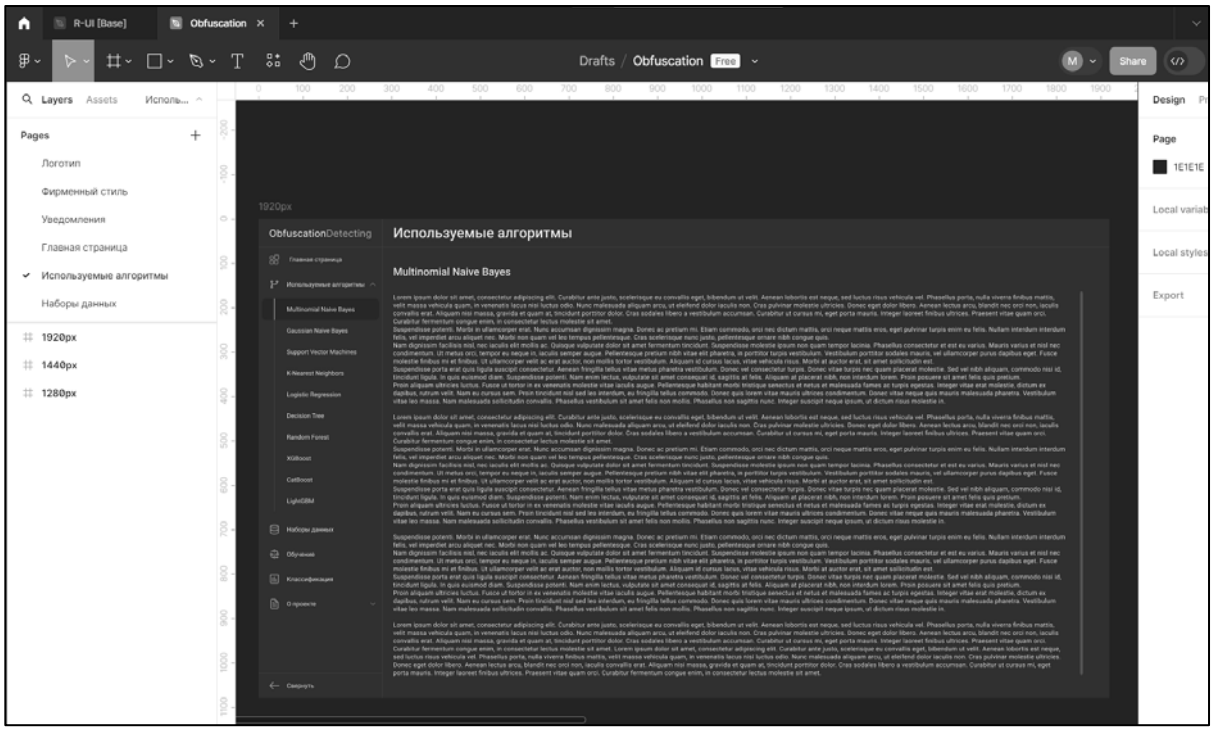


Рисунок 15 – Макет экрана «Используемые алгоритмы»

Макет экрана «Наборы данных» представлен на рисунке 16.

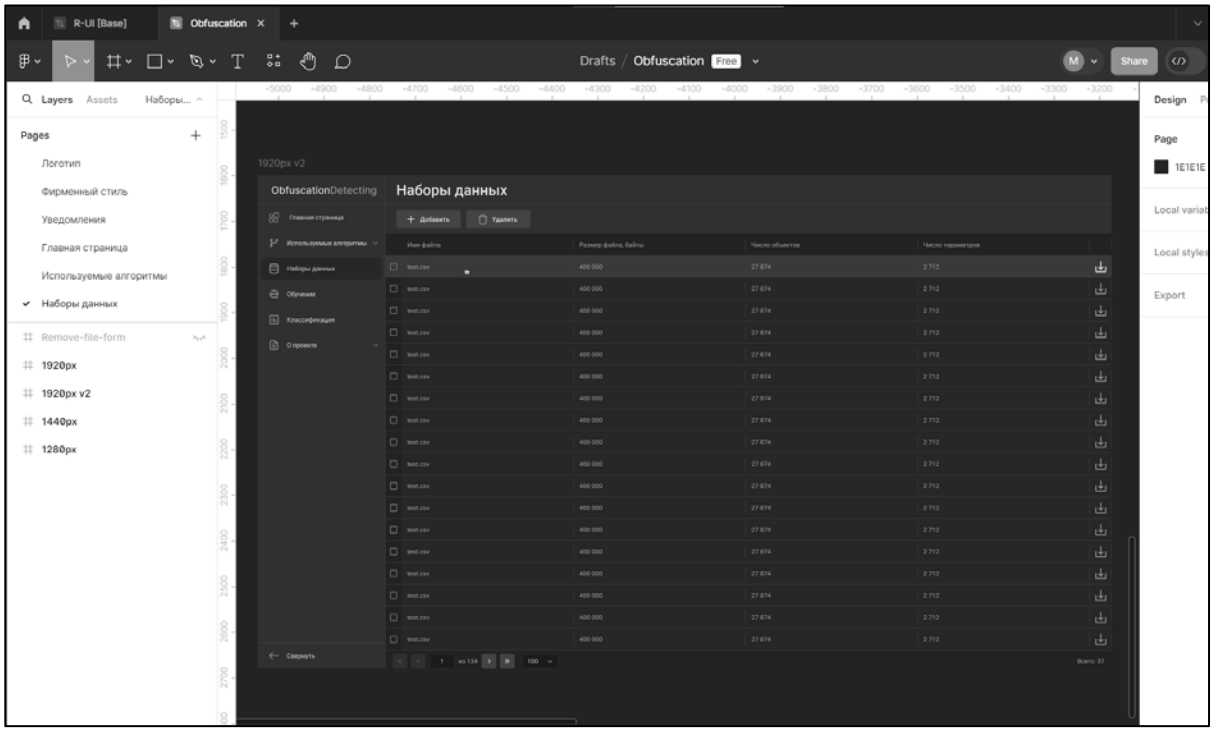


Рисунок 16 – Макет экрана «Наборы данных»

Формы для загрузки и удаления предобработанных наборов данных представлены на рисунке 17.

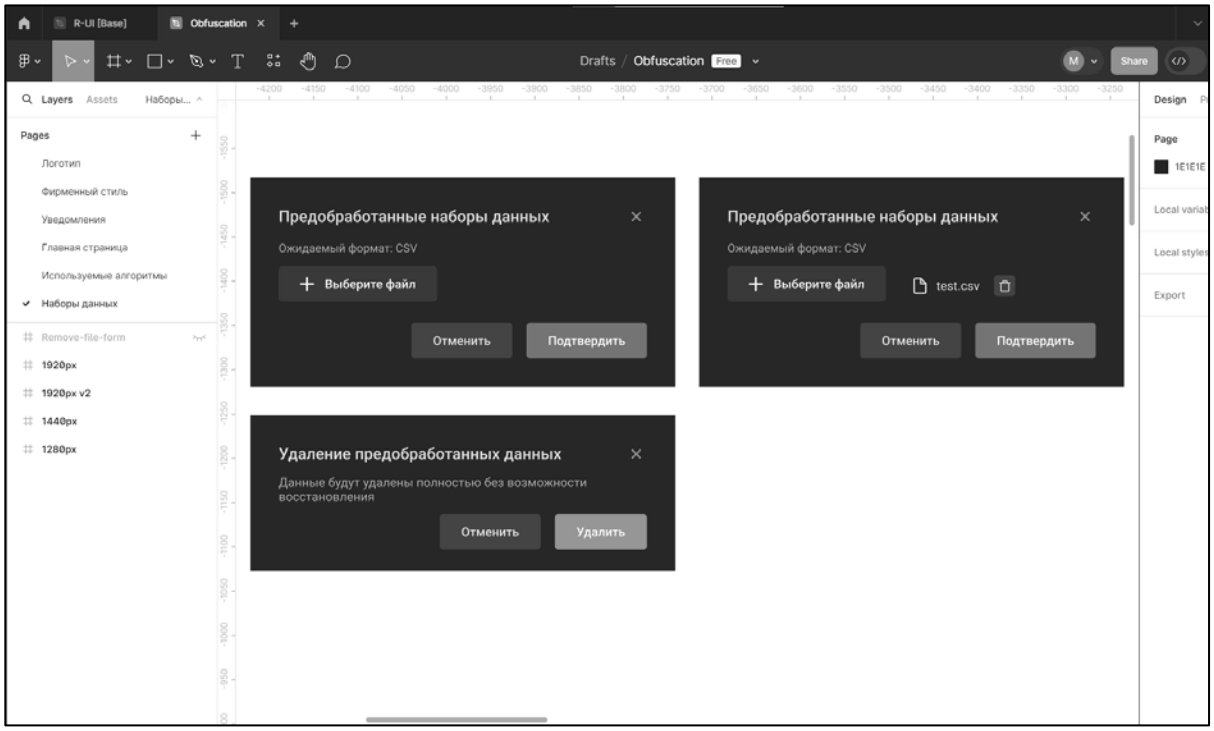


Рисунок 17 – Формы для загрузки и удаления предобработанных наборов данных

Макет экрана «Обучение» представлен на рисунке 18.

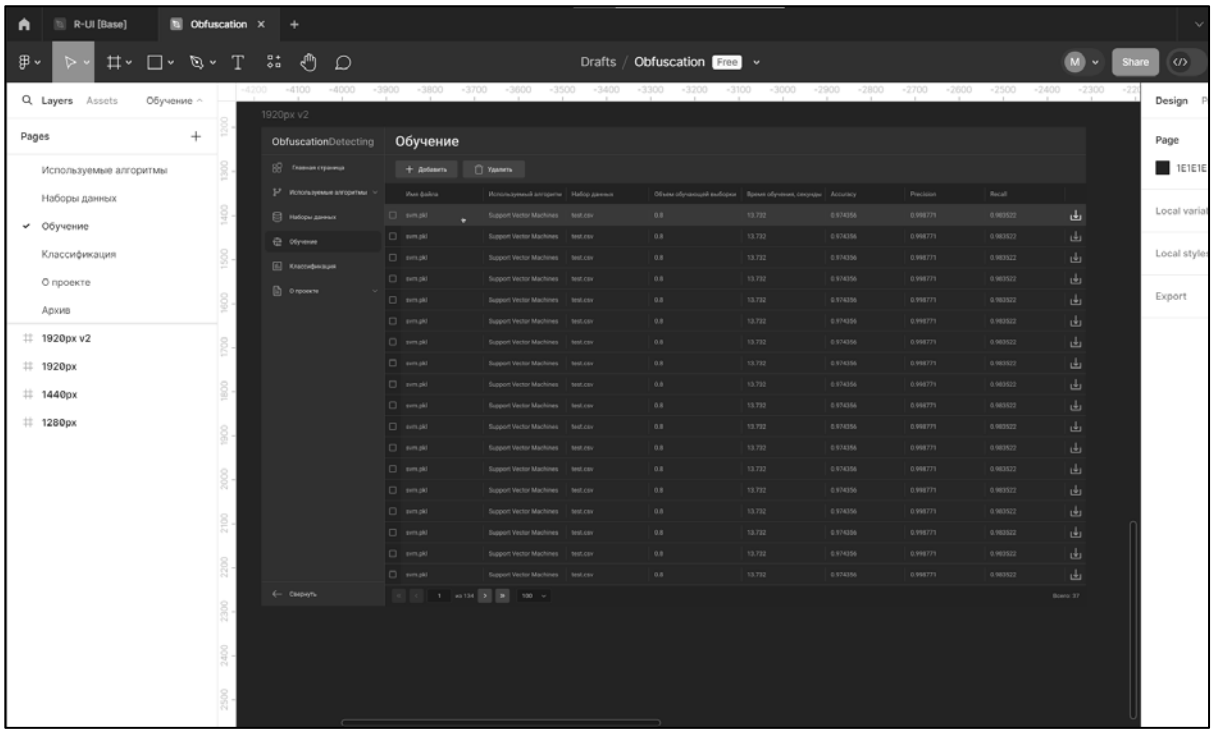


Рисунок 18 – Макет экрана «Обучение»

Формы для обучения модели и удаления обученных моделей представлены на рисунке 19.

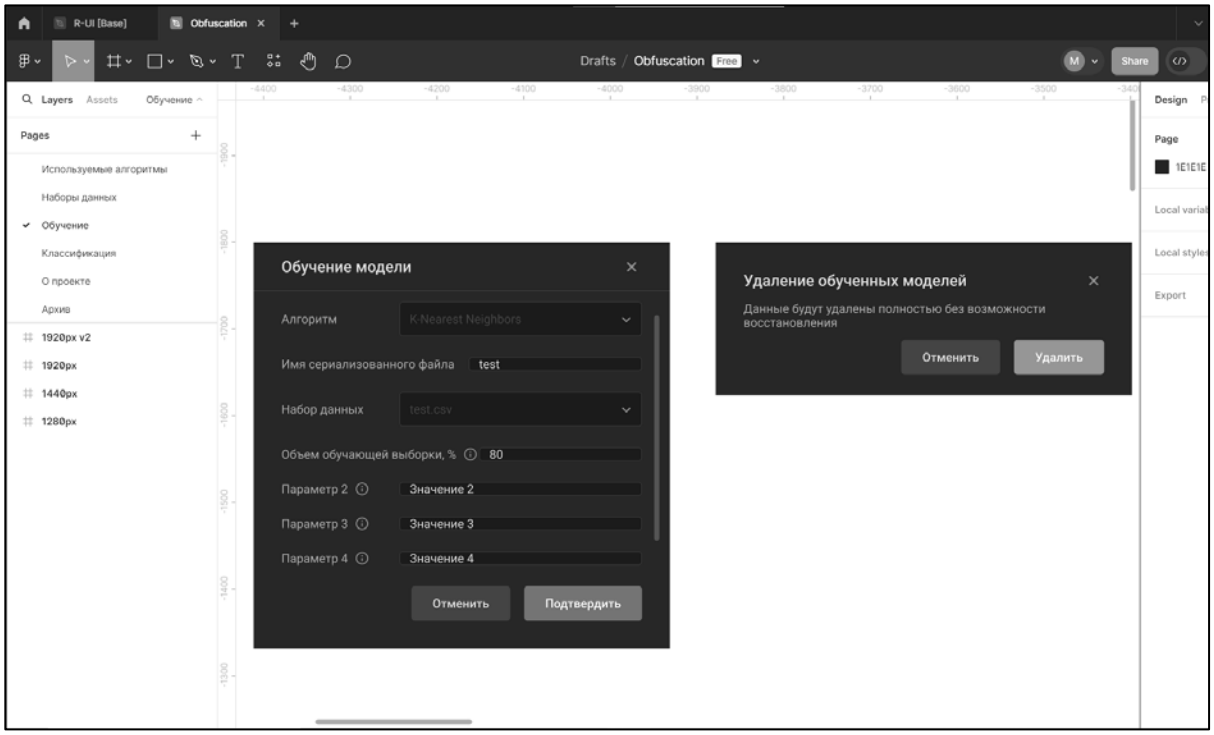


Рисунок 19 – Формы для обучения модели и удаления обученных моделей

Макет экрана «Классификация» представлен на рисунках 20 и 21.

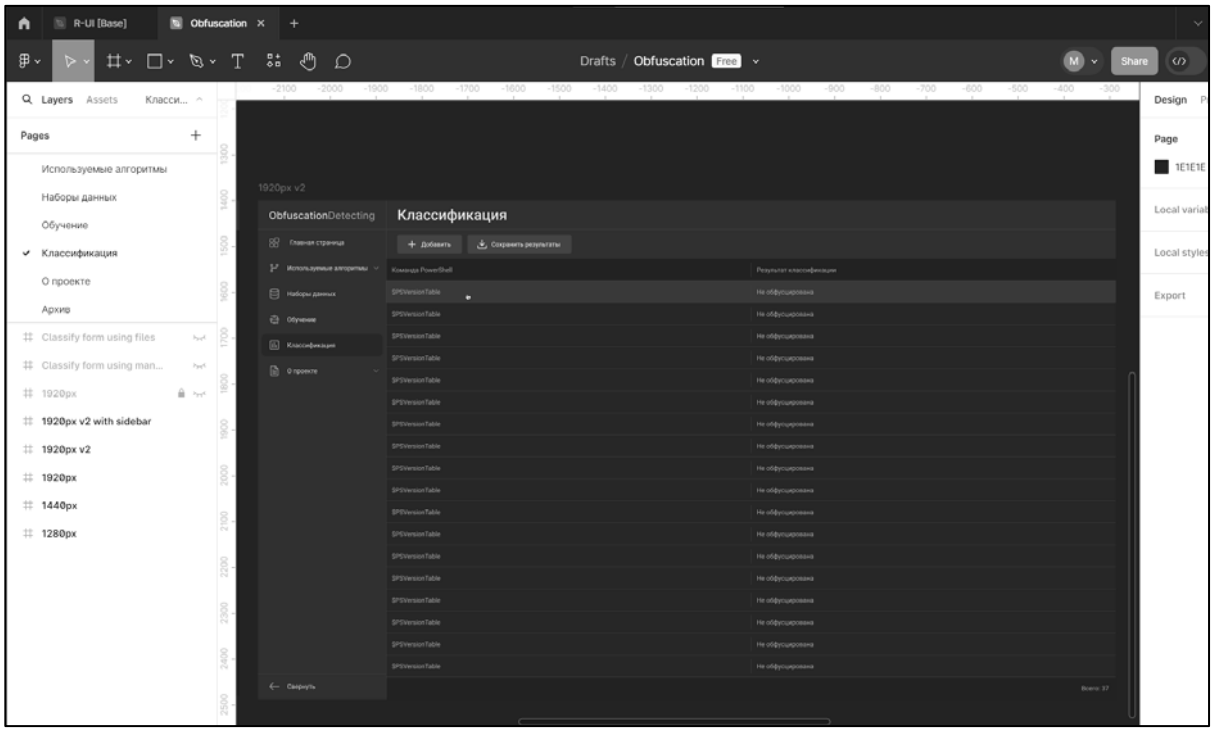


Рисунок 20 – Макет экрана «Классификация»

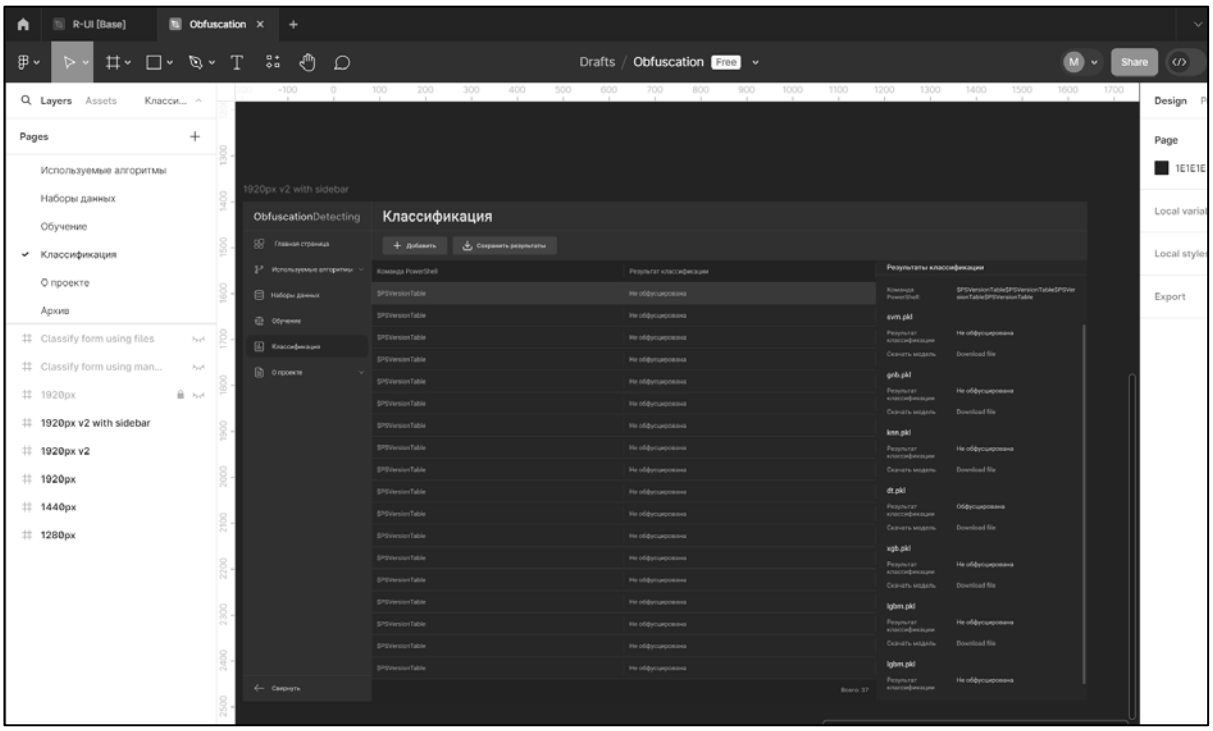


Рисунок 21 – Макет экрана «Классификация»

Формы для классификации команд PowerShell и сохранения результатов классификации представлены на рисунке 22.

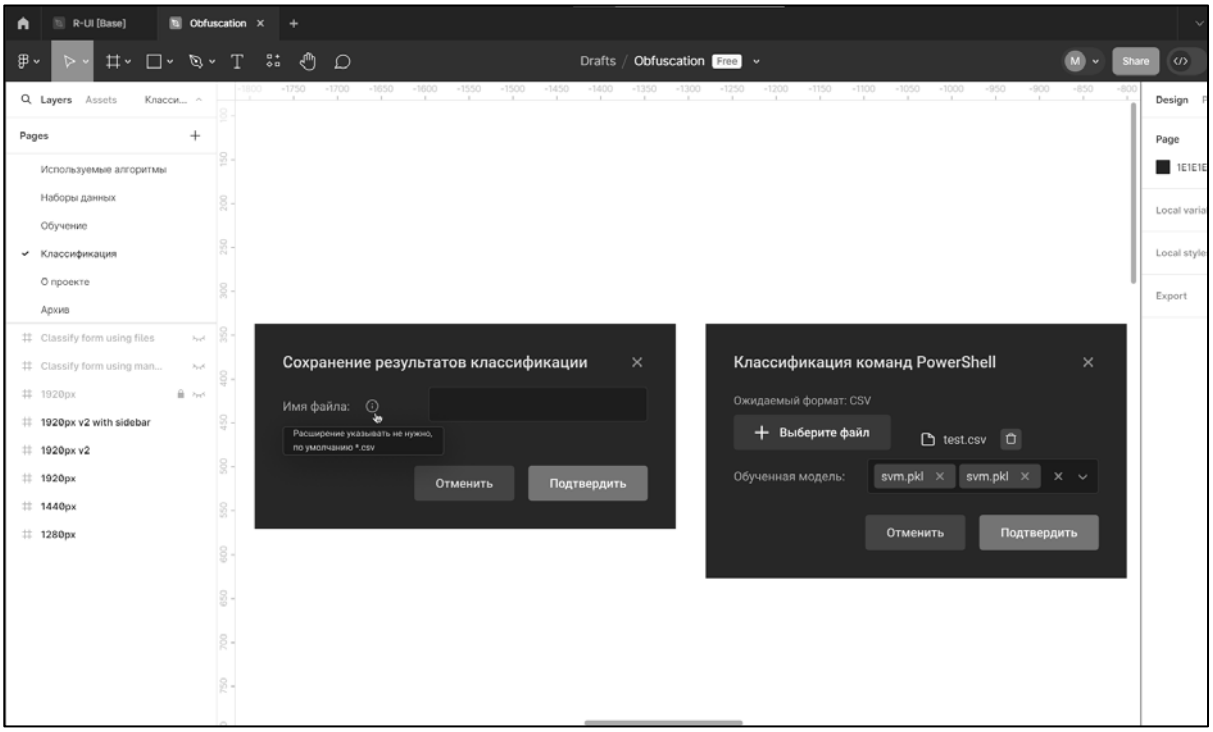


Рисунок 22 – Формы для классификации команд PowerShell и сохранения результатов классификации

Макет экрана «О проекте», на котором открыт раздел, содержащий справочную информацию, представлен на рисунке 23.

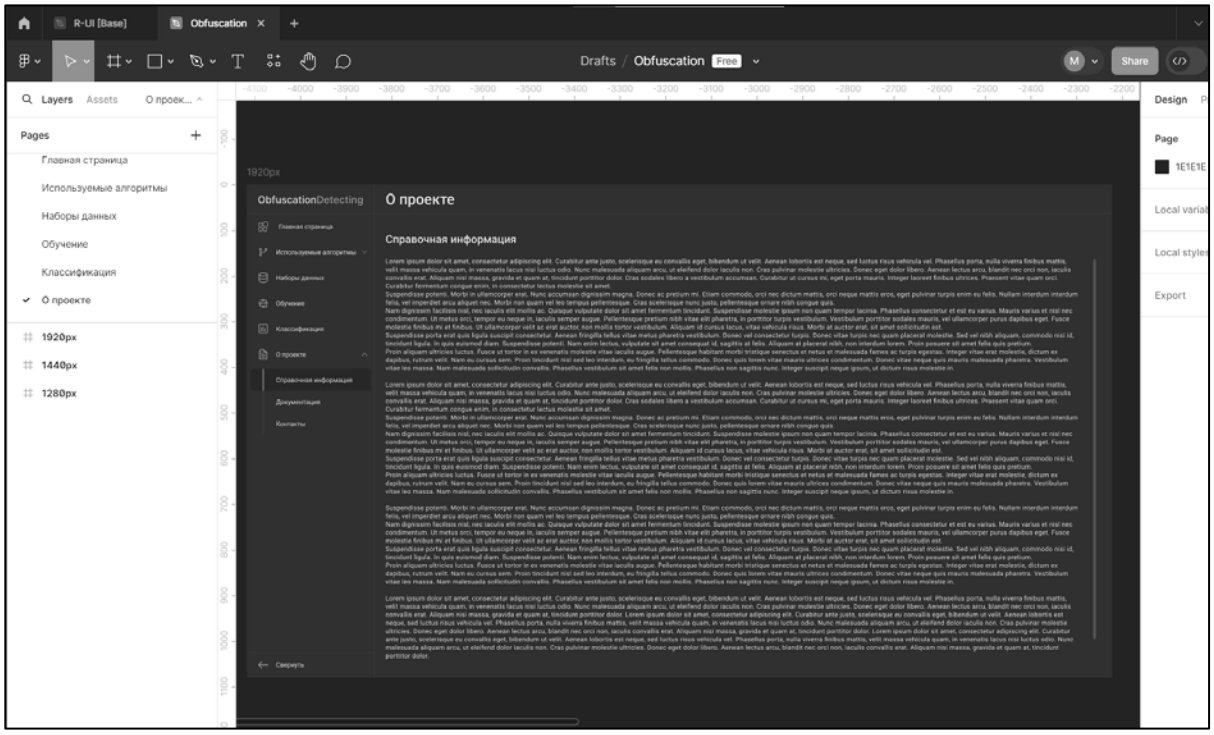


Рисунок 23 – Макет экрана «О проекте»

Выводы по второй главе

В процессе проектирования приложения были определены и сформулированы функциональные и нефункциональные требования, представлена и описана диаграмма вариантов использования, схема архитектуры приложения и базы данных, а также макеты приложения.

3. РЕАЛИЗАЦИЯ

3.1. Программные средства разработки

Для разработки серверной части веб-приложения использовался язык программирования Python 3.10 и веб-фреймворк FastAPI [36]. Для разработки клиентской части веб-приложения использовался язык программирования TypeScript 4.9 и библиотека React [37]. Разработка веб-приложения велась в IDE PyCharm 2023.1.4, а для анализа, обработки набора данных и обучения моделей использовалась интерактивная среда разработки Jupyter Notebook [38].

Для управления зависимостями и пакетами в приложении использовались менеджеры пакетов poetry [39] и npm [40].

Для предобработки данных были использованы следующие библиотеки языка программирования Python:

1) pandas – для работы с табличными данными (Excel, CSV) и их обработки [41];
2) numpy – для производительной работы с данными, представленными в виде массивов [42];
3) matplotlib [43] и seaborn [44] – для визуализации данных.

Библиотеки clr-loader [45] и pythonnet [46] были использованы для подключения встроенных в dotnet DLL-библиотек, необходимых для парсинга и токенизации команд PowerShell, и собственной DLL-библиотеки, содержащей необходимые классы и методы утилиты Revoke-Obfuscation для подсчета статистик.

Для реализации алгоритмов машинного обучения были использованы библиотеки scikit-learn [47], xgboost [30], catboost [31] и lightgbm [32]. Для сериализации и десериализации обученных моделей использовалась библиотека pickle [48].

Для взаимодействия с MinIO использовалась библиотека minio [49]. Для работы с MongoDB использовалась библиотека pymongo [50]. Для выполнения асинхронных задач использовалась библиотека celery [51]. Для взаимодействия с Redis использовалась библиотека redis [52].

Совместно с FastAPI для работы с моделями данных использовалась библиотека pydantic [53].

Совместно с React для создания интерфейса была использована корпоративная библиотека компонентов R-UI.

Для запросов клиентской части веб-приложения к серверной использовалась библиотека axios [54].

Для развертывания веб-приложения использовался Docker [55].

3.2. Предобработка данных

Размеченный набор данных был преобразован в набор признаков, характеризующих команды PowerShell. Для этого использовалась DLL-библиотека System.Management.Automation из dotnet для парсинга команды и преобразования в AST и собранная вручную DLL-библиотека Revoke-Obfuscation для подсчета метрик на полученном AST представлении команды.

В результате было получено 29 730 объектов и 4 998 параметров, а также целевой признак «obfuscated». Объектов стало меньше, чем в исходном размеченном наборе данных, из-за возникших ошибок парсинга. Код функции для преобразования команды представлен в листинге 1.

Листинг 1 – Функция для преобразования команды в набор признаков

```
def tokenize_without_features(command: str, methods: list,
cumulative_features: list[int]) -> list[float]:
    ast, token, parsing_error =
System.Management.Automation.Language.Parser.ParseInput(command)
    if parsing_error:
        raise ValueError(parsing_error)
    tokenized_values = np.zeros(cumulative_features[-1], dtype="float64")
    left_index = 0
    for i in range(len(methods)):
        right_index = cumulative_features[i]
        tokenized_values[left_index:right_index] =
list(methods[i](ast).Values)
        left_index = right_index
    return list(tokenized_values)
```

Распределение целевого признака в наборе данных до предобработки представлено на рисунке 24.

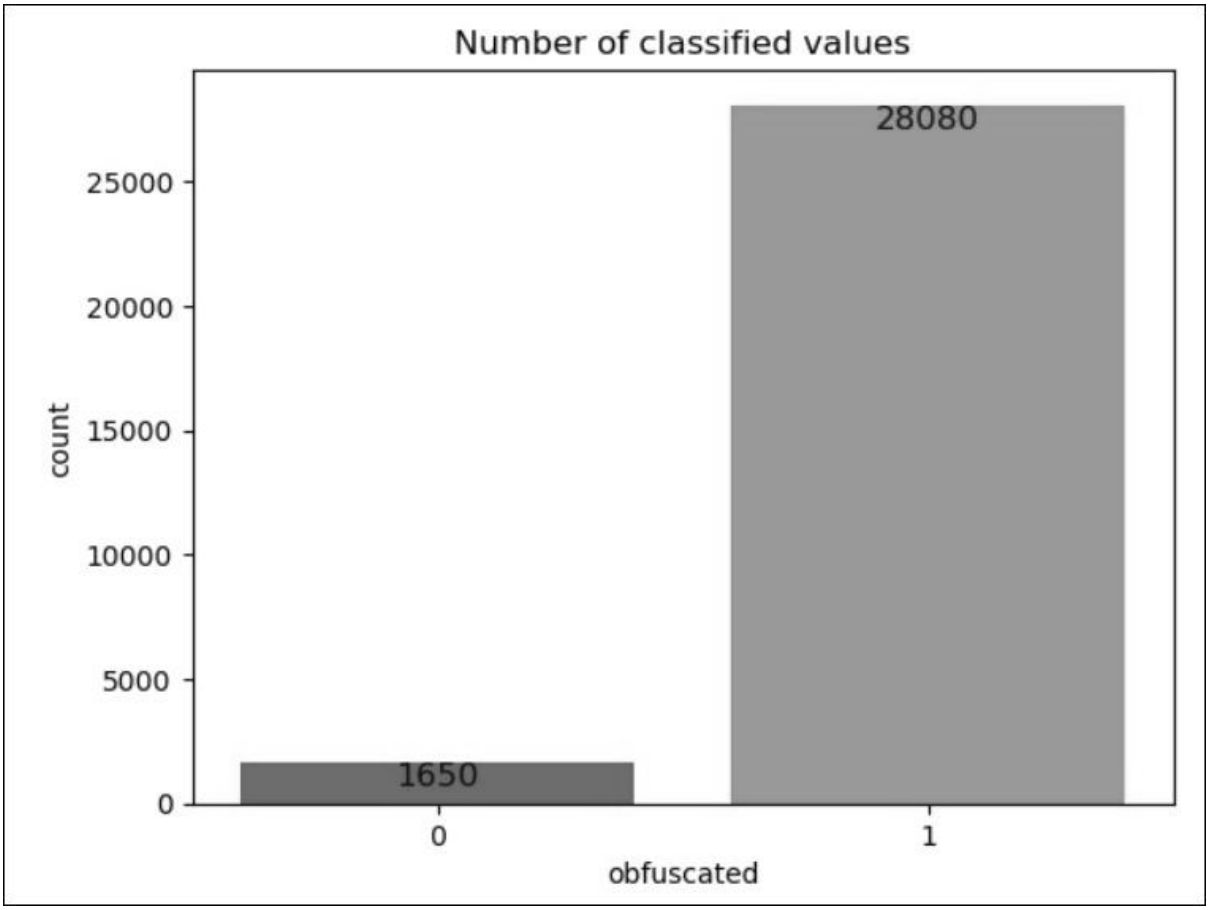


Рисунок 24 – Распределение целевого признака до предобработки

Показатель Imbalance Ratio (IR) составил 0,0588. Таким образом, число необфусцированных команд составляло около 6% от обфусцированных.

Для обучения моделей полученный набор данных необходимо предварительно обработать. Были проделаны следующие этапы предобработки:

1) очистка и преобразование данных;
2) трансформация признаков;
3) отбор признаков;
4) балансировка классов.

На этапе очистки и преобразования данных значения признаков были проверены на наличие пропусков, таких случаев не оказалось. Все признаки в наборе данных являлись количественными. Кроме того, 2 290 признаков

из 4 998 содержали единственное уникальное значение и были удалены. Полученный набор данных включал в себя 2 708 признаков, не включая целевой.

На этапе трансформации признаков были исследованы их распределения, корреляция с целевым признаком и воздействие на него.

Пример распределений первых 35 признаков представлен на рисунке 25.

Рисунок 25 – Распределение первых 35 признаков

Большинство признаков имеют полиномиальное распределение, в основном распределение Бернулли. Только у нескольких признаков распределение похоже на нормальное. Таким образом, для нормализации признаков использовался `MinMaxScaler` из библиотеки scikit-learn.

Корреляция с целевым признаком показала, что существуют не коррелирующие признаки, но также есть признаки, значение корреляции для которых достигает значения 0,3 по модулю.

На этапе отбора признаков были рассмотрены методы фильтрации (Filter Methods), методы обертки (Wrapper Methods) и встроенные методы (Embedded Methods) из библиотеки scikit-learn.

С помощью методы главных компонент (Principal Component Analysis – PCA) было установлено, что для пороговых значений дисперсии, равных 0,95 и 0,99 (95% и 99%), достаточно 138 и 513 признаков из исходных 2 708. Результаты данного измерения представлены на рисунке 26.

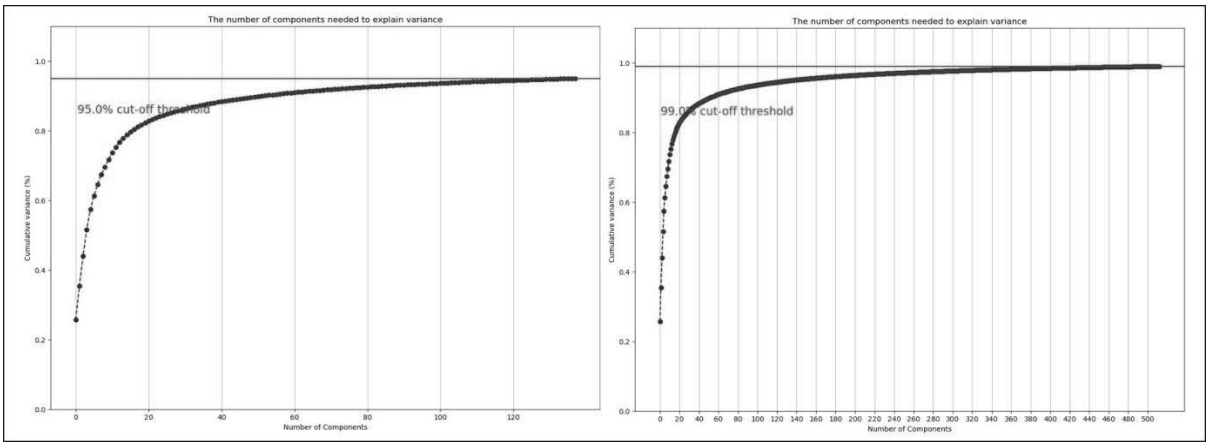


Рисунок 26 – Число признаков, покрывающих заданные пороговые значения дисперсии

С помощью методов фильтрации `SelectKBest` и `SelectPercentile` и моделей оценки `chi2`, `f_classif` и `mutual_info_classif` были выделены 138 и 513 самых значимых признаков для каждого метода с соответствующей моделью.

Для этого была реализована функция `select_features`, представленная в листинге 2, которая для настроенного входного метода отбора признаков определяет необходимые признаки из набора данных.

Листинг 2 – Функция выделения из набора данных подмножества с заданными числом признаков

```
def select_features(selector, X: pd.DataFrame, y: pd.Series) -> pd.Data-
Frame:
    selector.fit(X, y)
    selected_columns = X.columns[selector.get_support()]
    selected_features: pd.DataFrame = X[selected_columns]
    return selected_features
```

Основываясь на полученных результатах, были определены самые значимые признаки в наборе данных, учитывая результат каждого метода

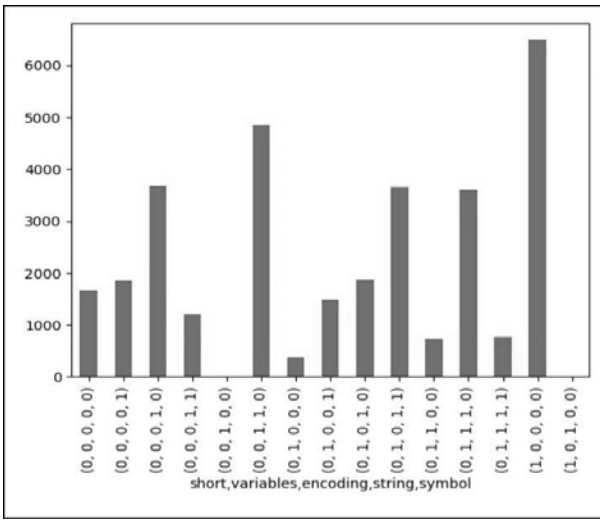
фильтрации: для каждого признака из всего набора данных было подсчитано число методов с моделями, которые определили его как значимый. Результат для первых 10 признаков представлен на рисунке 27.

```
Признак 'Method_with_parametersAstGroupedAstTypes_CommandAst_Count': количество методов = 12.
Признак 'Method_with_parametersAstGroupedAstTypes_CommandExpressionAst_Count': количество методов = 12.
Признак 'Method_with_parametersAstGroupedAstTypes_NamedBlockAst_Count': количество методов = 12.
Признак 'Method_with_parametersAstGroupedAstTypes_ScriptBlockAst_Count': количество методов = 12.
Признак 'Method_with_parametersAstCmdletMetrics_CharacterDistribution_:_3a_Count': количество методов = 12.
Признак 'Method_with_parametersAstCmdletMetrics_CharacterDistribution_/_2f_Percent': количество методов = 12.
Признак 'Method_with_parametersAstCmdletMetrics_CharacterDistribution_\_5c_Count': количество методов = 12.
Признак 'Method_with_parametersAstCmdletMetrics_CharacterDistribution_2_32_Count': количество методов = 12.
Признак 'Method_with_parametersAstCmdletMetrics_CharacterDistribution_3_33_Count': количество методов = 12.
Признак 'Method_with_parametersAstCmdletMetrics_CharacterDistribution_t_74_Count': количество методов = 12.
```

Рисунок 27 – Первые 10 самых значимых признаков в наборе данных

Для решения проблемы несбалансированности набора данных были использованы алгоритмы сэмплирования (undersampling и oversampling) из библиотеки imblearn.

Перед их применение были оценено распределение объектов по комбинациям примененных к ним техник обфускации, которое представлено на рисунке 28.



short	variables	encoding	string	symbol	
0	0	0	0	0	1665
				1	1853
			1	0	3670
				1	1208
		1	0	0	2
			1	0	4847
	1	0	0	0	384
				1	1491
			1	0	1878
				1	3649
		1	0	0	719
			1	0	3601
				1	758
1	0	0	0	0	6484
		1	0	0	1

Рисунок 28 – Распределение объектов по комбинациям примененных техник обфускации

Перед использованием методов сэмплирования был установлен лимит в 2 000 экземпляров для каждой комбинации.

Пример сравнения визуализации на плоскости объектов исходного набора данных и полученного в результате применения undersampling метода InstanceHardnessThreshold представлен на рисунке 29. Слева изображены объекты исходного набора данных, а справа после применения метода сэмплирования.

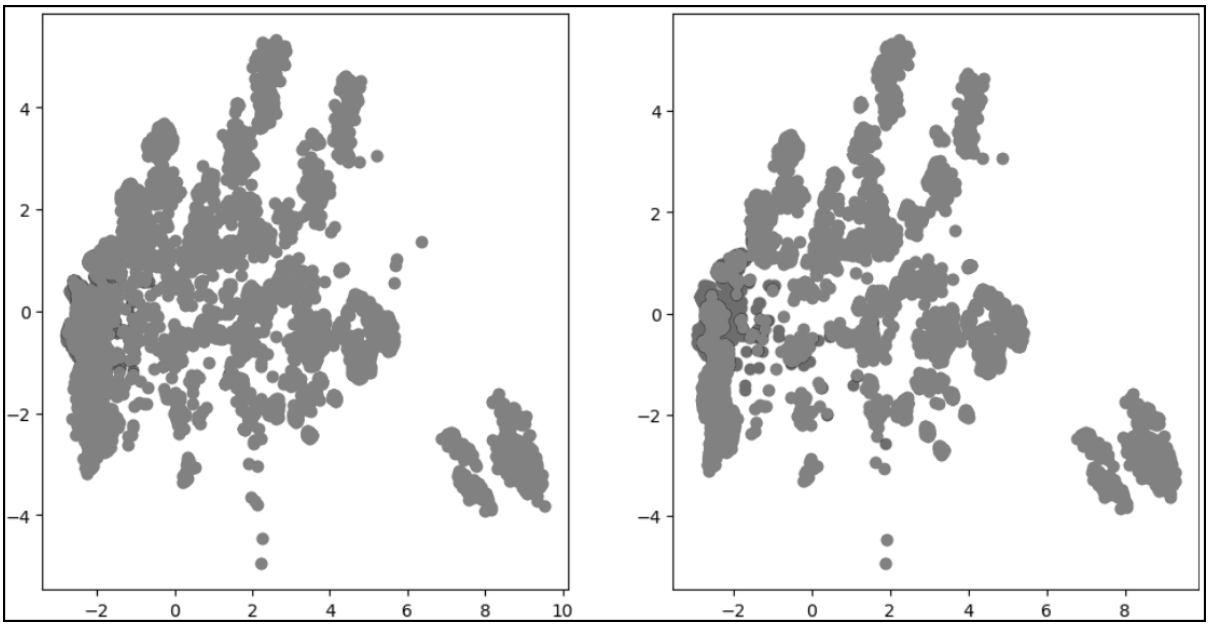


Рисунок 29 – Набор данных до и после применения метода сэмплирования

Зеленым цветом выделены необфусцированные объекты, а красным – обфусцированные.

Из второго графика видно, что метод сэмплирования не справляется с задачей балансировки данных и не разделяет объекты по кластерам, тем самым не создает между объектами разных классов гиперплоскость, которая будет явно сигнализировать о принадлежности объекта к конкретному классу. Данная проблема была замечена после применения каждого метода сэмлирования.

По этой причине было решено балансировать данные вручную.

Первым шагом являлось удаление из набора данных команд, содержащих использование оболочки cmd. Таких команд оказалось 5 605. Обновленный набор данных содержал 24 125 объектов. Распределение комбинаций примененных техник обфускации для команд, использующих cmd оболочку, представлено на рисунке 30. Подразумевается следующий порядок техник обфускации: short, variables, encoding, string и symbol. Значение 1 означает применение техники обфускации, а 0 – отсутствие.

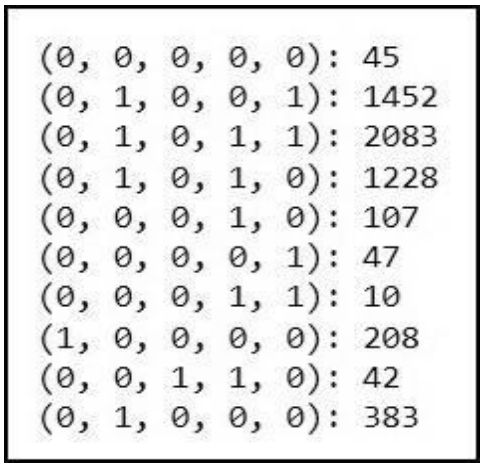


Рисунок 30 – Распределение комбинаций примененных техник обфускации к командам, использующим оболочку cmd

Визуализация команд на плоскости по примененным к ним комбинациям техник обфускации показала, что необфусцированные команды расположены очень близко с обфусцированными, к которым применена только техника обфускации short, или перекрываются ими.

Таким образом, было решено сделать обфусцированные команды с единственной использованной техникой обфускации short необфусцированными. Обновленных необфусцированных команд стало 7 881, а оставшихся обфусцированных 16 244.

Так как распределение команд по примененным к ним техникам обфускации неравномерно, то был поставлен лимит в 770 команд на каждую комбинацию техник обфускации. В итоге получилось 5 364 обфусцированные команды.

Чтобы уравнять число чистых команд, было уменьшено число добавленных обновленных необфусцированных команд, для которых была применена техника обфускации short, с 6 276 до 3 759.

В результате было получено 5 364 необфусцированные команды и полученный набор данных стал сбалансирован. Распределение целевого признака в предобработанном наборе данных представлено на рисунке 31.

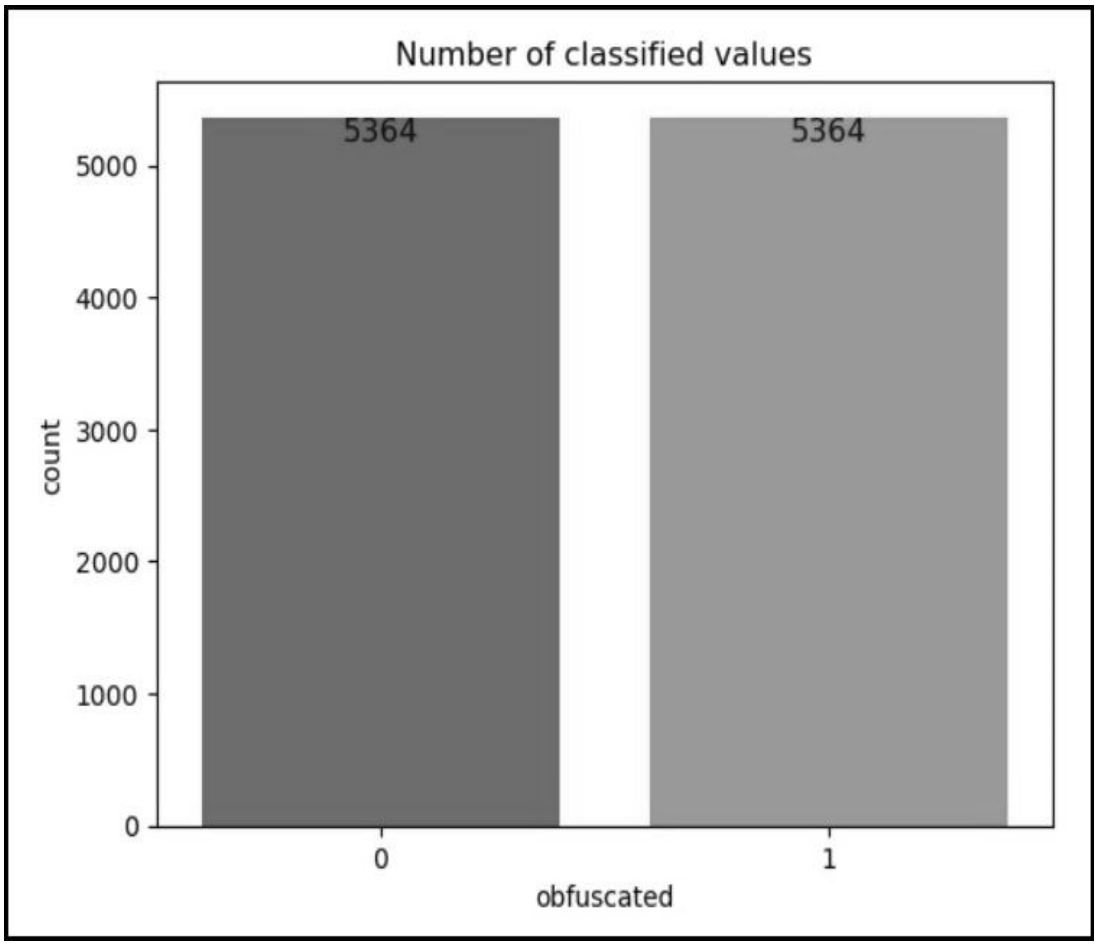


Рисунок 31 – Распределение целевого признака после предобработки

Предобработанный набор данных содержал 10 728 объектов и 2 708 параметров, а также целевой признак «obfuscated».

Чтобы соблюдать пайплайн обучения моделей, предобработанный набор данных был поделен на 3 подвыборки – train, test и validate – для обучения, тестирования и валидации, в объеме 60/20/20 от всего набора данных.

3.3. Реализация и обучение алгоритмов машинного обучения

Для обучения выбрано 10 алгоритмов машинного обучения.

Обучение основано на переборе различных комбинаций гиперпараметров с помощью алгоритма GridSearchCV из библиотеки scikit-learn с целью выявления заданного или максимально возможного числа комбинаций, на которых обученные модели показывают наилучшие результаты.

Затем на полученных комбинациях гиперпараметров модели обучаются заново и среди них выбирается одна, показавшая наилучшие результаты по другим метрикам (accuracy, precision и recall) на валидационном наборе данных.

Весь пайплайн обучения представлен в листинге 3.

Листинг 3 – Пайплайн обучения моделей

```
def process_model(model_class, hyperparameters: list[dict[str, Iterable]],
filename_to_save: str, models_to_validation: int = 5) -> None:
    learning_pipeline = ModelLearningPipeline(
        model_class=model_class,
        hyperparameters=hyperparameters,
        models_to_validation=min(models_to_validation, calculate_hy-
perparams_combinations_quantity(hyperparams_list=hyperparameters))
    )

    initial_scores: np.ndarray[float] = learning_pipeline.cross_vali-
date_initially()
    print(f"Scores from initial cross validation: {initial_scores},\nmean
value = {initial_scores.mean()}.")

    learning_pipeline.tuning()
    learning_pipeline.test()
    learning_pipeline.validate()

    learning_pipeline.save(pkl_filename=filename_to_save)
```

Исходный код класса ModelLearningPipeline, содержащего всю логику обучения, и класса MetricAnalyzer, в котором считаются и выводятся различные метрики, представлен в листингах 1 и 2 приложения А.

Исходные гиперпараметры, среди которых выбирались наилучшие комбинации, и набор гиперпараметров, на котором модель показала наилучшие результаты метрик, для каждого алгоритма представлены в таблице 10.

Таблица 10 – Исходные гиперпараметры и лучший набор гиперпараметров для каждого алгоритма

Алгоритм	Исходные гиперпараметры	Лучший набор гиперпараметров
Multinomial Naïve Bayes	alpha=[0.00001, 0.0001, 0.001, 0.1, 1, 10, 100, 1000]	alpha=1e-05
Gaussian Naïve Bayes	var_smoothing=np.logspace(0, -9, num=10)	var_smoothing=1e-06
Support Vector Machines	C=np.logspace(-3, 3, 7), kernel=[«linear», «rbf», «poly»], gamma=[«scale», np.linspace(0.1, 0.9, 9), np.linspace(0.01, 0.05, 5)], degree=[2, 3, 4]	kernel=«linear», C=0.1, degree=3, gamma=«scale»
K-Nearest Neighbors	n_neighbors=[5, 7, 9, 11, 13, 15], weights=[«uniform», «distance»], metric=[«minkowski», «euclidean», «manhattan»]	metric=«minkowski», n_neighbors=5, weights=«uniform»
Logistic Regression	solver=[«lbfgs», «sag», «liblinear»], penalty=[«l1», «l2»], C=np.logspace(-3, 3, 7), multi_class=[«ovr»], max_iter=[1000]	solver=«liblinear», penalty=«l1», C=0.1, multi_class=«ovr», max_iter=1000
Decision Tree	criterion=[«gini», «entropy»], max_depth=[None, np.linspace(10, 90, 9)], min_samples_leaf=[1, 2, 4], min_samples_split=[2, 5, 10]	criterion=«entropy», max_depth=20, min_samples_leaf=4, min_samples_split=2
Random Forest	n_estimators=[10, np.linspace(100, 500, 5), max_depth=[None, np.linspace(10, 90, 9)], min_samples_leaf=[1, 2, 4], min_samples_split=[2, 5, 10]	n_estimators=100, max_depth=80, min_samples_leaf=2, min_samples_split=10
XGBoost	n_estimators=[50, 100, 200, 500], max_depth=[3, 5, 7, 9], learning_rate=[0.01, 0.05, 0.15, 0.3]	n_estimators=500, max_depth=9, learning_rate=0.01
CatBoost	iterations=[100, 200, 500, 1000], learning_rate=[0.01, 0.05, 0.15, 0.3], depth=[4, 6, 8, 10]	iterations=100, learning_rate=0.05, depth=6
LightGBM	num_leaves=[31, 50, 100, 150], max_depth=[3, 5, 7, 9], learning_rate=[0.01, 0.05, 0.15, 0.3], n_estimators=[50, 100, 200, 500]	num_leaves=31, learning_rate=0.01, max_depth=5, n_estimators=50

Качество обученных моделей на различных наборах данных (для обучения, тестирования и валидации) после 5-фолдовой кросс-валидации представлено в таблицах 11–13. Для этого были использованы метрики accuracy, recall, precision и f1.

Таблица 11 – Качество моделей на обучающей выборке

Алгоритм	Accuracy	Recall	Precision	F1
Multinomial Naïve Bayes	0,91749	0,84524	0,98875	0,91130
Gaussian Naïve Bayes	0,98570	0,98421	0,98736	0,98574
Support Vector Machines	0,99285	0,99071	**0,99505**	0,99286
K-Nearest Neighbors	0,97032	0,96936	0,97152	0,97041
Logistic Regression	0,99114	0,98855	0,99380	0,99115
Decision Tree	0,99160	0,99071	0,99255	0,99101
Random Forest	0,98881	0,98916	0,98951	0,98965
XGBoost	**0,99300**	**0,99288**	0,99319	**0,99303**
CatBoost	**0,99300**	0,99195	0,99411	**0,99303**
LightGBM	0,99285	0,99257	0,99320	0,99287

Таблица 12 – Качество моделей на тестовой выборке

Алгоритм	Accuracy	Recall	Precision	F1
Multinomial Naïve Bayes	0,90353	0,81773	0,98421	0,89317
Gaussian Naïve Bayes	0,98322	0,98394	0,98215	0,98302
Support Vector Machines	0,98788	0,98206	0,99333	0,98765
K-Nearest Neighbors	0,95713	0,95279	0,96021	0,95639
Logistic Regression	0,98369	0,97827	0,98861	0,98336
Decision Tree	0,98742	0,98489	0,98954	0,98675
Random Forest	0,98602	0,98016	0,99047	0,98524
XGBoost	**0,99208**	**0,99150**	0,99245	**0,99196**
CatBoost	**0,99208**	0,98961	**0,99430**	0,99194
LightGBM	0,99068	0,99055	0,99060	0,99055

Таблица 13 – Качество моделей на валидационной выборке

Алгоритм	Accuracy	Recall	Precision	F1
Multinomial Naïve Bayes	0,90820	0,82958	0,98455	0,90035
Gaussian Naïve Bayes	0,98415	0,98882	0,97977	0,98424
Support Vector Machines	0,98462	0,97671	0,99245	0,98449
K-Nearest Neighbors	0,96225	0,95620	0,96828	0,96191
Logistic Regression	0,97437	0,95995	0,98851	0,97398
Decision Tree	0,99114	0,98881	**0,99349**	0,99066
Random Forest	0,98368	0,98229	0,99163	0,98642
XGBoost	**0,99254**	**0,99254**	0,99257	**0,99254**
CatBoost	0,99021	0,98881	0,99160	0,99019
LightGBM	0,98835	0,98695	0,98972	0,98832

Из результатов таблиц можно заключить, что выбранные алгоритмы машинного обучения эффективно справляются с поставленной задачей.

3.4. Реализация серверной части

Серверная часть приложения включает в себя 6 основных компонентов:

1) NoSQL база данных MongoDB, в которой хранится информация об основных сущностях;

2) объектное хранилище MinIO, необходимое для хранения файлов;

3) HTTP API-сервис, реализованный с помощью веб-фреймворка FastAPI;

4) инициализатор данных, который загружает исходный предобработанный набор данных и обученные модели в MinIO и MongoDB;

5) асинхронная очередь задач Celery, необходимая для распределенной обработки задач вне основной программы;

6) NoSQL база данных Redis, используемая Celery для обработки и хранения информации о задачах и результатах их выполнения.

База данных MongoDB включает в себя следующие 4 коллекции:

1) коллекция Datasets, необходимая для хранения документов с информацией о доступных предобработанных наборах данных;

2) коллекция Models, необходимая для хранения документов с информацией о доступных обученных моделях;

3) коллекция Classifications, необходимая для хранения документов с информацией о выполненных классификациях команд, содержащей результат классификации одной отдельно взятой выбранной обученной модели из доступных;

4) коллекция CommonClassifications, необходимая для хранения документов с информацией о выполненных классификациях команд, содержащей совокупный результат классификаций всех выбранных обученных моделей.

Коллекции базы данных представлены на рисунке 32.

Рисунок 32 – Список коллекций базы данных MongoDB

Объектное хранилище MinIO включает в себя 2 bucket-а:

1) datasets, который содержит уникальные файлы предобработанных наборов данных;

2) models, который содержит уникальные файлы обученных моделей.

Доступные bucket-ы объектного хранилища MinIO представлены на рисунке 33.

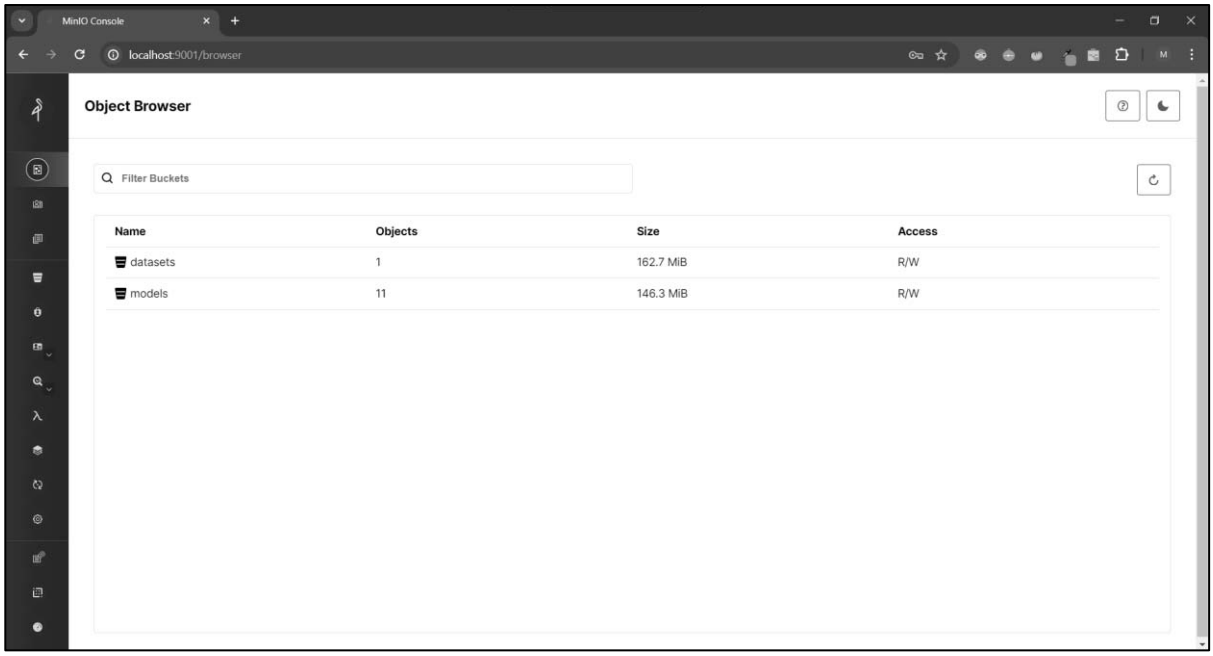


Рисунок 33 – Список bucket-ов объектного хранилища MinIO

Уникальность файлов достигается через их хеш-суммы.

Инициализатор представляет собой цепочку из двух задач – загрузка набора данных и загрузка обученных моделей, которые запускаются последовательно с помощью возможностей Celery в момент, когда обработчик задач настроен и готов к работе. Набор данных, обученные модели и информация о них, необходимая для базы данных, хранятся локально. Последовательность задач необходима, чтобы получить id загруженного набора данных в базе данных и указать его в виде внешнего ключа в документах моделей.

FastAPI приложение включает в себя 13 роутов, имеющих общий префикс /api/v1 в url. Информация о них представлена в таблице 1 приложения Б.

Для получения информации об обученных моделях и классификациях были добавлены Data Transfer Object (DTO) модели и дополнительная логика компоновки данных, так как схемы документов, хранящихся в базе данных, не соответствуют схемам, которые ожидает клиентская часть приложения.

Celery использует 1 обработчик (воркер) и 1 очередь задач по умолчанию.

Для мониторинга задач, отслеживания прогресса их выполнения и управления обработчиками используется веб-приложение с открытым исходным кодом Flower.

Celery позволяет задавать гибкие настройки задачам.

Благодаря этой возможности были реализованы распределенные блокировки следующих задач:

1) загрузка наборов данных – файлы, содержащие одинаковую хеш-сумму, не могут загружаться одновременно;

2) обучение моделей машинного обучения – модели с одинаковыми алгоритмами, параметрами обучения и наборами данных не могут обучаться одновременно;

3) удаление наборов данных и обученных моделей – из-за каскадного удаления данные должны удаляться изолировано, чтобы избежать конфликтов и неожиданного поведения при одновременном удалении и изменении разделяемых ресурсов.

Пример распределенной блокировки для загрузки наборов данных представлен в листинге 4, содержащем реализацию класса задачи `UploadingDatasetTask`, унаследованного от базового класса `Task` из celery.

Каскадное удаление для наборов данных подразумевает, что удаление объекта родительской сущности ведет к его удалению в связанных объектах дочерних сущностей. Каскадное удаление обученной модели приводит к удалению классификаций, основанных на этой модели, если они есть, а также к пересчету результата совокупной классификации каждой команды, для которой была классификация этой модели.

Кроме того, когда запускается классификация загруженных команд на выбранных обученных моделях, все предыдущие классификации удаляются из базы данных.

Также добавлено логирование, реализованное с помощью встроенной библиотеки logging. Пример логирования представлен в листинге 4.

Листинг 4 – Реализация класса задачи для загрузки наборов данных

```
class UploadingDatasetTask(Task):
    redis: Redis
    locked_task_expiration: int

    def before_start(self, task_id, args, kwargs) -> None:
        logger.info(f"Start calculating md5 hash for dataset {kwargs['file-
name']!r}")
        md5 = core_service.calculate_md5(
            file_data=kwargs["file_data"],
            chunk_size=global_config.minio.part_size
        )
        logger.info(f"Md5 hash for dataset {kwargs['filename']!r}:
{md5!r}")
        locking_status = self.redis.set(md5, 'lock',
ex=self.locked_task_expiration, nx=True)
        if not locking_status:
            logger.error(f"Uploading dataset {kwargs['filename']!r} has al-
ready locked by another task")
            raise LockException()
        setattr(self, 'md5', md5)

    def on_success(self, retval, task_id, args, kwargs) -> None:
        self.redis.delete(self.md5)
```

```
    def on_failure(self, exc, task_id, args, kwargs, einfo) -> None:
        if isinstance(exc, LockException):
            return
        self.redis.delete(self.md5)
```

Метод `before_start` выполняется до запуска задачи, для которой объявленный класс будет базовым, `on_success` выполняется при успешном выполнении задачи, а `on_failure` при неудачном.

3.5. Реализация клиентской части

Клиентская часть приложения реализована на языке программирования TypeScript с использованием библиотеки React и библиотеки компонентов R-UI.

Для выполнения запросов к серверной части приложения использовались библиотеки axios и react-query. Пример запроса представлен в листинге 5.

Листинг 5 – Запрос информации о доступных наборах данных с сервера

```
async getDatasets(page: number, size: number) {
    return await axios.get<IPageData<TDataset>>(
`${process.env.REACT_APP_API_URL}/api/v1/datasets/?page=${page}&size=${size
}`,
    );
}
```

Для инициализации приложения использовались компоненты R-UI, такие как `RuiConfig` и `AppContainer`, и `QueryClientProvider` из библиотеки react-query.

`RuiConfig` позволяет задать тему приложения, а `AppContainer` содержит в себе свойства `sidebar` и `header`.

Для настройки маршрутизации была использована библиотека react-router-dom.

Для работы с формами использовалась библиотека react-hook-form, а также компоненты и хуки из R-UI. Поля и кнопки были основаны на готовых компонентах, но дополнены собственными стилями, свойствами и обработчиками.

Для работы с таблицами и панелями управления использовались компоненты `ActionPanel` и `Table`. Таблица включает в себя пагинацию и позволяет переключаться между разными страницами данных, а также изменять размер данных на одной странице.

Для взаимодействия с состоянием компонентов использовались хуки, такие как, `useState`, `useEffect` и `useCallback`.

Для генерации форм из JSON-схем параметров обучения алгоритмов была написана функция, которая выбирает из схемы переданные параметры, их типы, возможные значения и значения по умолчанию, и кладет их в компоненты.

Ниже представлены основные страницы приложения.

Главная страница представлена на рисунке 34.

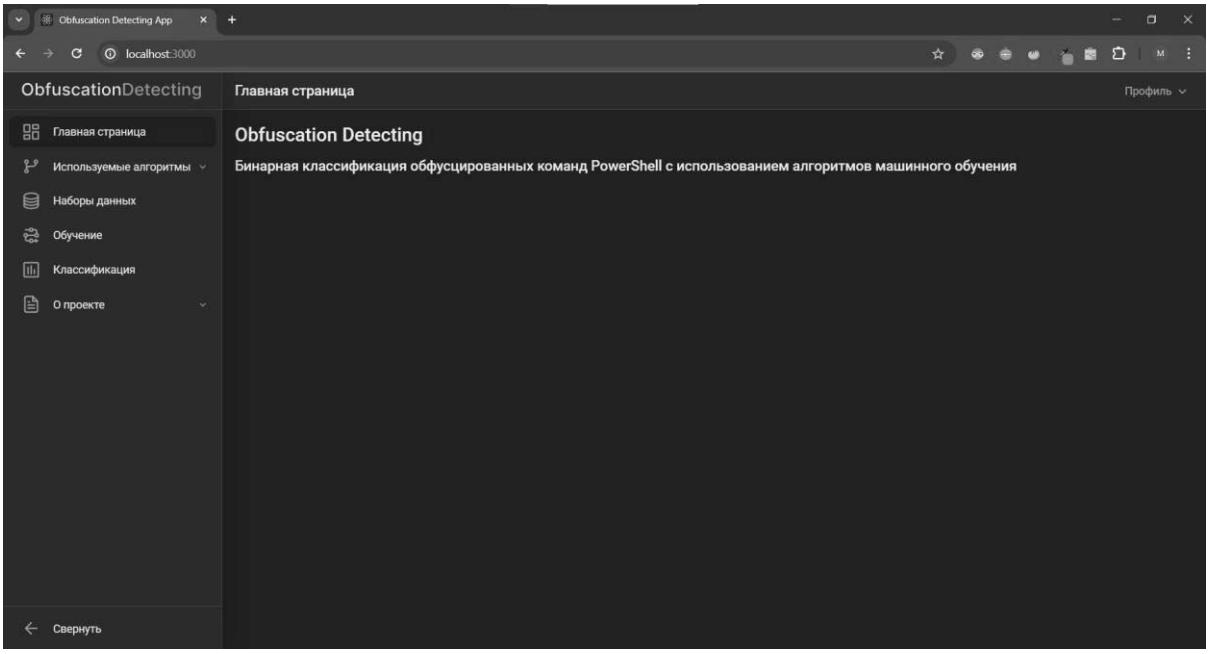


Рисунок 34 – Главная страница веб-приложения

Пример страницы с теорией используемого алгоритма представлен на рисунке 35.

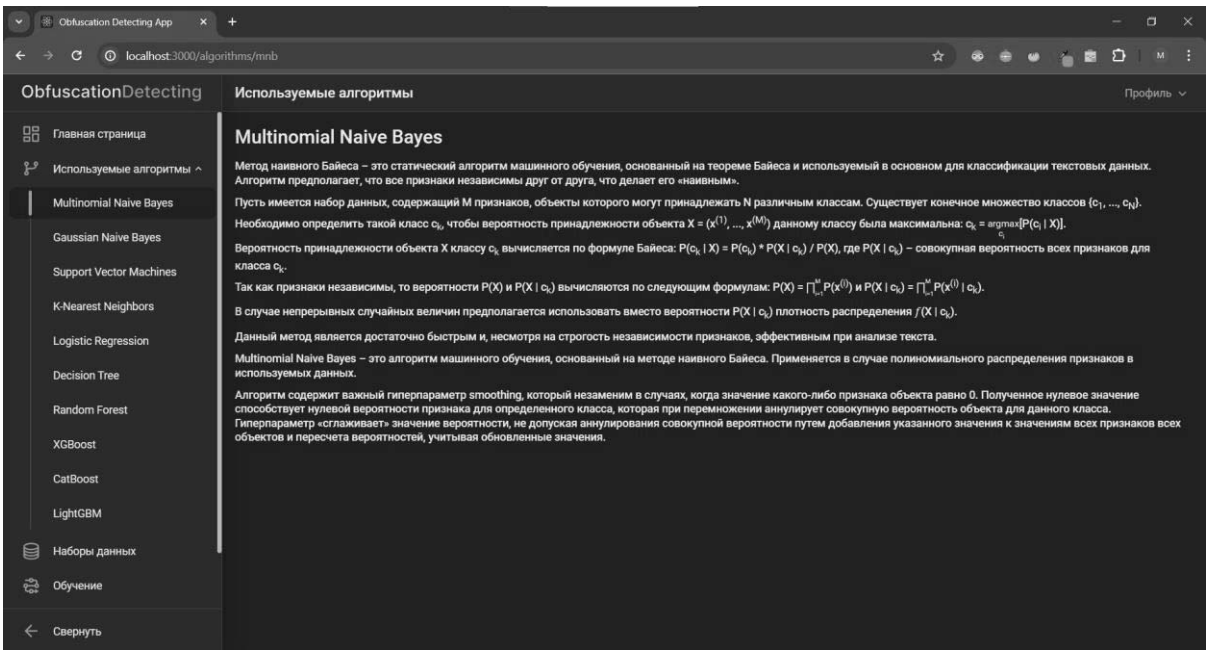


Рисунок 35 – Страница с теорией алгоритма Multinomial Naive Bayes

Страница с наборами данных представлена на рисунке 36.

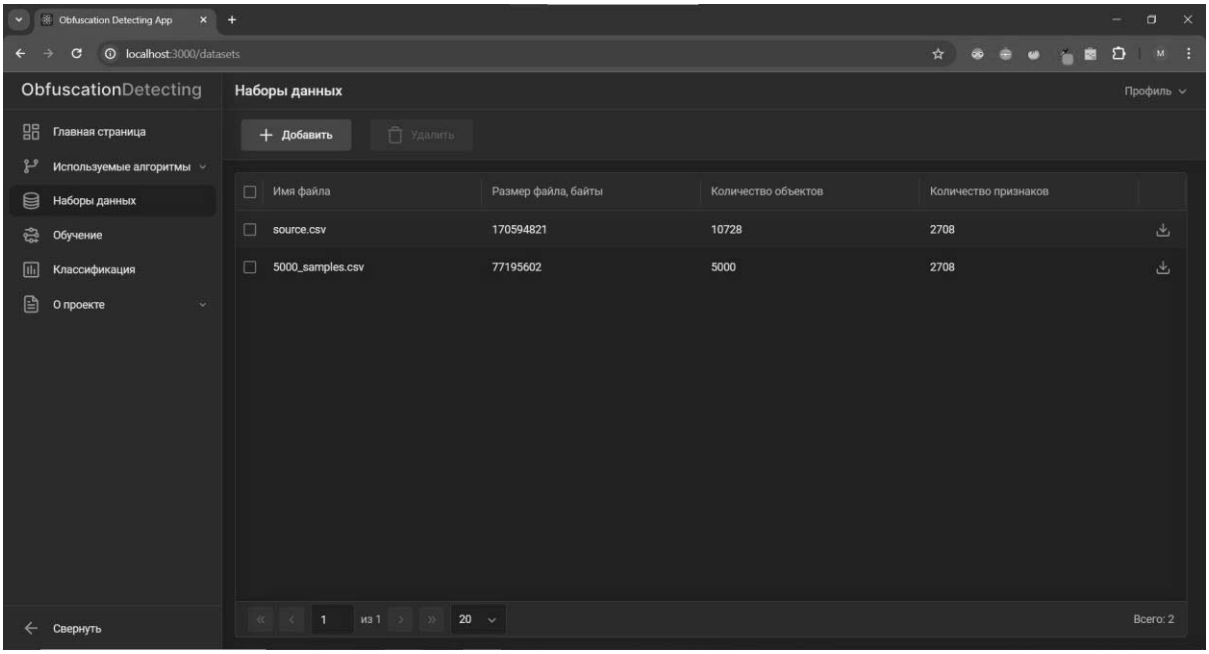


Рисунок 36 – Страница с наборами данных

Страница с обученными моделями представлена на рисунке 37.

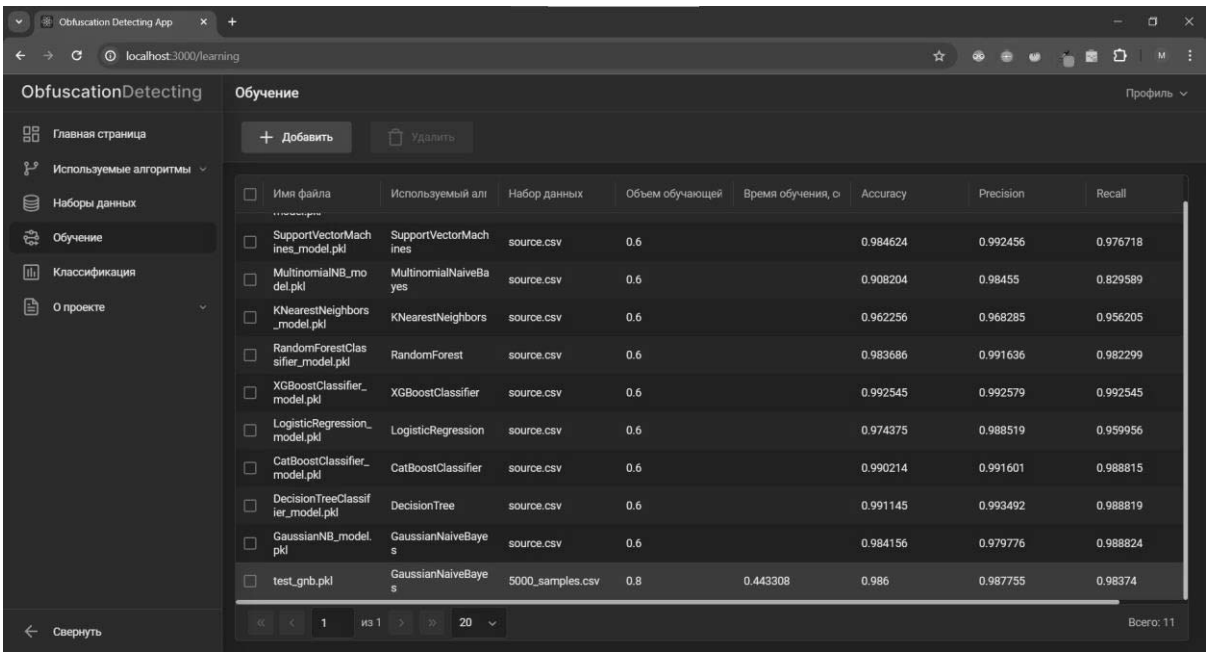


Рисунок 37 – Страница с обученными моделями

Страница с результатами классификации представлена на рисунке 38.

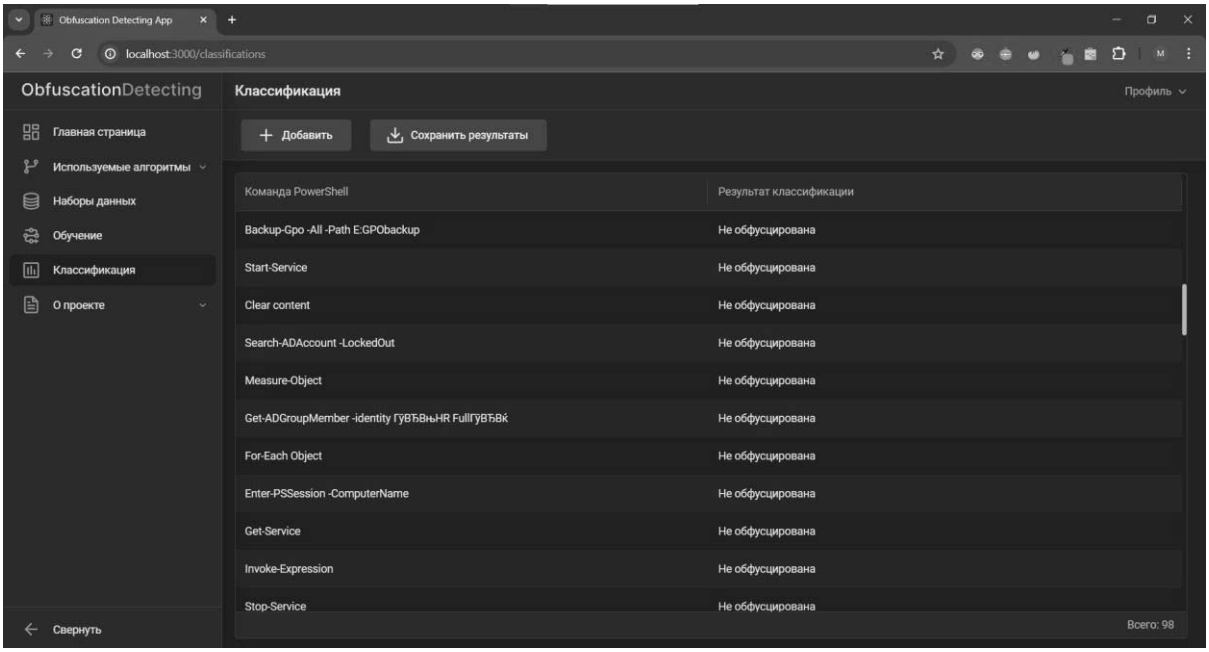


Рисунок 38 – Страница с результатами классификации

Пример страницы с контактами представлен на рисунке 39.

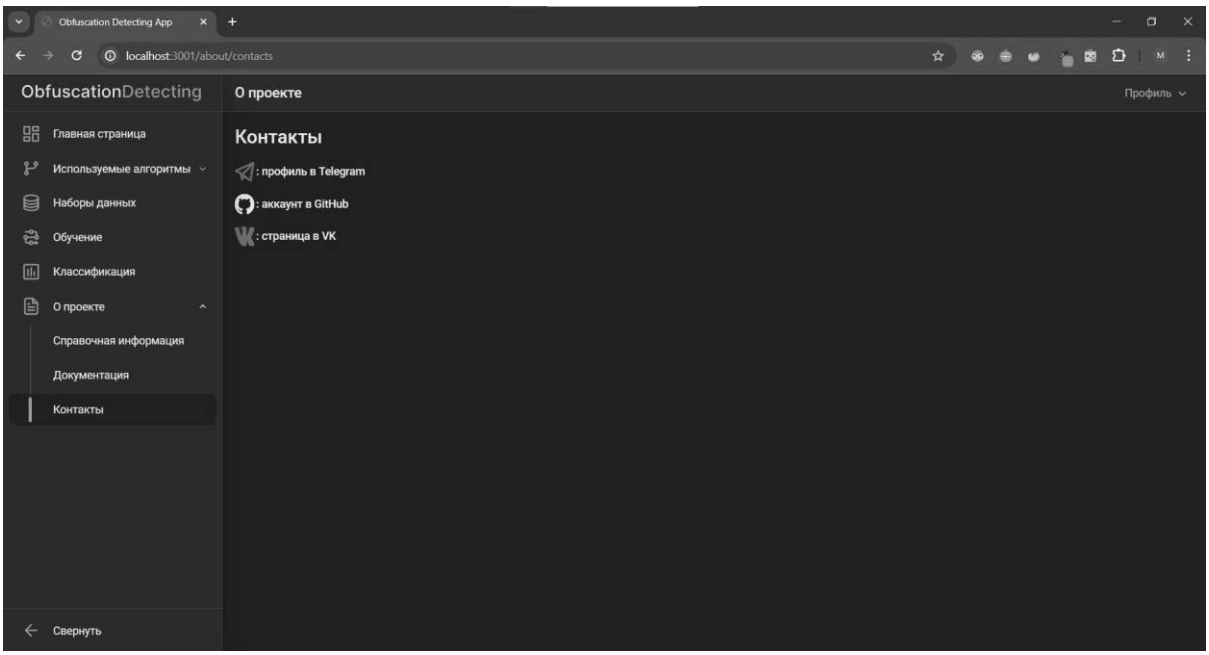


Рисунок 39 – Страница с контактами

3.6. Развертывание веб-приложения

Для развертывания веб-приложения использовался Docker.

Чтобы развернуть MinIO, Redis, MongoDB и Flower использовались готовые образы с DockerHub.

Для автоматизированного создания необходимых bucket-ов в MinIO используется отдельный сервис, использующий публичный образ minio/mc из DockerHub и UNIX команды.

Для запуска серверной части приложения, обработчика celery и клиентской части приложения нужно было создать собственные Docker файлы. Для серверной части приложения и обработчика celery это общий образ, базовым для которого являлся Python образ. Для клиентской части приложения базовым образом являлся образ node.

Для обратного проксирования и балансировки трафика между сервисами использовался Traefik, образ для которого был взят с DockerHub.

В результате в файле «docker-compose.yml» было объявлено 9 сервисов. Для сервисов были определены их базовые образы, пути и контекст для сборки образов, если их не существует, политики скачивания образов с

DockerHub, порты и переменные среды, а также зависимости от других сервисов и volume для хранения данных.

Запущенные в Docker Desktop контейнеры представлены на рисунке 40.

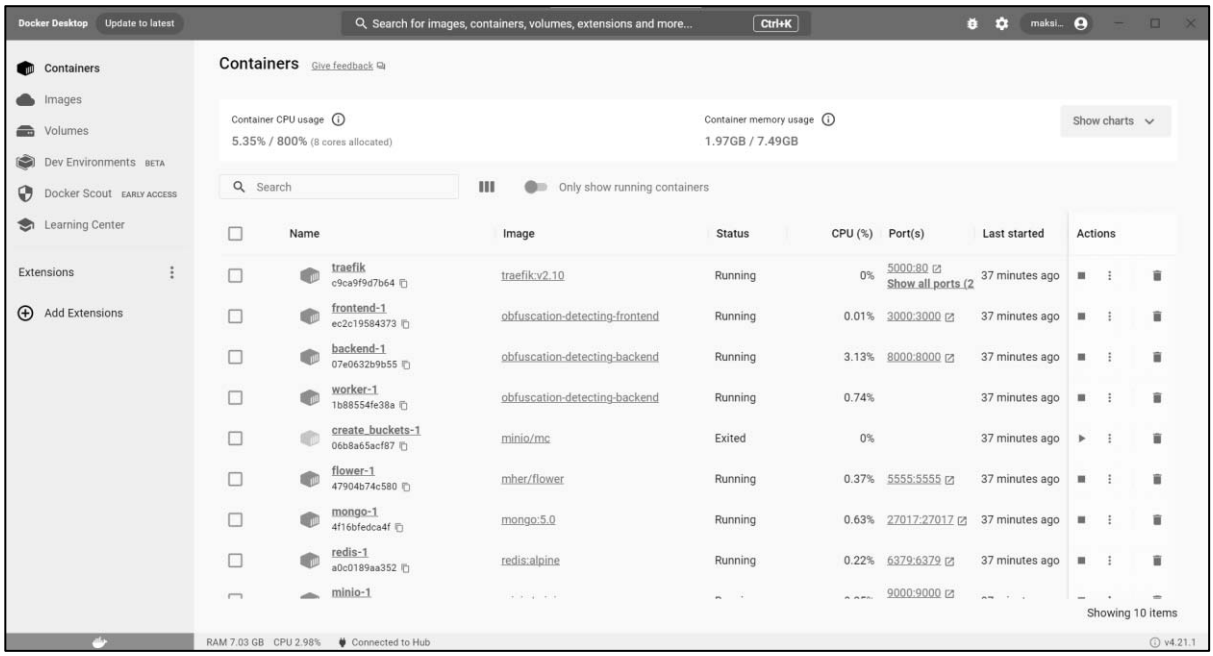


Рисунок 40 – Запущенные контейнеры в Docker Desktop

Выводы по третьей главе

В третьей главе были определены программные средства разработки и используемые технологии, описаны все этапы предобработки данных и обучения алгоритмов, представлены реализации серверной и клиентских частей приложения, а также его развертывание с помощью Docker.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование алгоритмов машинного обучения

В данном пункте представлены этапы тестирования реализованных моделей машинного обучения с учителем: A/B тестирование и тестирование на нескольких дополнительных наборах данных.

A/B тестирование

A/B тестирование в машинном обучении используется для сравнения эффективности двух или более вариантов модели с разными значениями гиперпараметров.

Исходный предобработанный набор данных был поделен на 2 части: обучающая и тестовая выборки, составляющие 60% и 40% от объема всего набора данных.

Для сравнения результатов моделей использовалась метрика accuracy. Чтобы полученные значения были корректнее, была использована 5-фолдовая кросс-валидация для усреднения значения метрики.

Результаты тестирования представлены в таблицах 14-23.

Таблица 14 – Тестирование модели Multinomial Naïve Bayes

Набор гиперпараметров	Accuracy
alpha=1e-05	0,90587418
alpha=0,001	0,90400937
alpha=0,1	0,90400910

Таблица 15 – Тестирование модели Gaussian Naïve Bayes

Набор гиперпараметров	Accuracy
var_smoothing=1e-09	0,98485608
var_smoothing=1e-06	0,98485554
var_smoothing=0,01	0,89934873

Таблица 16 – Тестирование модели Support Vector Machines

Набор гиперпараметров	Accuracy
C=0.1	0,92078635
C=1	0,98462271
C=10	0,98835014

Таблица 17 – Тестирование модели K-Nearest Neighbors

Набор гиперпараметров	Accuracy
n_neighbors=3	0,96039032
n_neighbors=5	0,96365373
n_neighbors=9	0,96388818

Таблица 18 – Тестирование модели Logistic Regression

Набор гиперпараметров	Accuracy
C=0.001	0,90680495
C=0.1	0,98019679
C=10	0,98858270

Таблица 19 – Тестирование модели Decision Tree

Набор гиперпараметров	Accuracy
criterion=«entropy»	0,98974738
criterion=«gini»	0,98788312
criterion=«log_loss»	0,99137854

Таблица 20 – Тестирование модели Random Forest

Набор гиперпараметров	Accuracy
n_estimators=100	0,98648669
n_estimators=300	0,98741882
n_estimators=500	0,98648642

Таблица 21 – Тестирование модели XGBoost

Набор гиперпараметров	Accuracy
n_estimators=100	0,99370846
n_estimators=300	0,99370846
n_estimators=500	0,99324280

Таблица 22 – Тестирование модели CatBoost

Набор гиперпараметров	Accuracy
learning_rate=0.01	0,99184366
learning_rate=0.05	0,99137773
learning_rate=0.15	0,99137800

Таблица 23 – Тестирование модели LightGBM

Набор гиперпараметров	Accuracy
n_estimators=25	0,99207811
n_estimators=50	0,99324280
n_estimators=100	0,99324253

В вышеперечисленных результатах существуют примеры моделей, которые имеют показатели метрики выше, чем на выбранных гиперпараметрах. Обусловлено это тем, что в обучении использовалась комбинация гиперпараметров и учитывалось не только среднее значение метрики, но и стандартное отклонение ее результатов для всех запусков перекрестной проверки с тем же набором гиперпараметров.

Тестирование на дополнительных наборах данных

Так как в предобработанном наборе данных, который применялся для обучения моделей, содержались не все команды, входящие в исходный набор данных, было решено оставшиеся команды использовать для тестирования обученных моделей. Среди них были команды, использующие cmd оболочку, и часть обфусцированных команд PowerShell, которые были исключены в процессе балансировки. На полученных наборах данных было оценено качество моделей с помощью метрик accuracy, recall, precision и f1. Результаты тестирования представлены в таблицах 24–25.

Таблица 24 – Качество моделей на наборе данных, содержащем команды Powershell, использующие cmd оболочку

Алгоритм	Accuracy	Recall	Precision	F1
Multinomial Naïve Bayes	0,44478	0,41854	1,00000	0,59009
Gaussian Naïve Bayes	0,92471	0,95123	0,96935	0,96020
Support Vector Machines	0,70972	0,69600	1,00000	0,82076
K-Nearest Neighbors	0,38965	0,36584	0,98640	0,53373
Logistic Regression	0,67029	0,65490	0,99971	0,79138
Decision Tree	0,58091	0,5611	1,00000	0,71885
Random Forest	0,66209	0,64611	1,00000	0,78502
XGBoost	0,67226	0,65676	1,00000	0,79283
CatBoost	0,67404	0,65863	1,00000	0,79419
LightGBM	0,68421	0,66928	1,00000	0,80188

Таблица 25 – Качество моделей на наборе данных, содержащем оставшиеся обфусцированные команды Powershell

Алгоритм	Accuracy	Recall	Precision	F1
Multinomial Naïve Bayes	0,89087	0,86590	0,99968	0,92799
Gaussian Naïve Bayes	0,98246	0,98456	0,99378	0,98915
Support Vector Machines	0,99448	0,99320	1,00000	0,99659
K-Nearest Neighbors	0,98238	0,97831	1,00000	0,98904

Окончание таблицы 25

Алгоритм	Accuracy	Recall	Precision	F1
Logistic Regression	0,99545	0,99439	1,00000	0,99719
Decision Tree	0,99298	0,99274	0,99861	0,99567
Random Forest	0,99746	0,99688	1,00000	0,99844
XGBoost	0,99627	0,99540	1,00000	0,99770
CatBoost	0,99694	0,99623	1,00000	0,99811
LightGBM	0,99433	0,99301	1,00000	0,99650

Также был сгенерирован набор данных, содержащий слабо обфусцированные команды. Такие команды были получены следующим образом: были взяты все необфусцированные команды из исходного набора данных, на каждые 20 символов в них был добавлен символ «`», игнорируемый интерпретатором PowerShell и сигнализирующий о наличии обфусцированности. Результаты тестирования представлены в таблице 26.

Таблица 26 – Качество моделей на наборе данных, содержащем слабо обфусцированные команды Powershell

Алгоритм	Accuracy	Recall	Precision	F1
Multinomial Naïve Bayes	0,03693	0,03693	1,00000	0,07123
Gaussian Naïve Bayes	0,96165	0,96165	1,00000	0,98045
Support Vector Machines	0,28054	0,28054	1,00000	0,43816
K-Nearest Neighbors	0,17116	0,17116	1,00000	0,29230
Logistic Regression	0,42543	0,42543	1,00000	0,59691
Decision Tree	0,93182	0,93182	1,00000	0,96471
Random Forest	0,75497	0,75497	1,00000	0,86038
XGBoost	0,93537	0,93537	1,00000	0,96661
CatBoost	0,97301	0,97301	1,00000	0,98632
LightGBM	0,93466	0,93466	1,00000	0,96623

В четвертый дополнительный набор данных входили команды, содержащие кириллические символы. Всего таких команд было 8, написаны они были собственноручно, так как подобные команды отсутствовали в исходном наборе данных. Результаты тестирования представлены в таблице 27.

Таблица 27 – Качество моделей на наборе данных, содержащем команды Powershell, в которых присутствуют кириллические символы

Алгоритм	Accuracy	Recall	Precision	F1
Multinomial Naïve Bayes	1,00000	0,00000	0,00000	0,00000
Gaussian Naïve Bayes	0,12500	0,00000	0,00000	0,00000
Support Vector Machines	0,25000	0,00000	0,00000	0,00000

Окончание таблицы 27

Алгоритм	Accuracy	Recall	Precision	F1
K-Nearest Neighbors	1,00000	0,00000	0,00000	0,00000
Logistic Regression	1,00000	0,00000	0,00000	0,00000
Decision Tree	1,00000	0,00000	0,00000	0,00000
Random Forest	0,37500	0,00000	0,00000	0,00000
XGBoost	1,00000	0,00000	0,00000	0,00000
CatBoost	0,50000	0,00000	0,00000	0,00000
LightGBM	1,00000	0,00000	0,00000	0,00000

Так как в наборе данных, содержащем команды PowerShell, в которых присутствовали кириллические символы, были только необфусцированные команды, метрики recall, precision и f1 были игнорированы.

Из приведенных выше результатов можно сделать вывод, что только модель Gaussian Naive Bayes показывает стабильные и хорошие результаты на всех дополнительных наборах данных.

Тестирование времени классификации

В данном пункте описано измерение времени классификации входных данных и тестирование его зависимости от их размера.

Для этого использовались дополнительные вредоносные обфусцированные команды, выданные сотрудниками компании R-Vision, которые перед классификацией были токенизированы и предобработаны.

Данные подавались частями, по 8 раз, размер каждой части варьировался от 250 до 2000 команд. Входные данные переиспользовались, если их было недостаточно. В среднем на каждые 250 команд пропускалось около 10 из-за ошибок токенизации.

Среднее время в секундах классификации входных данных для алгоритмов на разных размерах частей представлено в таблице 28.

Таблица 28 – Среднее время классификации входных данных на разных размерах частей

Алгоритм	250	500	750	1000	1250	1500	1750	2000
Multinomial Naïve Bayes	32,31	64,75	96,63	128,80	161,11	193,85	225,70	258,63
Gaussian Naïve Bayes	32,51	64,76	96,96	131,70	162,06	193,71	224,96	258,66

Окончание таблицы 28

Алгоритм	250	500	750	1000	1250	1500	1750	2000
Support Vector Machines	33,11	65,32	97,63	130,52	163,09	195,33	227,54	260,82
K-Nearest Neighbors	71,05	141,27	210,57	278,57	349,93	422,70	490,22	561,61
Logistic Regression	33,11	65,62	98,59	131,03	163,72	196,69	229,23	262,26
Decision Tree	32,84	65,63	98,25	130,97	163,64	196,59	228,77	262,09
Random Forest	37,17	74,68	111,59	148,72	186,04	223,33	259,67	297,03
XGBoost	49,08	98,03	147,16	194,88	246,14	291,69	341,60	389,44
CatBoost	81,13	132,62	197,92	263,59	330,00	396,57	460,45	528,92
LightGBM	40,81	82,64	121,74	161,44	202,35	241,90	284,58	324,75

Таким образом, наиболее быстрая классификация входных данных происходит у алгоритмов Multinomial Naïve Bayes, Gaussian Naïve Bayes, Support Vector Machines, Logistic Regression и Decision Tree. Самая долгая классификация у алгоритмов K-Nearest Neighbors и CatBoost.

На рисунках 41–50 представлена зависимость времени классификации входных данных от их размера для каждого алгоритма.

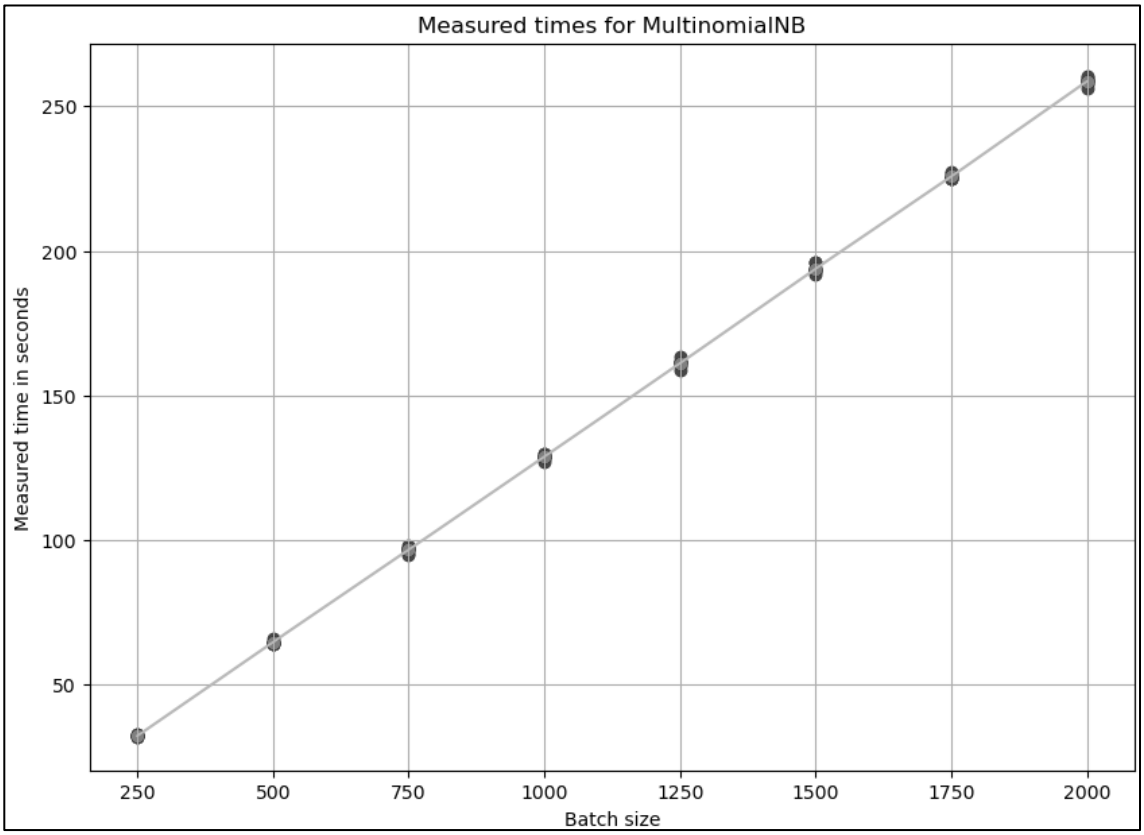


Рисунок 41 – Зависимость времени классификации входных данных от их размера для алгоритма Multinomial Naïve Bayes

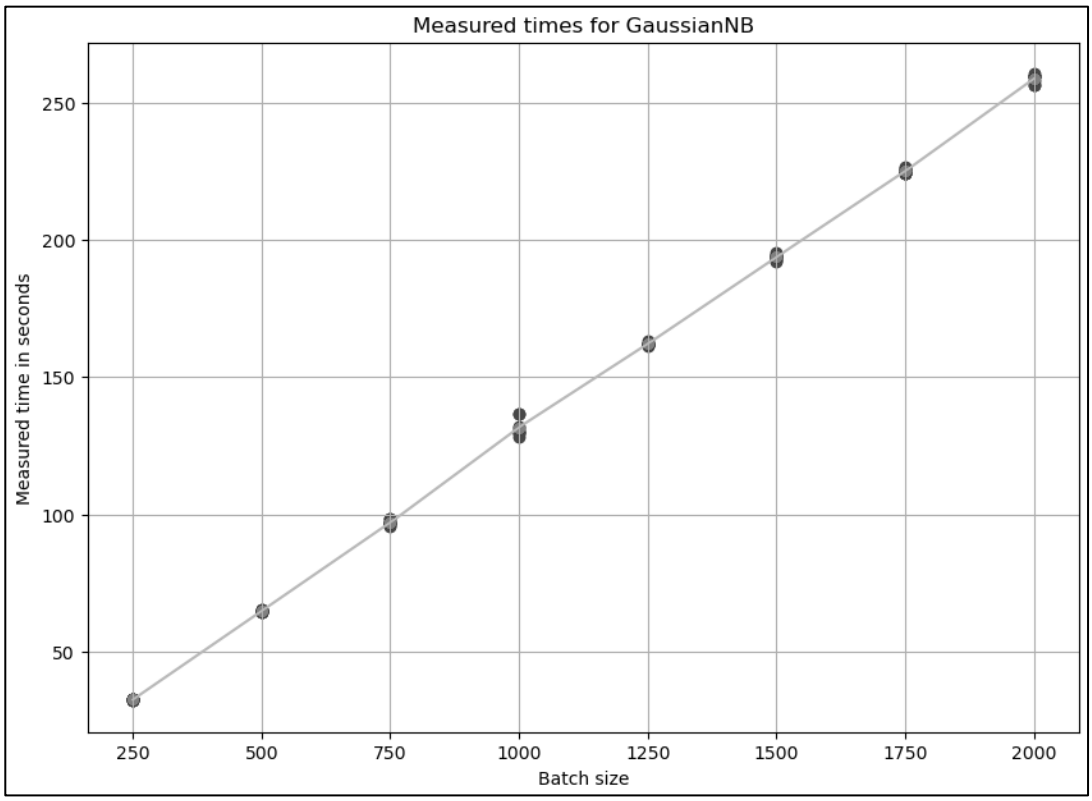


Рисунок 42 – Зависимость времени классификации входных данных от их размера для алгоритма Gaussian Naïve Bayes

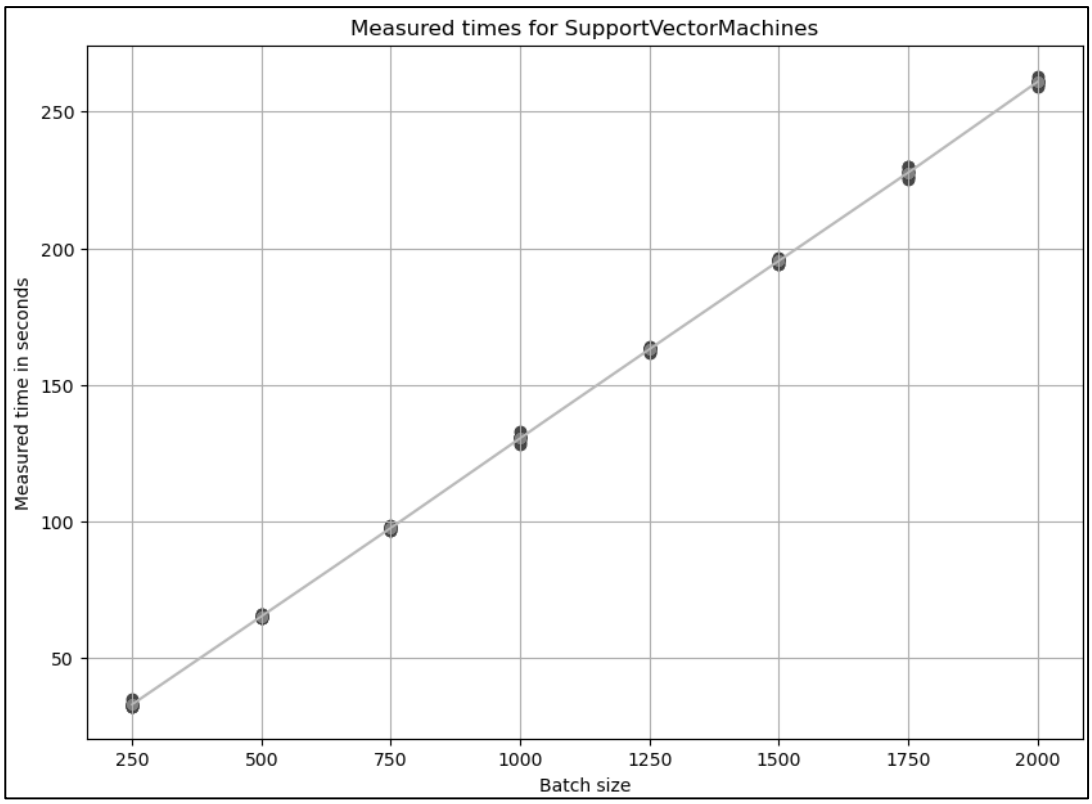


Рисунок 43 – Зависимость времени классификации входных данных от их размера для алгоритма Support Vector Machines

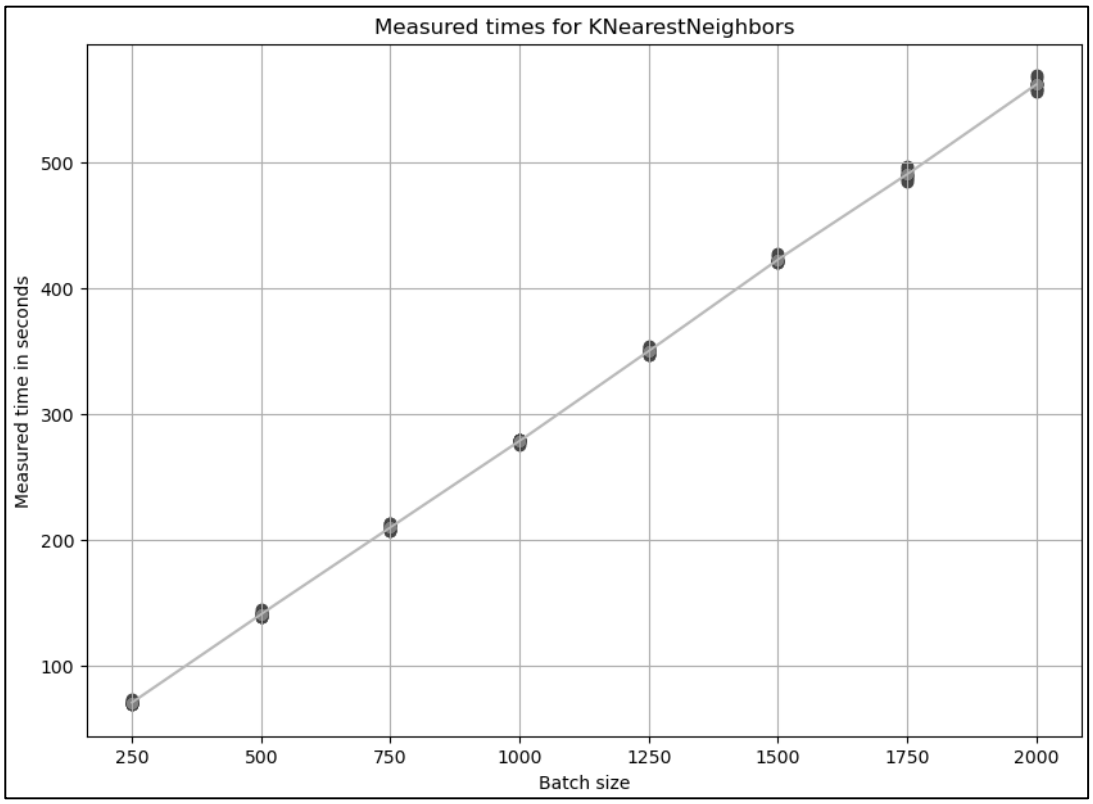


Рисунок 44 – Зависимость времени классификации входных данных от их размера для алгоритма K-Nearest Neighbors

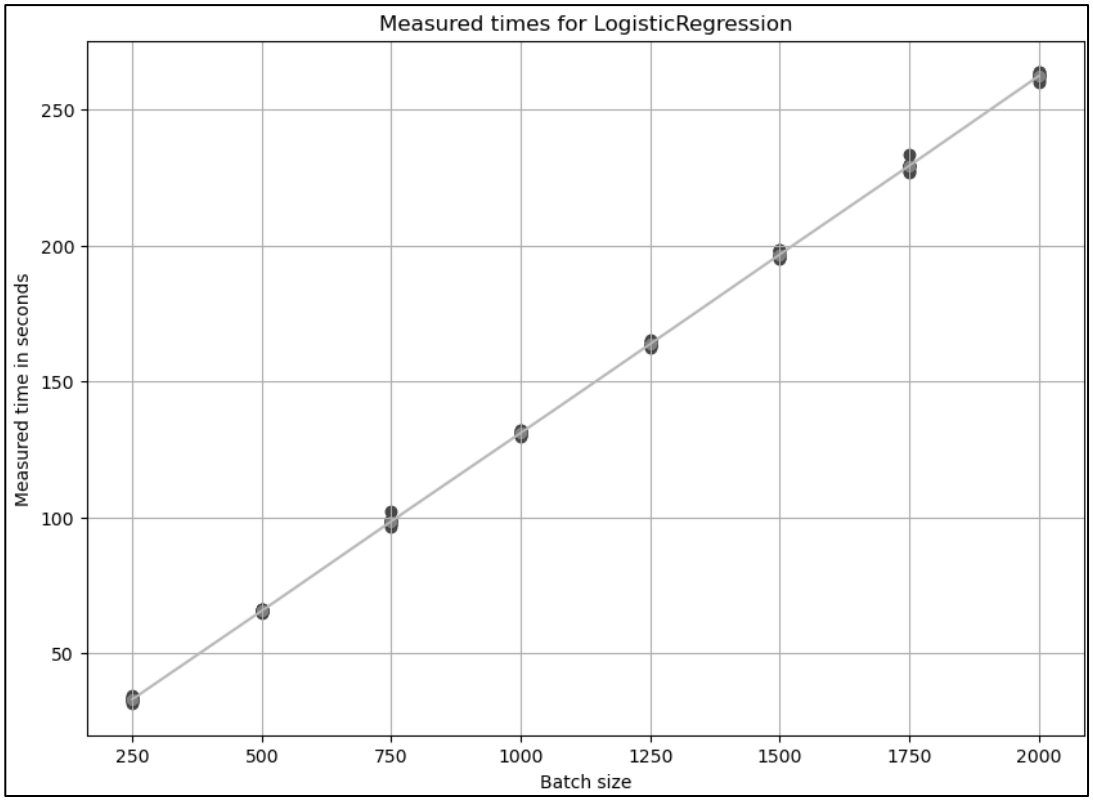


Рисунок 45 – Зависимость времени классификации входных данных от их размера для алгоритма Logistic Regression

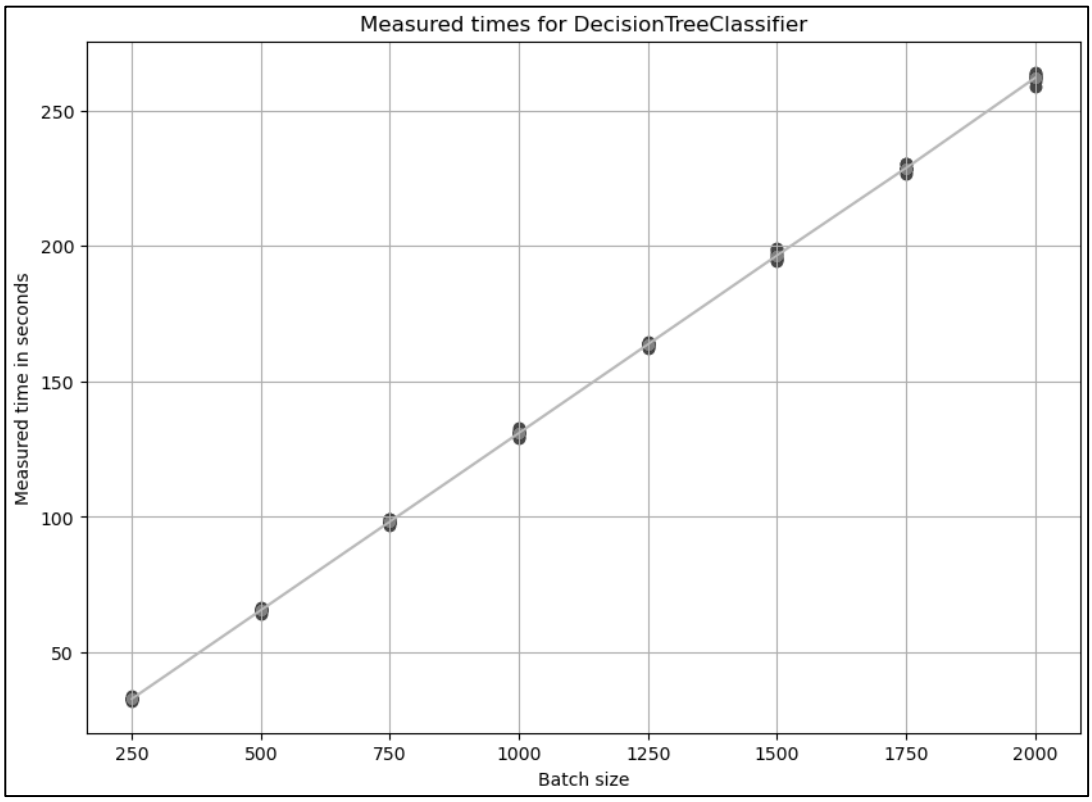


Рисунок 46 – Зависимость времени классификации входных данных от их размера для алгоритма Decision Tree

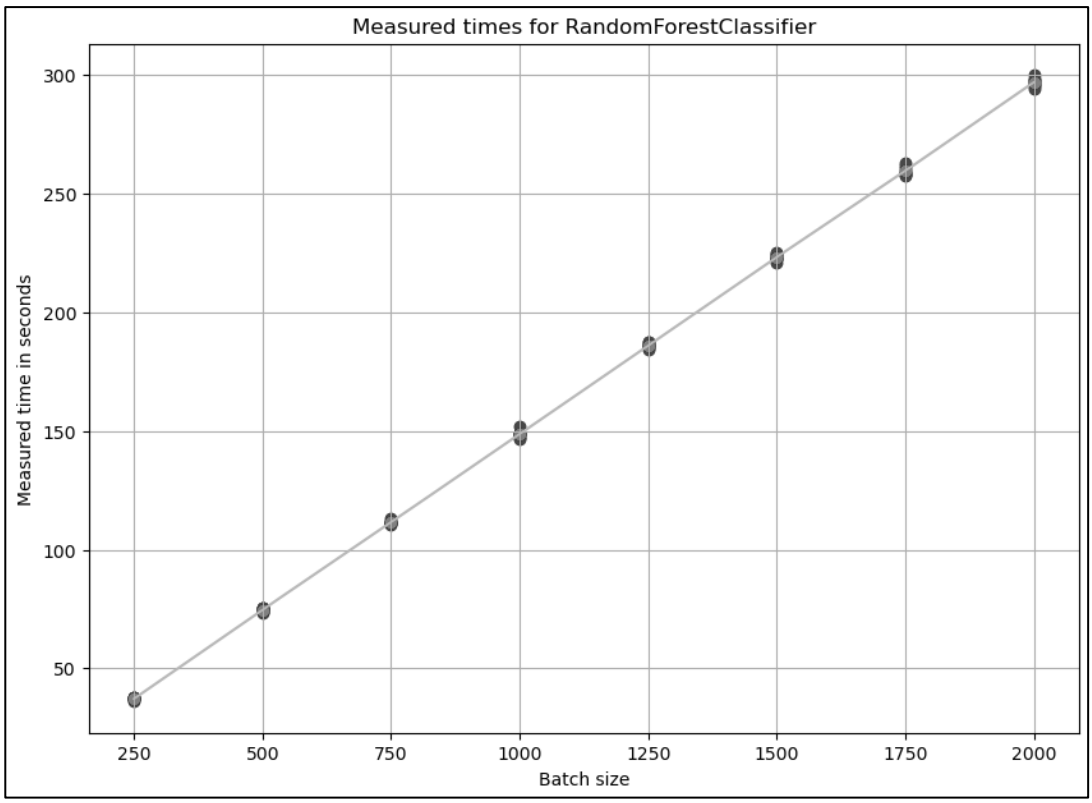


Рисунок 47 – Зависимость времени классификации входных данных от их размера для алгоритма Random Forest

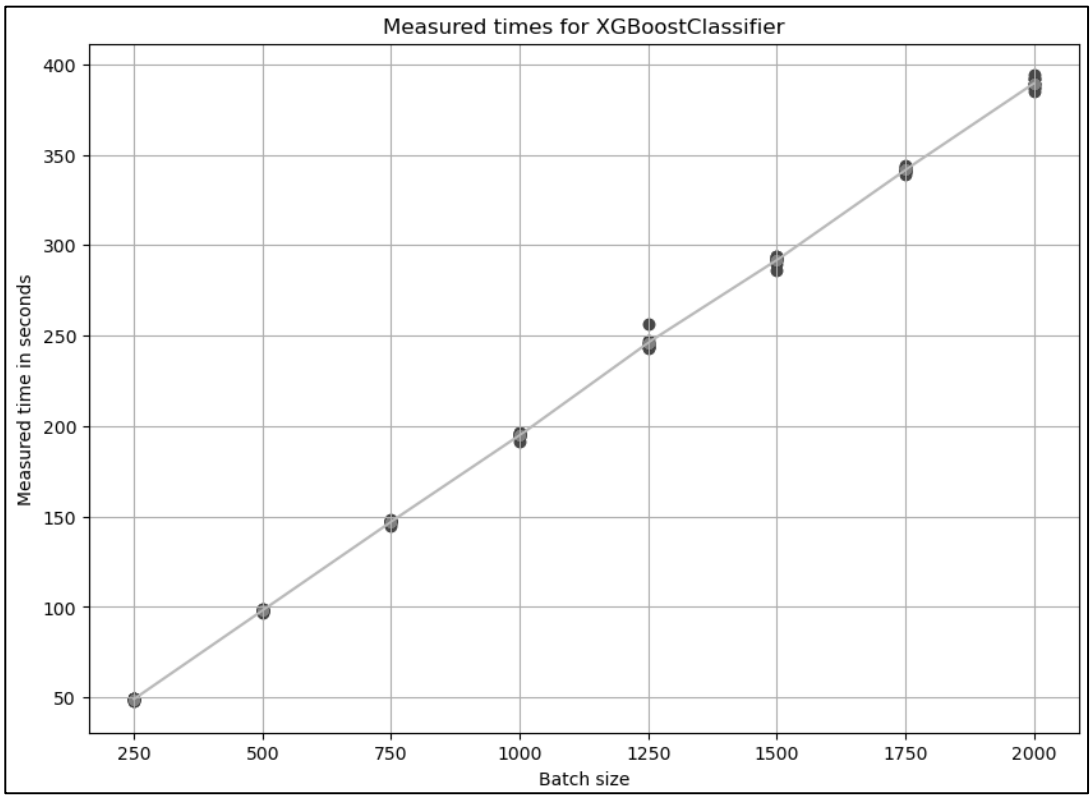


Рисунок 48 – Зависимость времени классификации входных данных от их размера для алгоритма XGBoost

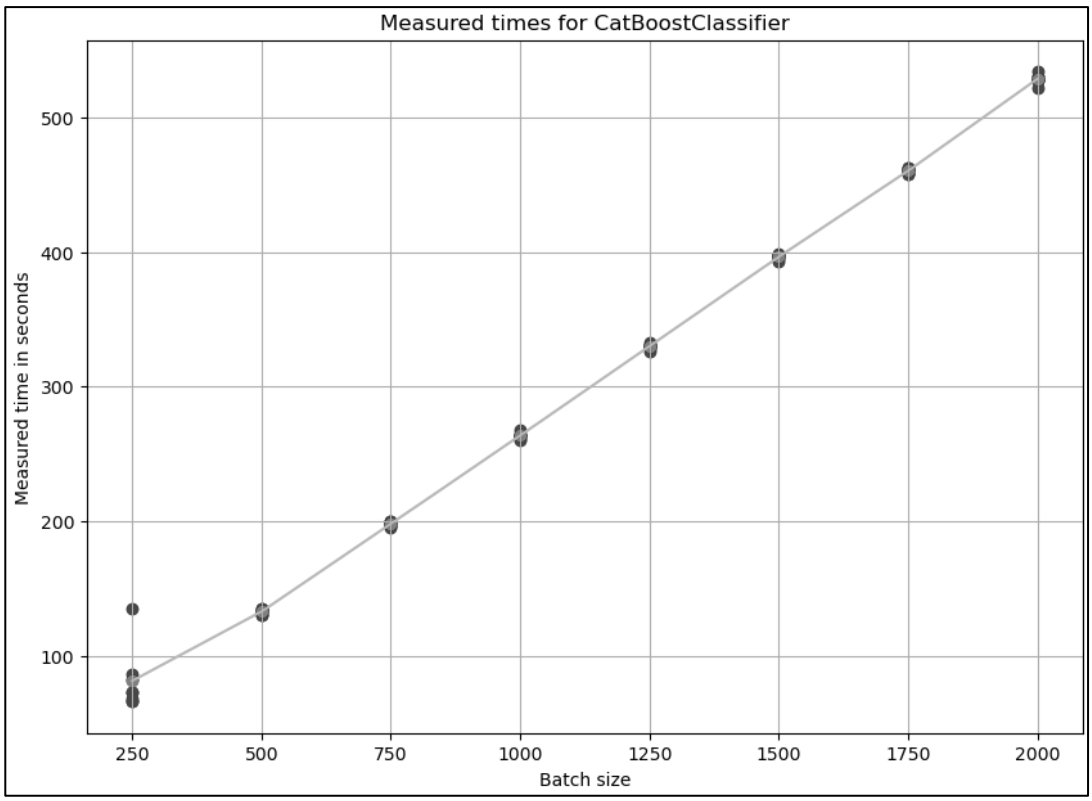


Рисунок 49 – Зависимость времени классификации входных данных от их размера для алгоритма CatBoost

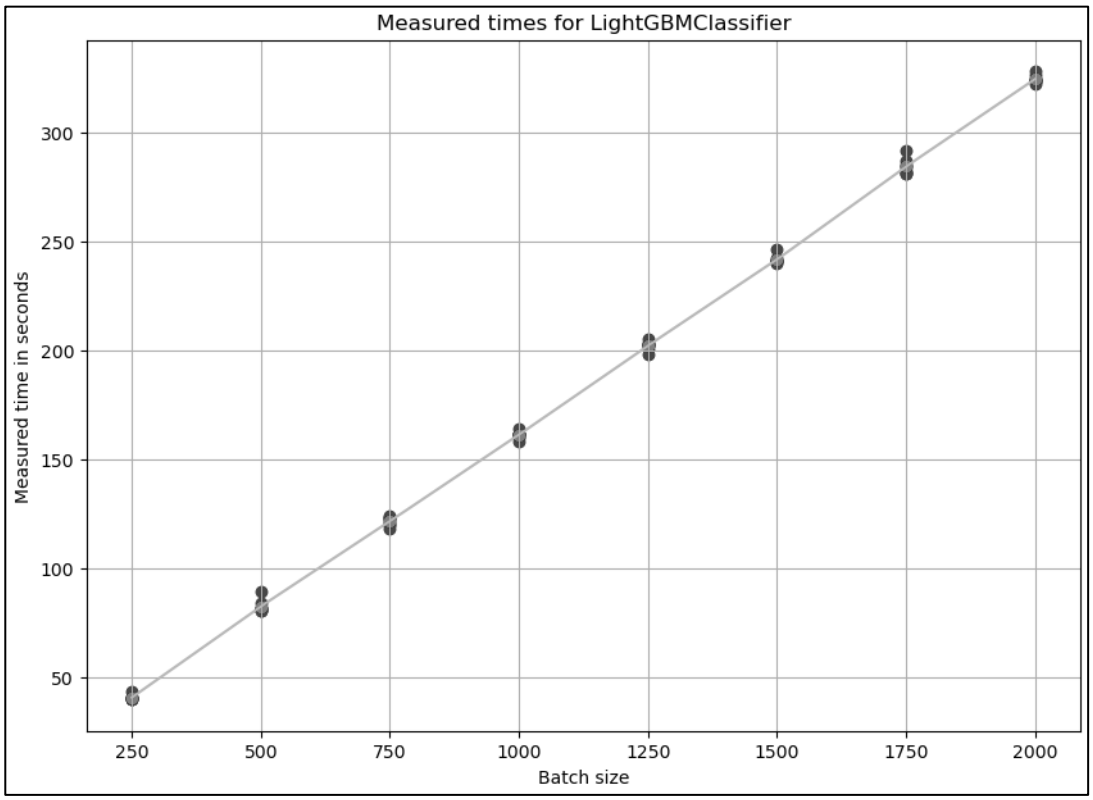


Рисунок 50 – Зависимость времени классификации входных данных от их размера для алгоритма LightGBM

Таким образом, для всех моделей была получена линейная зависимость времени классификации входных данных от их объема.

4.2. Функциональное тестирование веб-приложения

Функциональное тестирование – это вид тестирования, направленный на проверку корректности работы функциональности приложения [56].

Набор функциональных тестов представлен в таблице 2 приложения В.

4.2. Тестирование Docker контейнеров

В данном пункте представлено тестирование корректности запуска контейнеров Docker.

Всего в docker-compose.yml объявлено 9 сервисов: traefik, backend, frontend, minio, create_buckets, mongo, worker, flower и redis.

Некоторые из них зависят от запуска других сервисов. Например, сервис backend не начнет свою работу, пока не запустится сервис worker, а сервис create_buckets не запустится, пока не начнет работать minio.

Убедиться в том, что сервисы запущены, можно с помощью команды `docker ps` в терминале. Пример результата выполнения команды при успешном запуске всех сервисов представлен на рисунке 51.

Рисунок 51 – Список запущенных контейнеров

В данной таблице отсутствует контейнер сервиса create_buckets, так как его логика подразумевает выполнение конечного скрипта и завершение работы.

Как можно заметить, контейнер сервиса traefik использует 2 порта хост-машины:

1) 8080 порт, который используется для мониторинга транспорта трафика;

2) 5000 порт, который используется как обратный прокси сервер, через него производится балансировка трафика.

В виртуальном адресном пространстве Docker это 8080 и 80 порты, соответственно. На них будут проксироваться порты на хост-машине.

Веб-приложение для мониторинга транспорта трафика, доступное по 8080 порту, представлено на рисунке 52.

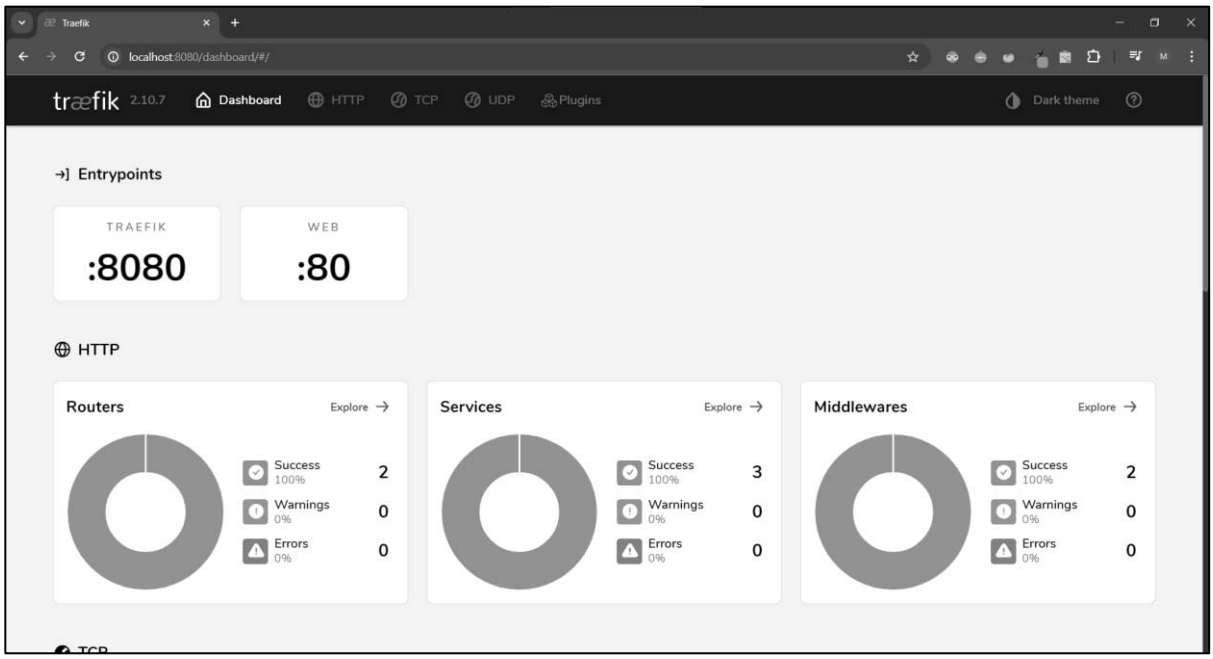


Рисунок 52 – Веб-приложение Traefik

Контейнер сервиса frontend использует 3000 порт хост-машины, который в адресном пространстве Docker проксируется также на 3000 порт. Главная страница клиентской части веб-приложения представлена на рисунке 53.

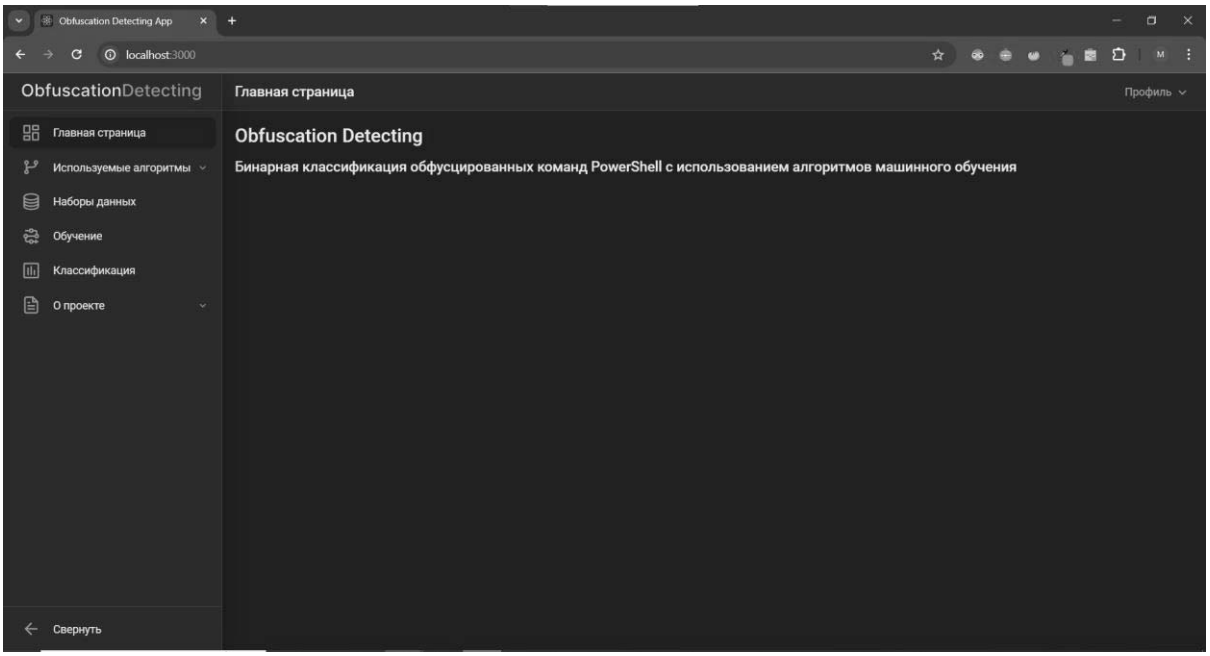


Рисунок 53 – Главная страница клиентской части веб-приложения

Контейнер сервиса backend использует 8000 порт хост-машины, который в адресном пространстве Docker проксируется также на 8000 порт. Swagger UI, доступный по url /docs, представлен на рисунке 54.

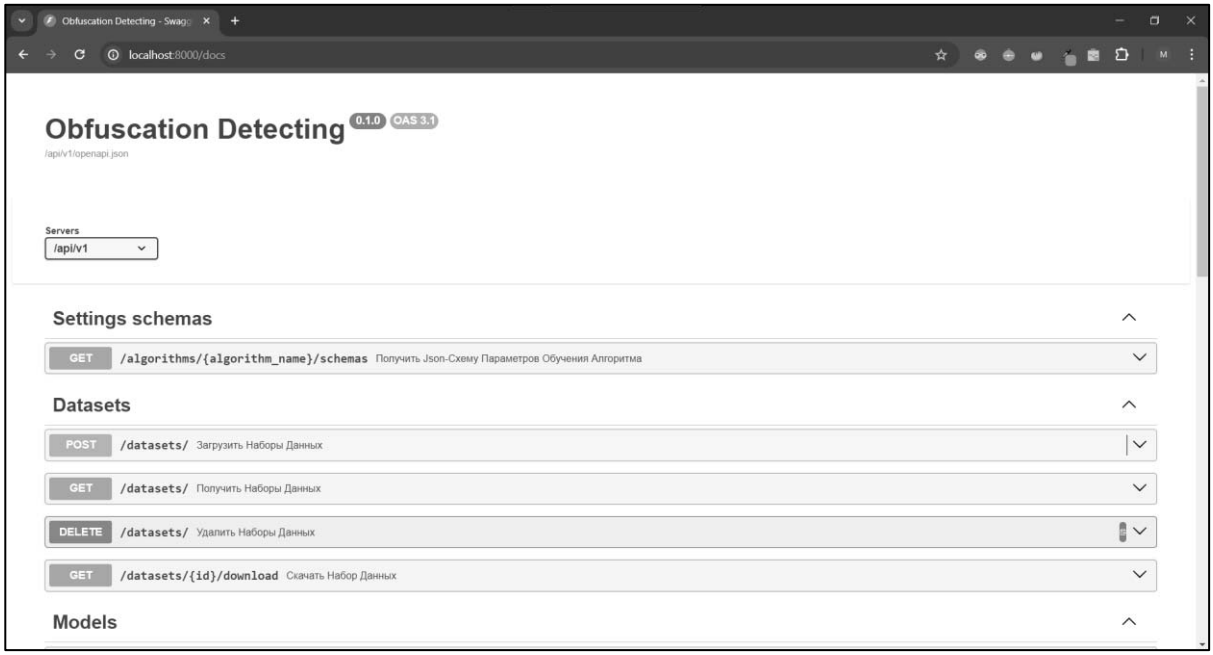


Рисунок 54 – Swagger UI серверной части веб-приложения

Контейнер сервиса flower использует 5555 порт хост-машины, который в адресном пространстве Docker проксируется также на 5555 порт. Веб-приложение для мониторинга обработки задач представлено на рисунке 55.

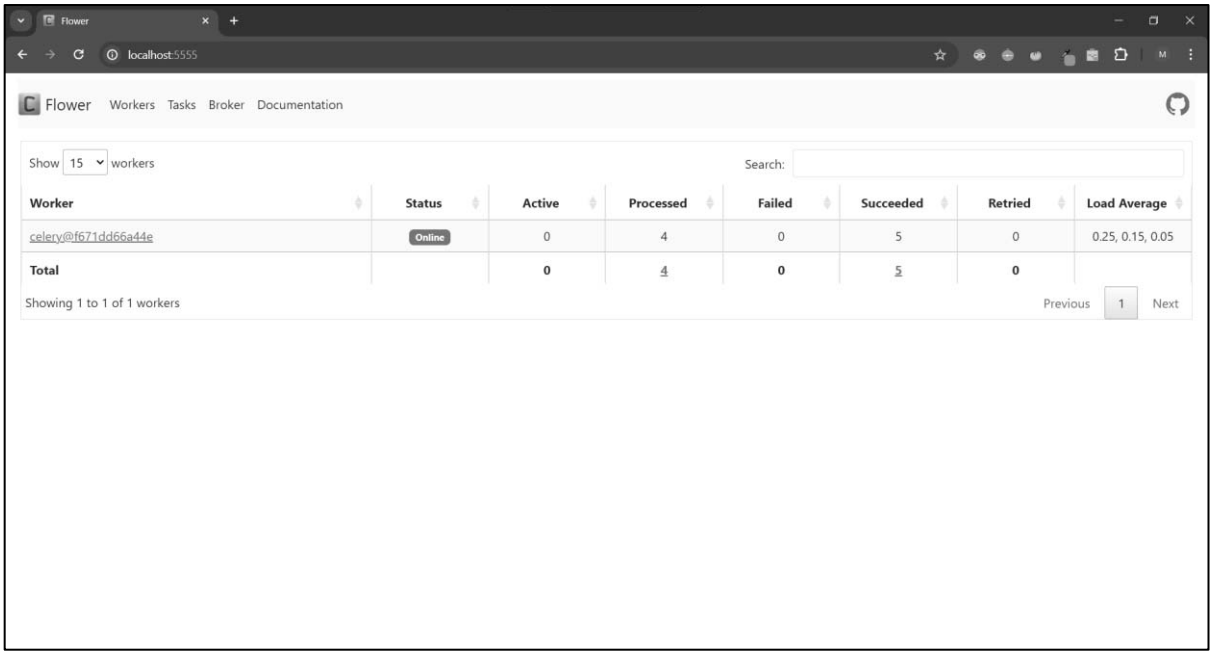


Рисунок 55 – Веб-приложение Flower

Контейнер сервиса minio использует 9000 порт хост-машины, который в адресном пространстве Docker проксируется на 9001 порт. Веб-приложение для хранения файлов представлено на рисунках 56 и 57.

Для входа необходимо пройти аутентификацию на странице входа, расположенной по url /login, затем у пользователя появляется возможность посмотреть доступные backet-ы с загруженными данными по url /browser.

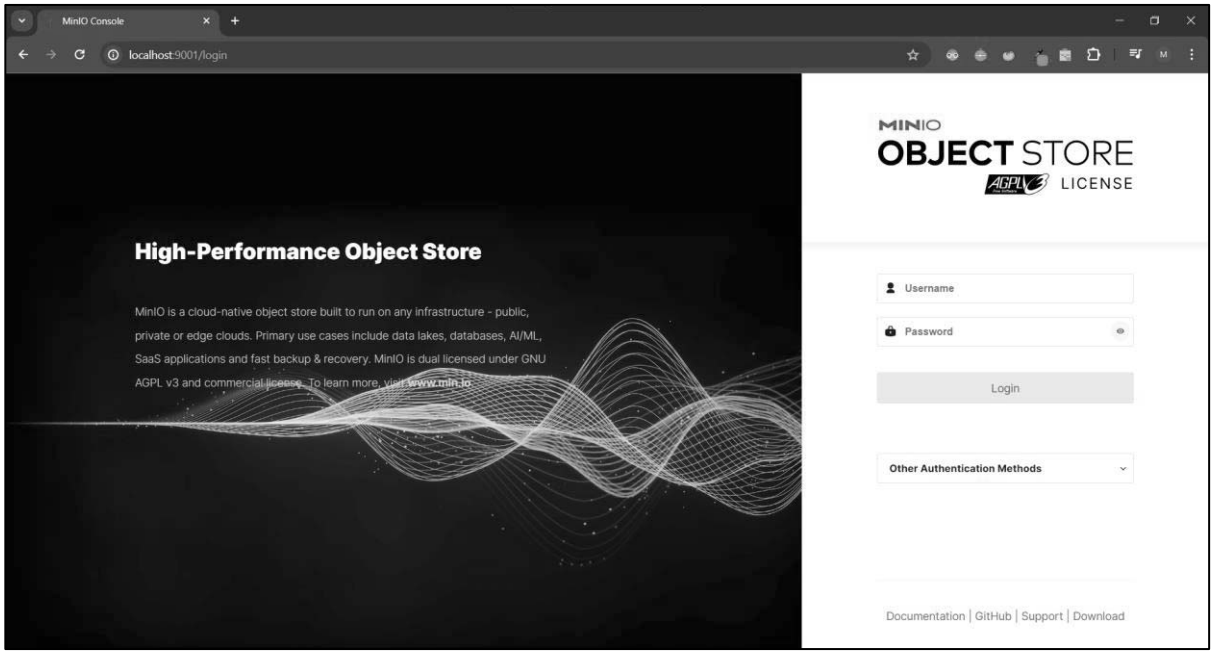


Рисунок 56 – Страница входа в аккаунт MinIO

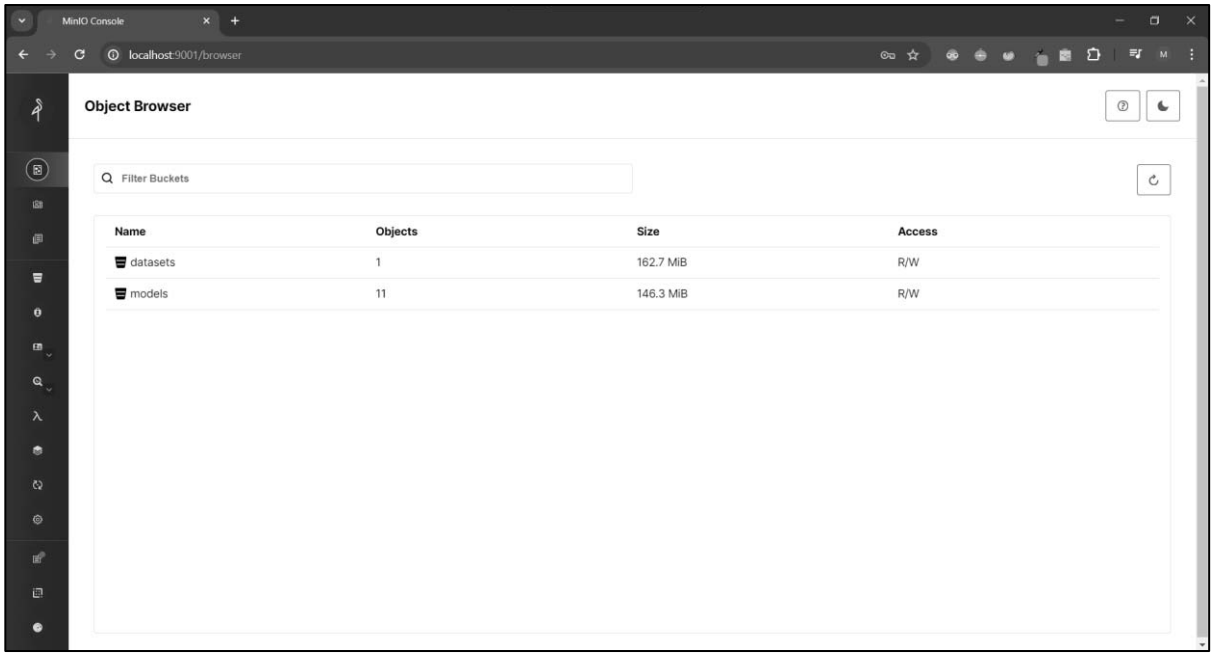


Рисунок 57 – Данные в объектном хранилище MinIO

Данные для аутентификации, названия bucket-ов вынесены в конфигурационные данные веб-приложения (.env файл), а bucket-ы создаются с помощью контейнера сервиса create_buckets. На рисунке 57 показано, что в объектном хранилище уже есть загруженные данные: это исходные данные, которые были загружены с помощью инициализатора в обработчике celery.

Контейнер сервиса worker является процессом, который осуществляет обработку задач в очереди. Также в нем инициализируются задачи для загрузки исходных данных в базу данных и объектное хранилище.

Для проверки корректности работы контейнеров сервисов redis и mongo попробуем подключиться к консолям баз данных. Сделать это можно с помощью команды `docker exec -it <id контейнера> <команда>`.

Контейнер сервиса mongo использует 27017 порт хост-машины, который в адресном пространстве Docker проксируется также на 27017 порт. Пример подключения к базе данных MongoDB и вывода списка коллекций представлен на рисунке 58.

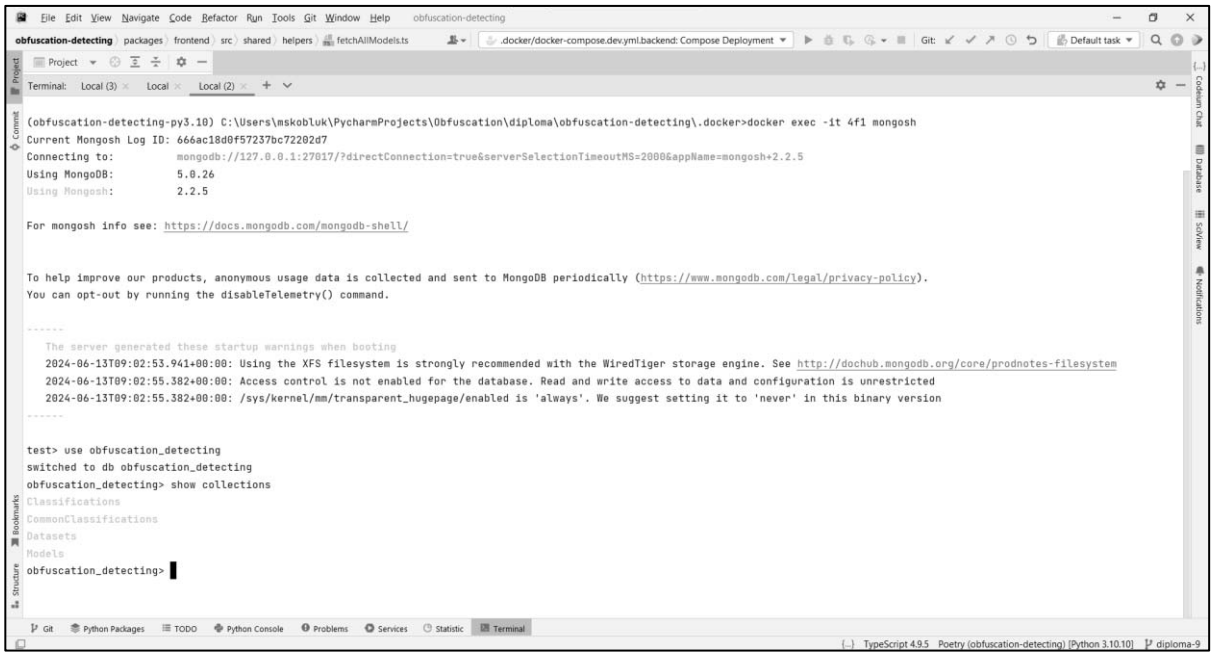


Рисунок 58 – Подключение к базе данных MongoDB и вывод доступных коллекций

Контейнер сервиса redis использует 6379 порт хост-машины, который в адресном пространстве Docker проксируется также на 6379 порт. Пример

подключения к базе данных Redis и вывода списка ключей представлен на рисунке 59.

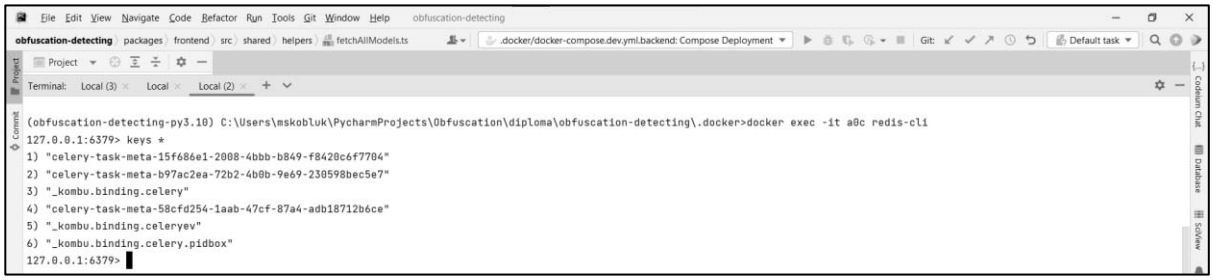


Рисунок 59 – Подключение к базе данных Redis и вывод списка ключей

Таким образом, все запущенные Docker контейнеры работают корректно.

Выводы по четвертой главе

В четвертой главе было проведено тестирование алгоритмов машинного обучения – проверка эффективности обученных моделей на новых наборах данных – исключенные в этапе предобработки команды, содержащие применение оболочки cmd, оставшиеся в наборе данных команды, которые были исключены для достижения баланса данных по классам, сгенерированные из необфусцированных данных слабообфусцированные команды, содержащие символы «`», а также команды, содержащие кириллические символы, вычисление производительности классификации данных с помощью моделей и зависимости времени классификации от размера входных данных, функциональное тестирование веб-приложения, в котором все тесты были успешно пройдены, а также тестирование Docker контейнеров, которое демонстрирует, что сервисы Docker настроены правильно и работают без ошибок.

ЗАКЛЮЧЕНИЕ

В соответствии с целью данной выпускной квалификационной работы было разработано веб-приложение для бинарной классификации обфусцированных команд PowerShell с использованием алгоритмов машинного обучения.

Были решены следующие поставленные задачи:

1) проведен обзор научной литературы;
2) подготовлен обучающий набор данных;
3) реализованы выбранные методы машинного обучения;
4) разработано веб-приложение для бинарной классификации обфусцированных команд PowerShell;
5) проведено тестирование разработанного приложения.

Реализованное приложение и обученные модели машинного обучения были внедрены в эксплуатацию компании R-Vision, что подтверждает сформированный акт о внедрении.

В будущем планируется увеличить объем данных и число используемых алгоритмов, использовать нейросети, а также реализовать многоклассовую классификацию команд, опираясь для начала на исходный размеченный набор данных, который необходимо для начала будет иначе предобработать.

ЛИТЕРАТУРА

1. Жерон А. Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow. // O'Reilly Media, 2019. – С. 2, 112.

2. Лутц М. Изучаем Python. Полное руководство для начинающих, 5-е издание. // Москва: Вильямс, 2019. – С. 34.

3. Холмс Л. Windows PowerShell Cookbook: Полное руководство по созданию сценариев для командной оболочки Microsoft. // O'Reilly Media, 2013. – С. 15.

4. Кац Дж. Криптография на языке Java. // O'Reilly Media, 2017. – С. 254.

5. Уилсон Э. Лучшие практики PowerShell 2.0. // O'Reilly Media, 2019. – С. 494.

6. На Хасселл Дж. Изучение PowerShell: автоматизация администрирования Windows. // O'Reilly Media, 2015. – С. 319.

7. Вандер Плас Дж. Python для анализа данных. // Вильямс, 2016. – С. 145.

8. Буч Г., Рамбо Дж., Якобсон И. Руководство пользователя языка моделирования UML. // Addison-Wesley Professional, 1998. – С. 2.

9. McAfee report. [Электронный ресурс] URL: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-mar-2018.pdf (дата обращения: 14.01.2024 г.).

10. Symantec blog. PowerShell Threats Grow Further and Operate in Plain Sight. [Электронный ресурс] URL: https://www.symantec.com/blogs/threat-intelligence/powershell-threats-grow-further-and-operate-plain-sight (дата обращения: 15.01.2024 г.).

11. Исследование отчета McAfee. [Электронный ресурс] URL: https://startpack.ru/articles/20210415-macafe (дата обращения: 17.01.2024 г.).

12. Analysis Shows Attackers Favor PowerShell, File Obfuscation. [Электронный ресурс] URL: https://www.darkreading.com/cybersecurity-analytics/analysis-shows-attackers-favor-powershell-file-obfuscation (дата обращения: 19.01.2024 г.).

13. (Ex)Cobalt в новом обличье: исследование последней атаки известной группировки. [Электронный ресурс] URL: https://rt-solar.ru/solar-4rays/blog/3954/ (дата обращения: 19.01.2024 г.).

14. Вредонос Gootkit обзавелся новыми компонентами и методами обфускации. [Электронный ресурс] URL: https://is-systems.org/blog_article/11675176581 (дата обращения: 19.01.2024 г.).

15. Cynet. Powershell Obfuscation Demystified Series Chapter 1: Intro. [Электронный ресурс] URL: https://www.cynet.com/attack-techniques-hands-on/powershell-obfuscation-demystified-series-chapter-1-intro/ (дата обращения: 26.01.2024 г.).

16. Meng-Han T., Chia-Ching L., Zheng-Gang H., Wei-Chieh Y., Chin-Laung L. PowerDP: De-Obfuscating and Profiling Malicious PowerShell Commands With Multi-Label Classifiers. // December 26, 2022. IEEE, 2022. – 99 pp. DOI: 10.1109/ACCESS.2022.3232505.

17. Revoke-Obfuscation. [Электронный ресурс] URL: https://github.com/danielbohannon/Revoke-Obfuscation (дата обращения: 02.02.2024 г.).

18. Danny H., Shay K., Amir R. Detecting Malicious PowerShell Commands using Deep Neural Networks [Электронный ресурс] // arXiv.org. 2018. Дата обновления: 14.04.2018 г. URL: https://arxiv.org/pdf/1804.04177 (дата обращения: 29.01.2024 г.).

19. Gili R., Abdullah A., Una-May O. AST-Based Deep Learning for Detecting Malicious PowerShell [Электронный ресурс] // arXiv.org. 2018. Дата обновления: 03.10.2018 г. URL: https://arxiv.org/pdf/1810.09230 (дата обращения: 29.01.2024 г.).

20. Jihyeon S., Jungtae K., Sunoh C., Jonghyun K., Ikkyun K. Evaluations of AI-based malicious PowerShell detection with feature optimizations. // November 25, 2020. DOI: 10.4218/etrij.2020-0215.

21. Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. // O'Reilly Media, 2019. – С. 37–38.

22. Scikit-learn: Naïve Bayes. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/naive_bayes.html (дата обращения: 24.01.2024 г.).

23. Scikit-learn: Multinomual Naive Bayes. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html (дата обращения: 24.01.2024 г.).

24. Scikit-learn: Gaussian Naive Bayes. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html (дата обращения: 24.01.2024 г.).

25. Scikit-learn: Support Vector Machines. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/svm.html (дата обращения: 28.01.2024 г.).

26. Scikit-learn: Nearest Neighbors. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/neighbors.html (дата обращения: 28.01.2024 г.).

27. Scikit-learn: Logistic Regression. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (дата обращения: 08.02.2024 г.).

28. Scikit-learn: Decision Trees. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/tree.html (дата обращения: 11.02.2024 г.).

29. Scikit-learn: Random Forest Classifier. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (дата обращения: 11.02.2024 г.).

30. XGBoost Documentation. [Электронный ресурс] URL: https://xgboost.readthedocs.io/en/stable/ (дата обращения: 13.02.2024 г.).

31. CatBoost Documentation. [Электронный ресурс] URL: https://catboost.ai/en/docs/ (дата обращения: 13.02.2024 г.).

32. LightGBM Documentation. [Электронный ресурс] URL: https://lightgbm.readthedocs.io/en/latest/Python-API.html (дата обращения: 13.02.2024 г.).

33. Wiegers K. E., Beatty J. «Software Requirements: Руководство по требованиям к программному обеспечению». 2-е изд. // Символ-Плюс. СПб, 2013. – С. 32.

34. Lauesen S. Software Requirements: Styles & Techniques. // Addison-Wesley Professional, 2002. – С. 77.

35. Смирнов М. В., Толмасов Р. С. «Графическая нотация моделирования объектных баз данных». 2021.

36. FastAPI Documentation. [Электронный ресурс] URL: https://fastapi.tiangolo.com/ (дата обращения: 14.03.2024 г.).

37. React Documentation. [Электронный ресурс] URL: https://legacy.reactjs.org/docs/getting-started.html (дата обращения: 15.04.2024 г.).

38. Jupyter Documentation. [Электронный ресурс] URL: https://docs.jupyter.org/en/latest/ (дата обращения: 09.02.2024 г.).

39. Poetry Documentation. [Электронный ресурс] URL: https://python-poetry.org/docs/ (дата обращения: 10.03.2024 г.).

40. Npm Documentation. [Электронный ресурс] URL: https://docs.npmjs.com/ (дата обращения: 12.04.2024 г.).

41. Pandas Documentation. [Электронный ресурс] URL: https://pandas.pydata.org/docs/ (дата обращения: 19.02.2024 г.).

42. Numpy Documentation. [Электронный ресурс] URL: https://numpy.org/doc/ (дата обращения: 21.02.2024 г.).

43. Matplotlib Documentation. [Электронный ресурс] URL: https://matplotlib.org/stable/index.html (дата обращения: 24.02.2024 г.).

44. Seaborn Documentation. [Электронный ресурс] URL: https://seaborn.pydata.org/ (дата обращения: 24.02.2024 г.).

45. Clr-loader Documentation. [Электронный ресурс] URL: https://pythonnet.github.io/clr-loader/ (дата обращения: 15.02.2024 г.).

46. Pythonnet Documentation. [Электронный ресурс] URL: https://pythonnet.github.io/pythonnet/ (дата обращения: 16.02.2024 г.).

47. Scikit-learn Documentation. [Электронный ресурс] URL: https://scikit-learn.org/stable/ (дата обращения: 26.02.2024 г.).

48. Pickle Documentation. [Электронный ресурс] URL: https://docs.python.org/3/library/pickle.html (дата обращения: 25.02.2024 г.).

49. MinIO Documentation. [Электронный ресурс] URL: https://min.io/docs/minio/linux/developers/python/API.html (дата обращения: 18.03.2024 г.).

50. PyMongo Documentation. [Электронный ресурс] URL: https://pymongo.readthedocs.io/en/stable/ (дата обращения: 20.03.2024 г.).

51. Celery Documentation. [Электронный ресурс] URL: https://docs.celeryq.dev/en/stable/ (дата обращения: 25.03.2024 г.).

52. Redis Documentation. [Электронный ресурс] URL: https://redis.io/docs/latest/ (дата обращения: 26.03.2024 г.).

53. Pydantic Documentation. [Электронный ресурс] URL: https://docs.pydantic.dev/latest/ (дата обращения: 14.03.2024 г.).

54. Axios Documentation. [Электронный ресурс] URL: https://axios-http.com/docs/intro (дата обращения: 18.04.2024 г.).

55. Docker Documentation. [Электронный ресурс] URL: https://docs.docker.com/ (дата обращения: 07.05.2024 г.).

56. Куликов С. Тестирование программного обеспечения. Базовый курс. 2-е издание. // Минск: Четыре четверти, 2020. – С. 80.

ПРИЛОЖЕНИЯ

Приложение А. Обучение алгоритмов машинного обучения

Листинг 1 – Реализация класса ModelLearningPipeline

```
class ModelLearningPipeline:
    def __init__(self, model_class: Model, hyperparameters: list[dict[str,
Iterable]], models_to_validation: int = 5) -> None:
        self.__model_class = model_class
        self.__hyperparameters = hyperparameters
        self.__best_model: Model | None = None
        self.__best_model_candidates: list[model_class | None] = [None] *
models_to_validation

    def cross_validate_initially(self, cv: int = 5) -> np.ndarray[float]:
        model = self.__model_class()
        y = train_labels.values.ravel() if type(train_labels) is pd.Data-
Frame else train_labels.ravel()
        return cross_val_score(estimator=model, X=train_features, y=y,
n_jobs=-1, cv=cv)

    def tuning(self, cv: int = 5, show: bool = True, scoring: str = "accu-
racy") -> None:
        model = self.__model_class()
        clf = GridSearchCV(estimator=model, param_grid=self.__hyperparame-
ters, cv=cv, n_jobs=-1, scoring=scoring, verbose=2)
        y = train_labels.values.ravel() if type(train_labels) is pd.Data-
Frame else train_labels.ravel()
        clf.fit(X=train_features, y=y)

        means: np.ndarray[float] = clf.cv_results_["mean_test_score"]
        stds: np.ndarray[float] = clf.cv_results_["std_test_score"]
        params_combinatons: np.ndarray[dict[str, str | int | float]] =
clf.cv_results_["params"]

        i: int = 0
        for mean, std, params in sorted(zip(means, stds, params_combina-
tons), key=lambda values: 1 - values[0] + values[1]):  # sorted from high
to low scores
            if np.isnan(mean):
                continue
            if i >= len(self.__best_model_candidates):
                break
            model = self.__model_class(**params)
            model.fit(X=train_features, y=y)
            self.__best_model_candidates[i] = model
            i += 1

        if show:
            MetricAnalyzer.show_tuning_result(means=means, stds=stds,
params_combinatons=params_combinatons)

    def test(self, show: bool = True) -> None:
        if not self.__exist_all_candidates():
            return

        accuracy_scores = [0.0] * len(self.__best_model_candidates)  # true
predictions (TP + TN) / all predictions (TP + TN + FP + FN)
        precision_scores = copy.copy(accuracy_scores)  # true positive pre-
dictions (TP) / all positive predictions (TP + FP)
        recall_scores = copy.copy(accuracy_scores)  # true positive predic-
tions (TP) / all positive labels (TP + FN)
```

```
        f1_scores = copy.copy(accuracy_scores)  # 2 * precision * recall /
(precision + recall)

        for i, candidate_model in enumerate(self.__best_model_candidates):
            scores = MetricAnalyzer.score_metrics(model=candidate_model,
X=test_features, y_true=test_labels)
            accuracy_scores[i] = scores.get("accuracy", None)
            precision_scores[i] = scores.get("precision", None)
            recall_scores[i] = scores.get("recall", None)
            f1_scores[i] = scores.get("f1", None)

        best_metrics_value = 0.

        for candidate_model, accuracy, precision, recall, f1 in
zip(self.__best_model_candidates, accuracy_scores, precision_scores, re-
call_scores, f1_scores):
            if accuracy + precision + recall <= best_metrics_value:
                continue
            best_metrics_value = accuracy + precision + recall
            self.__best_model = candidate_model

        if show:
            MetricAnalyzer.show_evaluated_result(best_models_candi-
dates=self.__best_model_candidates, accuracy_scores=accuracy_scores, preci-
sion_scores=precision_scores, recall_scores=recall_scores,
f1_scores=f1_scores)

    def validate(self) -> None:
        if self.__best_model is None:
            print("For checking the scores of the model on data for valida-
tion the model must be initialized!")
            return
        scores = MetricAnalyzer.score_metrics(model=self.__best_model,
X=validate_features, y_true=validate_labels)
        accuracy = scores.get("accuracy", None)
        precision = scores.get("precision", None)
        recall = scores.get("recall", None)
        f1 = scores.get("f1", None)

        MetricAnalyzer.show_evaluated_result(best_models_candi-
dates=[self.__best_model], accuracy_scores=[accuracy], preci-
sion_scores=[precision], recall_scores=[recall], f1_scores=[f1])

    def __exist_all_candidates(self) -> bool:
        for candidate_model in self.__best_model_candidates:
            if candidate_model is None:
                return False
        return True

    def save(self, pkl_filename: str) -> bool:
        if self.__best_model is None:
            print("For saving the model must be initialized!")
            return False
        try:
            with open(pkl_filename, 'wb') as file:
                pickle.dump(self.__best_model, file=file)
        except:
            return False
        return True
```

Листинг 2 – Реализация класса MetricAnalyzer

```
class MetricAnalyzer:
    @staticmethod
    def score_metrics(model: Model, X: pd.DataFrame, y_true: pd.Series) ->
dict[str, float]:
        metrics = {}
        y_pred = model.predict(X)
        metrics["accuracy"] = np.round(accuracy_score(y_true=y_true,
y_pred=y_pred), 5)
        metrics["precision"] = np.round(precision_score(y_true=y_true,
y_pred=y_pred), 5)
        metrics["recall"] = np.round(recall_score(y_true=y_true,
y_pred=y_pred), 5)
        metrics["f1"] = np.round(f1_score(y_true=y_true, y_pred=y_pred), 5)
        return metrics

    @staticmethod
    def show_tuning_result(means: np.ndarray[float], stds: np.ndar-
ray[float], params_combinatons: np.ndarray[dict[str, int | float | str]]) -
> None:
        print("Tuning Result:\n")
        for mean, std, params in sorted(zip(means, stds, params_combina-
tons), key=lambda values: 1 - values[0] + values[1]):
            if np.isnan(mean):
                continue
            print(f"Mean: {np.round(mean, 5)} || Standard deviation:
{np.round(std, 5)} || Hyperparameters: {params}\n")

    @staticmethod
    def show_evaluated_result(best_models_candidates: list[Model], accu-
racy_scores: list[float], precision_scores: list[float], recall_scores:
list[float], f1_scores: list[float]) -> None:
        print(f"Evaluated Result:\n")

        if not (len(best_models_candidates) == len(accuracy_scores) ==
len(precision_scores) == len(recall_scores) == len(f1_scores)):
            raise ValueError("Size of input data is incorrect")

        for candidate_model, accuracy, precision, recall, f1 in
zip(best_models_candidates, accuracy_scores, precision_scores, re-
call_scores, f1_scores):
            print(f"Model: {candidate_model} || Accuracy: {accuracy} ||
Precision: {precision} || Recall: {recall} || F1: {f1}\n")

    @staticmethod
    def show_confusion_matrix(ax, model: Model, X: pd.DataFrame, y_true:
pd.Series) -> None:
        y_pred = model.predict(X)
        cm = confusion_matrix(y_true=y_true, y_pred=y_pred)
        sns.heatmap(cm, annot=True, fmt="d", ax=ax)

    @staticmethod
    def show_classification_report(model: Model, X: pd.DataFrame, y_true:
pd.Series) -> None:
        y_pred = model.predict(X)
        print(classification_report(y_pred=y_pred, y_true=y_true))

    @staticmethod
    def show_roc_curve(ax, model: Model, X: pd.DataFrame, y_true: pd.Se-
ries) -> None:
        y_pred = model.predict(X)
```

```
fpr, tpr, thresholds = roc_curve(y_true=y_true, y_score=y_pred)
ax.plot(fpr, tpr, label='ROC curve')
ax.plot([0, 1], [0, 1], 'k--', label='Random guess')
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_title('ROC Curve')
ax.set_xlim([-0.02, 1])
ax.set_ylim([0, 1.02])
ax.legend(loc="lower right")
```

Приложение Б. Роуты FastAPI приложения

Таблица 1 – Роуты FastAPI приложения

Метод	Путь	Параметры	Принцип работы	Результат
GET	/algorithms/{algorithm_name}/schemas	Path: algorithm_name (название алгоритма)	Для выбранного алгоритма формируется JSON-схема параметров обучения, которая будет использоваться в генераторе форм	JSON-схема параметров обучения выбранного алгоритма
POST	/datasets/	Body: datasets (список файлов)	Добавление в очередь задачи загрузки файлов в базу данных и объектное хранилище	Ответ в формате JSON, возвращающий 201 статус (операция запущена)
GET	/datasets/	Query: page (сдвиг), size (лимит)	Загрузка документов из базы данных, содержащих информацию о наборах данных	Список документов и мета-информация о пагинации
DELETE	/datasets/	Query: ids (список id наборов данных в базе данных)	Добавление в очередь задач удаления наборов данных с заданными id из базы данных и объектного хранилища, а также обновление всех связанных моделей (удаление информации о наборе данных)	Ответ в формате JSON, возвращающий 201 статус (операция запущена)
GET	/datasets/{id}/download	Path: id (id набора данных в базе данных)	Поиск набора данных с заданным id в базе данных и скачивание его файла из объектного хранилища	CSV-файл

Продолжение таблицы 1 приложения Б

Метод	Путь	Параметры	Принцип работы	Результат
POST	/models/	Body: filename (имя файла), dataset_id (id набора данных в базе данных), training_params (параметры обучения), algorithm_name (название алгоритма), training_data_proportion (объем обучающей выборки)	Добавление в очередь задач задачи обучения выбранного алгоритма на заданных параметрах и наборе данных, тестирования модели на оставшихся данных и сохранения результатов в базу данных и объектное хранилище	Ответ в формате JSON, возвращающий 201 статус (операция запущена)
GET	/models/	Query: page (сдвиг), size (лимит)	Загрузка документов из базы данных, содержащих информацию об обученных моделях, поиск связанных наборов данных и возвращение DTO объектов	Список DTO объектов и мета-информация о пагинации
DELETE	/models/	Query: ids (список id моделей в базе данных)	Добавление в очередь задач задачи удаления обученных моделей с заданными id из базы данных и объектного хранилища, а также всех классификаций (включая пересчет совокупной классификации команд)	Ответ в формате JSON, возвращающий 201 статус (операция запущена)

Продолжение таблицы 1 приложения Б

Метод	Путь	Параметры	Принцип работы	Результат
GET	/models/{id}/ download	Path: id (id модели в базе данных)	Поиск обученной модели с заданным id в базе данных и скачивание её файла из объектного хранилища	Pickle-файл
POST	/classifications/	Body: models_ids (список id моделей в базе данных), commands (файл с предобработанными данными команд)	Добавление в очередь задач задачи классификации входных данных на заданных обученных моделях, включая подсчет совокупного результата обфусцированности команд на основе результатов всех моделей	Ответ в формате JSON, возвращающий 201 статус (операция запущена)
GET	/classifications/	Query: limit (максимальное число объектов)	Загрузка заданного числа документов, содержащих информацию о классификациях команд, из базы данных	Список документов, содержащих информацию о классификациях команд, размер которого не превышает limit и общее число доступных классификаций
GET	/classifications/ commands	Query: command (команда)	Загрузка всех документов из базы данных, содержащих информацию о классификации заданной команды каждой выбранной отдельно взятой обученной моделью	Список документов с информацией о классификациях заданной команды на отдельно взятых моделях

Окончание таблицы 1 приложения Б

Метод	Путь	Параметры	Принцип работы	Результат
GET	/classifications/ download	Query: filename (имя файла)	Формирование результатов классификации всех команд в совокупности и на отдельно взятых выбранных моделях в виде файла с ссылками на скачивание используемых моделей	CSV-файл с заданным именем

Приложение В. Функциональное тестирование приложения

Таблица 2 – Набор функциональных тестов

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Загрузка предобработанных наборов данных	1. Выбрать в боковой панели вкладку «Наборы данных». 2. Нажать на кнопку «Добавить». 3. В появившемся окне нажать на кнопку «Выбрать файл». 4. Выбрать один или несколько предобработанных наборов данных в формате CSV. 5. Нажать на кнопку «Подтвердить».	Наборы данных загружены в базу данных и объектное хранилище. Информация о загруженных файлах спустя некоторое время появится в таблице.	Да
2	Скачивание предобработанного набора данных	1. Выбрать в боковой панели вкладку «Наборы данных». 2. Нажать на кнопку скачивания в строке таблицы у набора данных.	Скачан файл в формате CSV, содержащий набор данных.	Да
3	Удаление предобработанных наборов данных	1. Выбрать в боковой панели вкладку «Наборы данных». 2. Заполнить чекбоксы в строках таблицы у наборов данных. 3. Нажать на кнопку «Удалить». 4. В появившемся окне нажать на кнопку «Удалить».	Наборы данных удалены из базы данных, объектного хранилища и таблицы. Во всех документах связанных с набором данных обученных моделях очищено поле, содержащее id набора данных.	Да
4	Обучение моделей	1. Выбрать в боковой панели вкладку «Обучение». 2. Нажать на кнопку «Добавить». 3. Выбрать алгоритм для обучения. 4. Заполнить имя сериализованного файла. 5. Выбрать набор данных для обучения. 6. Заполнить объем обучающей выборки. 7. Заполнить параметры обучения. 8. Нажать на кнопку «Подтвердить».	Модель выбранного алгоритма обучилась на заданных параметрах и объеме обучающей выборке выбранного набора данных, а затем была сохранена в базу данных и объектное хранилище.	Да

Продолжение таблицы 2 приложения В

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
5	Настройки обучения	1. Выбрать в боковой панели вкладку «Обучение». 2. Нажать на кнопку «Добавить». 3. Выбрать алгоритм для обучения. 4. Выбрать другой алгоритм для обучения.	Пересобрана форма параметров обучения.	Да
6	Скачивание обученной модели	1. Выбрать в боковой панели вкладку «Обучение». 2. Нажать на кнопку скачивания в строке таблицы у обученной модели.	Скачан файл в формате pkl с сериализованной обученной моделью.	Да
7	Удаление обученных моделей	1. Выбрать в боковой панели вкладку «Обучение». 2. Заполнить чекбоксы в строках таблицы у обученных моделей. 3. Нажать на кнопку «Удалить». 4. В появившемся окне нажать на кнопку «Удалить».	Обученные модели удалены из таблицы, базы данных и объектного хранилища. Все связанные с удаленными моделями классификации удалены из базы данных, а для команд, присутствующих в удаленных классификациях пересчитаны результаты совокупной классификации, которые затем обновятся в таблице.	Да
8	Классификация команд	1. Выбрать в боковой панели вкладку «Классификация». 2. Нажать на кнопку «Добавить». 3. В появившемся окне нажать на кнопку «Выбрать файл». 4. Выбрать файла в формате CSV, содержащий предобработанные команды. 5. Выбрать обученные модели для классификации. 6. Нажать на кнопку «Подтвердить».	Предыдущие классификации, если они есть, удаляются из таблицы. Появляются новые результаты в количестве, указанном внизу таблицы.	Да

Окончание таблицы 2 приложения В

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
9	Подробные результаты классификации команды	1. Выбрать в боковой панели вкладку «Классификация». 2. Нажать на строку таблицы с результатами классификации команды.	Для выбранной команды должна появиться карточка с подробными результатами классификации каждой выбранной обученной модели.	Да
10	Сохранение результатов классификации	1. Выбрать в боковой панели вкладку «Классификация». 2. Нажать на кнопку «Сохранить результаты».	Результаты подробной и общей классификации каждой команды сформированы в файл в формате CSV и скачаны.	Да