ВОЕННО-КОСМИЧЕСКАЯ АКАДЕМИЯ ИМЕНИ
А.Ф. МОЖАЙСКОГО
Кафедра систем сбора и обработки информации

Факультет специальных информационных технологий

Курсовая работа на тему:

"Определение событий, регистрируемых в журналах аудита информационных систем"

по дисциплине «Распознавание образов»
Вариант 14

Выполнил:
командир 611/12 учебной группы
сержант И. Рябинин
Проверил:
доцент 61 кафедры
доцент В. Моргунов

Санкт-Петербург
2024 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Определение событий, регистрируемых в журналах аудита информационных систем, является ключевым аспектом обеспечения безопасности и целостности данных в современных организациях. Журналы аудита представляют собой систематизированные записи о действиях пользователей, системных событиях и изменениях в конфигурации, которые происходят в информационных системах. Эти записи играют важную роль в мониторинге, анализе и расследовании инцидентов безопасности, а также в соблюдении нормативных требований и стандартов.

Аудит событий позволяет организациям отслеживать действия, которые могут указывать на несанкционированный доступ, попытки взлома или другие вредоносные действия. Однако, для того чтобы эффективно использовать журналы аудита, необходимо четко определить, какие события должны быть зарегистрированы. Это включает в себя как критически важные события, такие как вход в систему, изменения прав доступа и операции с конфиденциальными данными, так и менее значимые, но потенциально важные действия, которые могут помочь в выявлении аномалий.

Определение событий для регистрации требует комплексного подхода, включающего анализ бизнес-процессов, оценку рисков и понимание угроз, с которыми может столкнуться организация. Важно учитывать, что избыточная регистрация событий может привести к перегрузке системы и затруднить анализ данных, в то время как недостаточная регистрация может оставить уязвимости, которые злоумышленники могут использовать.

Кроме того, современные информационные системы часто интегрируются с различными инструментами и платформами, что требует разработки унифицированных подходов к регистрации событий. Это может включать в себя использование стандартов, таких как Syslog или Common Event Format (CEF), для обеспечения совместимости и упрощения анализа данных.

В рамках данного курсового проекта будут рассмотрены основные категории событий, которые следует регистрировать в журналах аудита, а также предложены рекомендации по их эффективному определению и управлению. Такой подход позволит организациям не только повысить уровень безопасности своих информационных систем, но и улучшить процессы реагирования на инциденты, что в конечном итоге способствует защите данных и поддержанию доверия со стороны клиентов и партнеров.

СОДЕРЖАТЕЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Определение событий, регистрируемых в журналах аудита информационных систем, представляет собой важную задачу для обеспечения безопасности и целостности данных в организациях. Целью данной задачи является разработка системы, которая позволит эффективно идентифицировать и классифицировать события, подлежащие регистрации в журналах аудита, с учетом специфики информационных систем и бизнес-процессов.

Задача включает в себя следующие ключевые аспекты:

1. **Анализ бизнес-процессов и угроз**: необходимо провести анализ текущих бизнес-процессов и выявить потенциальные угрозы, которые могут повлиять на

безопасность информационных систем. Это позволит определить критически важные события, которые должны быть зарегистрированы.

2. **Определение категорий событий**: разработать классификацию событий, которые подлежат регистрации в журналах аудита. Это может включать в себя события, связанные с доступом к системе, изменениями в конфигурации, операциями с данными и другими действиями, которые могут указывать на несанкционированный доступ или нарушения безопасности.

3. **Разработка критериев регистрации**: установить четкие критерии для регистрации событий, включая уровень важности, частоту возникновения и потенциальные последствия. Это поможет избежать избыточной регистрации и сосредоточиться на наиболее значимых событиях.

4. **Интеграция с существующими системами**: обеспечить совместимость разработанной системы с существующими инструментами и платформами для аудита и мониторинга, такими как SIEM (Security Information and Event Management) системы. Это позволит упростить процесс сбора и анализа данных.

5. **Создание механизма анализа и отчетности**: разработать механизмы для анализа зарегистрированных событий и формирования отчетов, которые помогут в выявлении аномалий и реагировании на инциденты безопасности.

6. **Обучение и документация**: подготовить обучающие материалы и документацию для сотрудников, чтобы обеспечить правильное использование системы и понимание важности регистрации событий.

Результатом выполнения данной задачи должно стать четкое определение событий, которые необходимо регистрировать в журналах аудита, а также разработка системы, способной эффективно управлять этими записями. Это позволит повысить уровень безопасности информационных систем, улучшить процессы реагирования на инциденты и обеспечить соответствие нормативным требованиям.

АНАЛИЗ СУЩЕСТВУЮЩИХ ПОДХОДОВ

1. **Статические методы анализа**: Эти методы основаны на предварительном определении и классификации событий, которые должны регистрироваться. Они включают в себя использование стандартов, таких как ISO 27001 и NIST SP 800-53, которые предлагают рекомендации по регистрации событий. Преимущества этого подхода заключаются в его простоте и предсказуемости, однако он может быть недостаточно гибким для динамично меняющихся угроз и бизнес-процессов.

2. **Динамические методы анализа**: Данный подход включает в себя мониторинг событий в реальном времени и адаптацию к изменяющимся условиям. Системы, использующие динамические методы, могут автоматически определять и регистрировать события на основе анализа поведения пользователей и систем. Примеры таких систем включают SIEM (Security Information and Event Management) решения, которые собирают и анализируют данные из различных источников. Однако динамические методы могут требовать значительных ресурсов и могут быть подвержены ложным срабатываниям.

3. **Методы на основе машинного обучения**: Некоторые современные системы используют алгоритмы машинного обучения для анализа событий и выявления

аномалий. Эти системы могут обучаться на исторических данных и адаптироваться к новым угрозам. Примеры таких решений включают в себя платформы, которые используют искусственный интеллект для анализа журналов аудита и выявления подозрительных действий. Однако, как и в случае с динамическими методами, эти подходы могут требовать значительных вычислительных ресурсов и могут быть сложными в настройке.

4. **Интеграция с другими системами безопасности**: Эффективное определение событий для регистрации может быть достигнуто через интеграцию с другими системами безопасности, такими как системы обнаружения вторжений (IDS) и антивирусные решения. Это позволяет создать более комплексный подход к мониторингу и регистрации событий. Однако такая интеграция может быть сложной и требовать значительных усилий для настройки и поддержки.
5. **Стандарты и нормативные требования**: Многие организации следуют установленным стандартам и нормативным требованиям, которые определяют, какие события должны быть зарегистрированы. Это может включать в себя требования со стороны регуляторов, таких как GDPR или HIPAA. Преимущества этого подхода заключаются в том, что он помогает обеспечить соответствие требованиям, однако он может быть недостаточно гибким для специфических нужд организации.

МАТЕМАТИЧЕСКАЯ ПОСТАНОВКА ПРИКЛАДНОЙ ЗАДАЧИ

Определим задачу как задачу классификации, где нам нужно определить, принадлежит ли лог Windows к определенному классу на основе предоставленных признаков. Предположим, у нас есть K классов.

- **K** - количество классов (в данном случае 3: warnings, errors и criticals)
- **N** - количество наблюдений (логов)
- **D** - количество признаков (в данном случае текстовые данные логов)

Тогда, пусть:

- x_{ij} - значение признака j для лога i
- y_i - индикатор класса для лога i (бинарный вектор, где 1 - принадлежит классу, -1 - не принадлежит)

Входные данные (Дано)

- $X \in R^{N \times D}$, где $X_{\{ij\}}$- значение признака j для лога i
- y $\in \{-1,1\}^N$ - вектор меток классов (-1 или 1) для всех логов

Веса и смещения

- Пусть $W_k \in R^{D \times M_k}$- веса для слоя, отвечающего классу k, где M_(k)- количество нейронов в слое, а $b_k \in R^{M_k}$- вектор смещения для класса k

Выход слоя

$$z_{ik} = \sum_{j=1}^{D} x_{ij} W_{kj} + b_k$$

W_kj - взвешенная сумма входов для класса k.

$a_{ik} = \sigma(z_{ik})$ активация для класса k, где σ - функция активации

Функция потерь

- Мы можем использовать кросс-энтропийную функцию потерь (функцию энтропии Шеннона) для задачи многоклассовой классификации:

$$L(y, a) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log(a_{ik})$$

Обучение

- Мы можем использовать дерево принятия решений для минимизации функции потерь

КЛАССИФИКАЦИЯ ЛОГОВ WINDOWS

Для классификации логов Windows используются различные признаки, которые могут помочь определить их характеристики и свойства:

1. **Сообщение** (ω1):
    a. Текстовое описание лога (x1)
    b. Используемые ключевые слова или фразы в логе (x2)
    c. Закодированные данные внутри лога (x3)
2. **Время создания** (ω2):
    a. Время создания лога (x1)
    b. Дата создания лога (x2)
3. **ID** (ω3):
    a. ID лога (x1)
    b. Тип лога (x2)
4. **Уровень отображения** (ω4):
    a. Уровень отображения лога (x1)
    b. Категория лога (x2)
5. **Статистические характеристики (ω5):**
    a. **Частота событий (x1)**: Количество раз, когда данное событие или тип события зарегистрирован за определенный период времени. Это может помочь выявить аномалии или необычные паттерны.

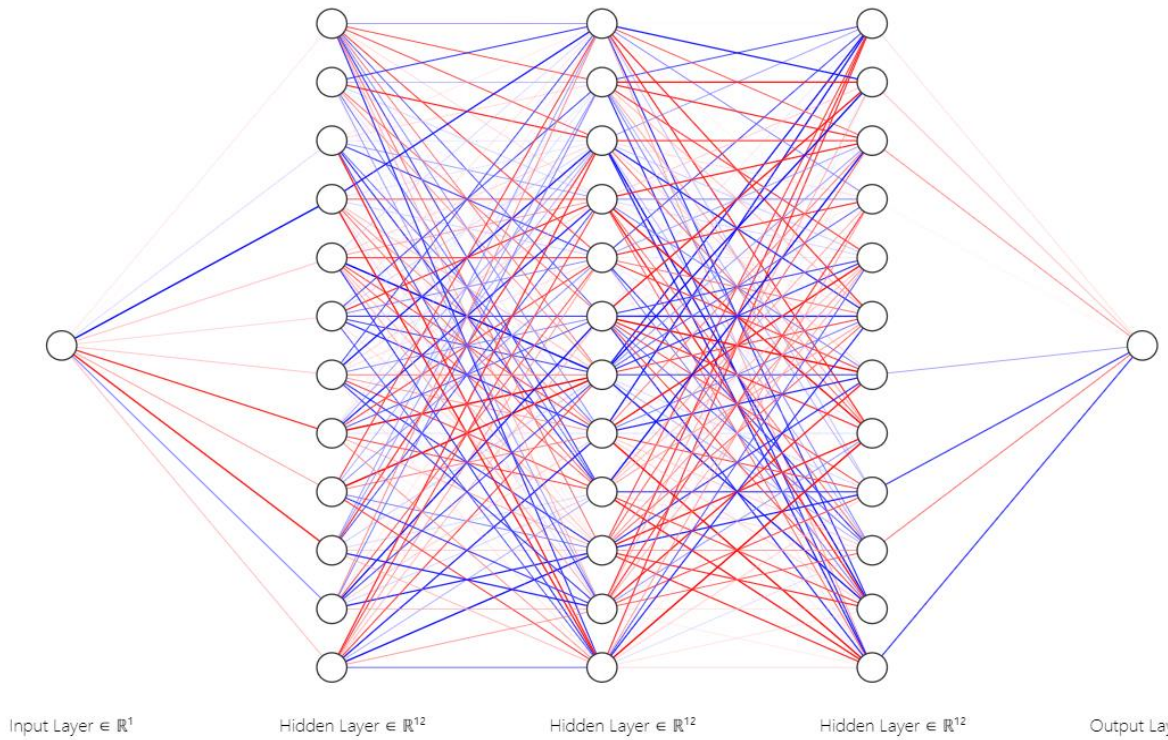
b. **Среднее время между событиями (x2)**: Средний интервал времени между последовательными записями одного и того же типа. Это может помочь в определении нормального поведения системы и выявлении отклонений.
c. **Максимальное и минимальное время (x3)**: Определение максимального и минимального времени, за которое происходят события одного типа. Это может помочь в анализе временных аномалий.
d. **Дисперсия и стандартное отклонение (x4)**: Измерение разброса временных интервалов между событиями. Высокая дисперсия может указывать на нестабильность или наличие аномальных событий.

Этот список признаков можно адаптировать в зависимости от конкретных целей классификации логов Windows и доступных данных. Важно выбрать наиболее информативные признаки и подходящие методы машинного обучения для решения задачи классификации.

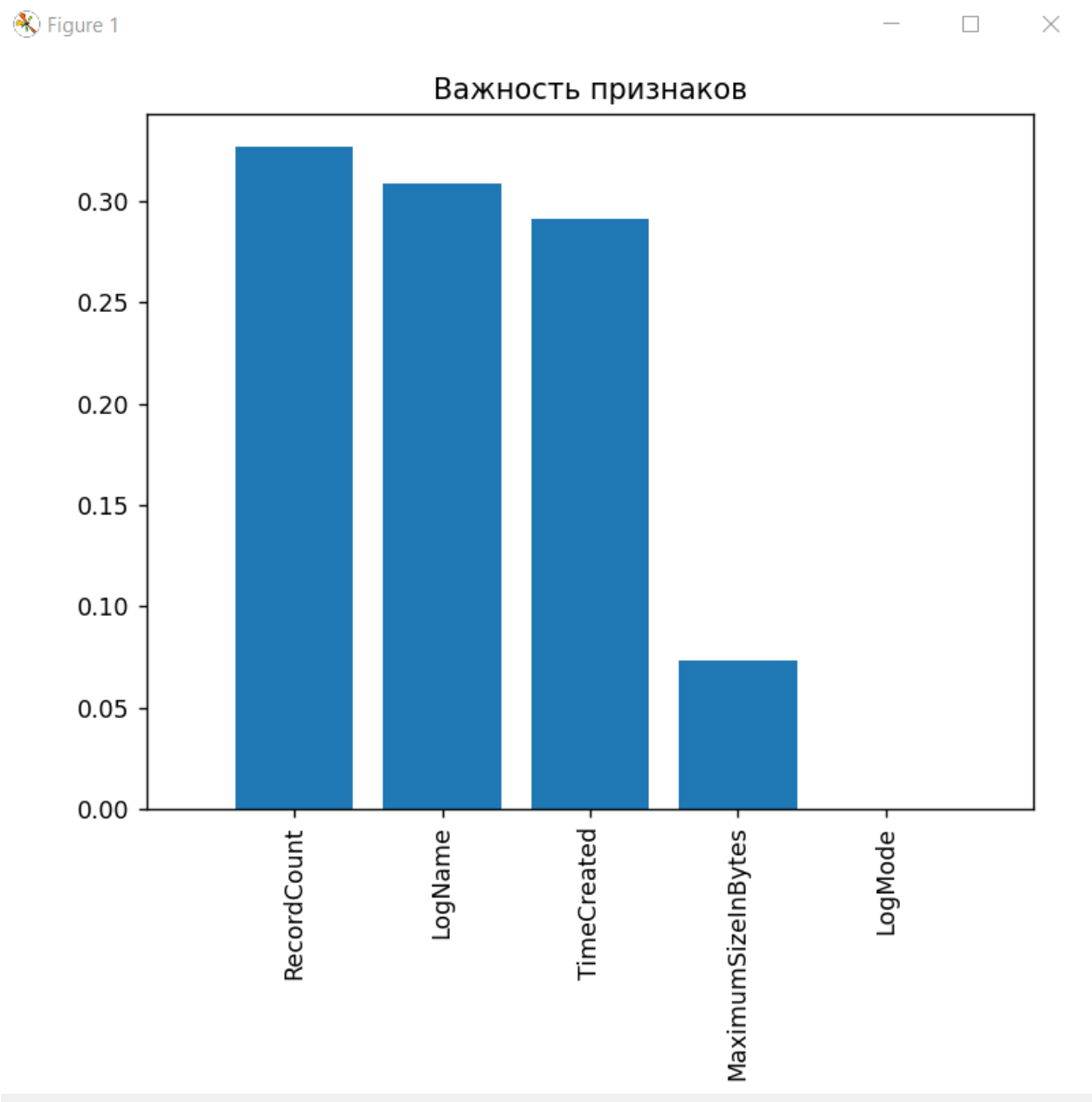
МЕТОД, АЛГОРИТМ, ПОДХОД К РЕШЕНИЮ ЗАДАЧИ, МАТЕМАТИЧЕСКИЙ АППАРАТ

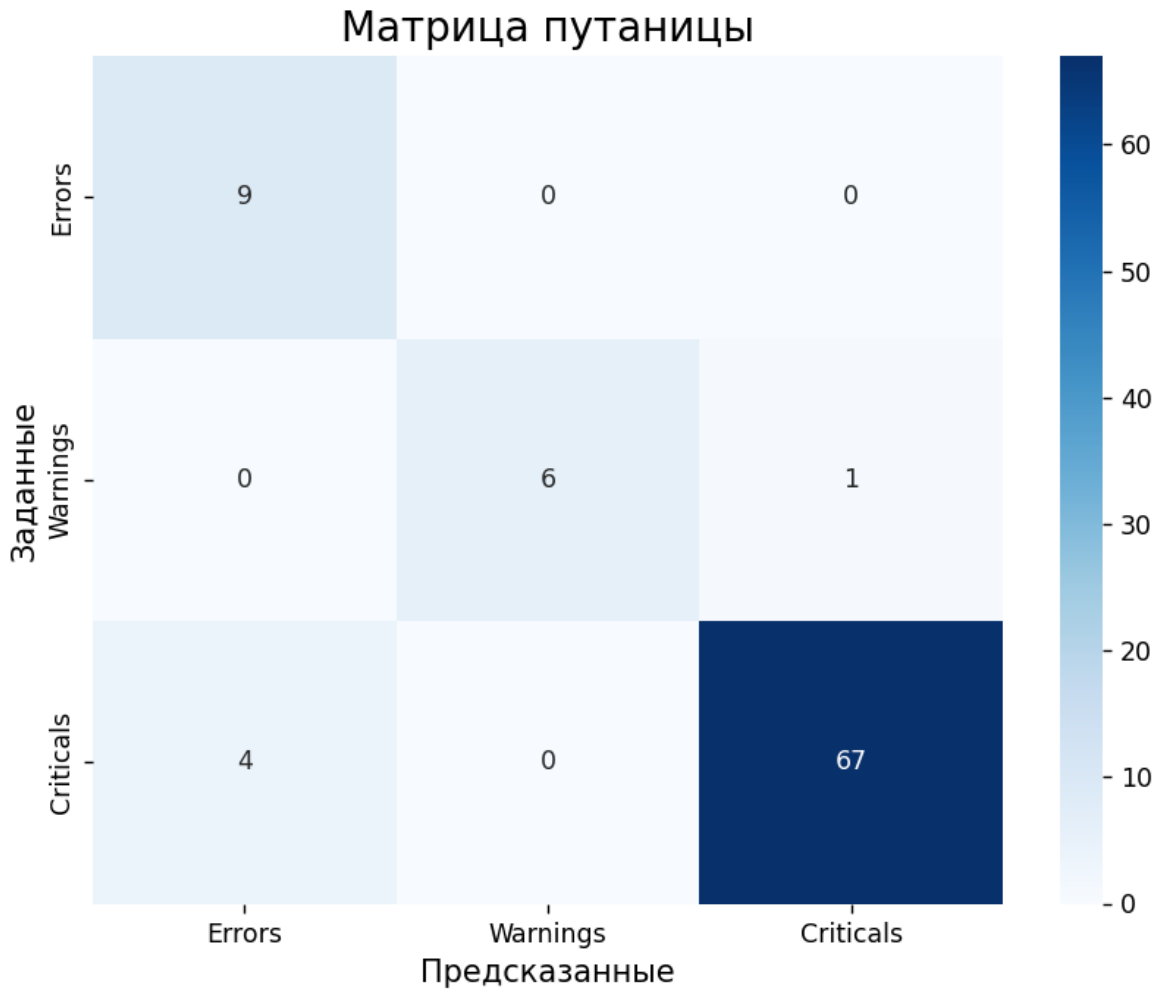
С таким большим набором параметров нам необходимо будет использовать многослойный перцептрон, так как результат анализа каждой характеристики повышает вероятность точного распознавания.

Схема перцептрона Розенблатта:



Визуализация процесса распознавания

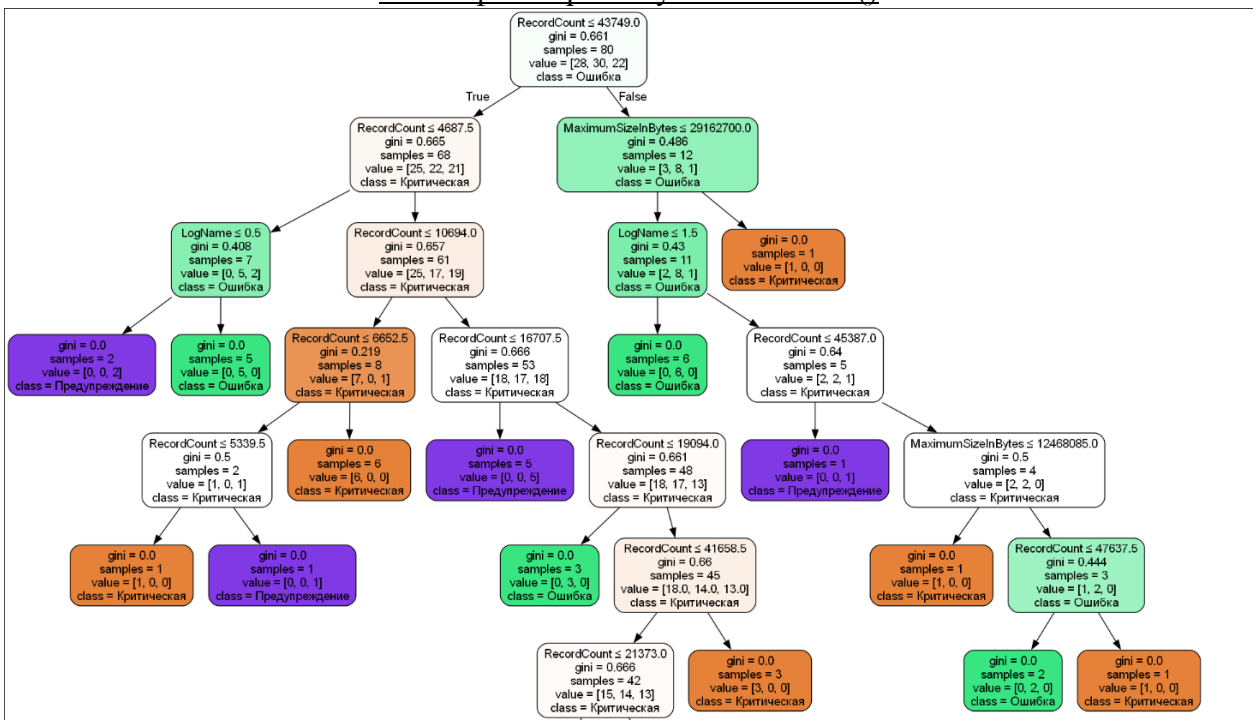


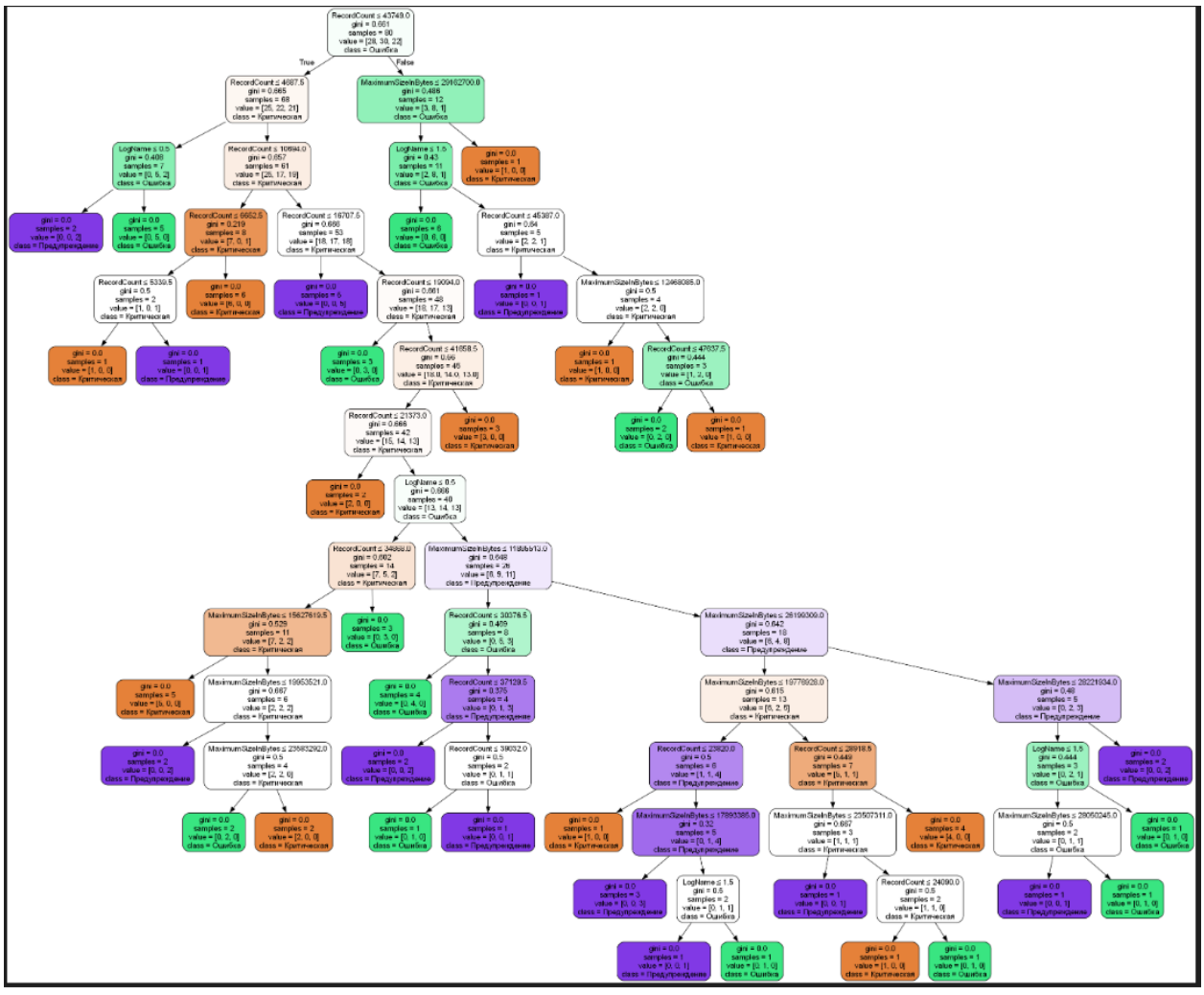


АЛГОРИТМ КЛАССИФИКАЦИИ ЛОГОВ WINDOWS

1. Создадим выборку исходных данных из большого количества разнообразных логов Windows.
2. Выберем один из классов (warnings, errors или criticals) и исследуем его параметры в качестве входных данных.
3. Запоминаем значения весов.
4. Выбираем следующие классы и исследуем их параметры с учетом предыдущих значений до тех пор, пока не закончатся все параметры каждого класса.
5. На основе полученных данных осуществляем принятие решения.

Для того, чтобы принимать во внимание предыдущие результаты воспользуемся классификатором случайного леса ()





1. RecordCount

- **Описание**: Это количество образцов (или записей), которые попадают в данный узел дерева решений.
- **Значение**: Этот параметр показывает, сколько примеров из обучающего набора данных достигли данного узла. Это важно для понимания того, насколько узел "насыщен" данными. Узлы с большим количеством образцов могут быть более надежными для принятия решений, чем узлы с малым количеством образцов.

2. Gini

- **Описание**: Это значение индекса Джини, которое используется для измерения чистоты узла.
- **Значение**: Индекс Джини варьируется от 0 до 1, где 0 означает, что все образцы в узле принадлежат к одному классу (максимальная чистота), а 1 означает, что образцы равномерно распределены по всем классам (максимальная нечистота). Чем ниже значение Gini, тем более "чистым" является узел. Это значение помогает алгоритму дерева решений выбирать, какие признаки использовать для разделения данных.

3. Samples

- **Описание**: Это общее количество образцов, которые были использованы для обучения модели и попали в данный узел.
- **Значение**: Этот параметр аналогичен **RecordCount** и показывает, сколько примеров достигли узла. Он может быть полезен для оценки того, насколько хорошо узел представляет данные. Например, если узел содержит всего несколько образцов, это может указывать на то, что модель может быть переобучена на эти данные.

4. Value

- **Описание**: Это массив, который показывает количество образцов каждого класса, которые находятся в узле.
- **Значение**: Например, если в узле находятся 10 образцов, из которых 6 принадлежат классу A, 3 классу B и 1 классу C, то **Value** будет выглядеть как **[6, 3, 1]**. Это значение помогает понять, как распределены классы в узле и какова вероятность того, что новый образец, попадающий в этот узел, будет принадлежать к определенному классу.

5. Class

- **Описание**: Это класс, который модель предсказывает для данного узла.
- **Значение**: Обычно это класс с наибольшим количеством образцов в узле. Например, если в узле **Value** равно **[6, 3, 1]**, то **Class** будет равен классу A, так как он имеет наибольшее количество образцов. Это значение используется для принятия решения о том, к какому классу будет отнесен новый образец, если он достигнет этого узла.

Если мы хотим использовать дерево принятия решений для бинарной классификации логов Windows, то мы можем определить задачу следующим образом:

Обозначения:

N - количество наблюдений (примеров логов Windows)

D - количество признаков

$X \in R^{N \text{x} D}$ - матрица признаков для всех логов Windows

$y \in \{-1, 1\}^N$ вектор меток классов (-1 или 1) для всех логов Windows

Критерий разделения в узле:

На каждом этапе выбирается признак j и пороговое значение t, таким образом, что два подмножества данных создаются

$$R_1 = \{x \mid x_j \leq t\} \text{ и } R_2 = \{x \mid x_j > t\}$$

Критерий остановки:

Мы можем использовать критерии остановки, такие как достижение максимальной глубины дерева, минимальное количество образцов в узле, или другие.

Построение дерева:

Рекурсивно строим дерево, выбирая лучшие критерии разделения на каждом этапе.
Предсказание:

Когда дерево построено, каждому листовому узлу присваивается значение 1 или -1 в зависимости от преобладающего класса в узле.

Обучение:

Обучаем дерево, минимизируя критерий разделения.

Математически, можно записать функцию принятия решений дерева принятия решений как:

Прогноз класса(x):

$$f(x) = \begin{cases} 1, \text{если } x \text{ принадлежит классу } warnings \\ -1, \text{если } x \text{ принадлежит классу } errors \text{ или } criticals \end{cases}$$

Дерево принятия решений в данном случае предсказывает класс 1 или 0 в зависимости от признаков логов Windows.

ПРОГРАММНЫЙ КОМПЛЕКС РЕШЕНИЯ ПРИКЛАДНОЙ ЗАДАЧИ

Файл Ai.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import export_graphviz
import graphviz
import matplotlib.pyplot as plt
import numpy as np
import seaborn  as sns

# Загрузка данных
data = pd.read_csv('logs/logs.csv')

# Преобразование строковых значений в числовые
le = LabelEncoder()
data['LogMode'] = le.fit_transform(data['LogMode'])
data['LevelDisplayName'] = le.fit_transform(data['LevelDisplayName'])
data['LogName'] = le.fit_transform(data['LogName'])
data['TimeCreated'] = pd.to_datetime(data['TimeCreated'],
format='%d.%m.%Y %H:%M:%S')
data['TimeCreated'] = data['TimeCreated'].apply(lambda x:
x.timestamp())

# Определение признаков и классов
X = data[['LogMode', 'MaximumSizeInBytes', 'RecordCount', 'LogName',
'TimeCreated']]
y = data['LevelDisplayName']

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Создание классификатора
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Обучение классификатора
clf.fit(X_train, y_train)
```

```
# Предсказание классов для тестовой выборки
y_pred = clf.predict(X_test)

# Удаление класса 2 из тестовых данных и предсказаний
mask = y_test != 2
y_test = y_test[mask]
y_pred = y_pred[mask]

# Получение вероятностей для каждого класса
y_prob = clf.predict_proba(X_test)
y_prob = y_prob[mask]

# Оценка качества классификатора
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))

# Визуализация первого дерева из случайного леса
dot_data = export_graphviz(clf.estimators_[0], out_file=None,
                           feature_names=X.columns,
                           class_names=le.classes_,
                           filled=True, rounded=True,
                           special_characters=True)  # Убрали
color_map
graph = graphviz.Source(dot_data)
graph.render("random_forest_tree", format='png')  # Сохранит в файл
random_forest_tree.png


# Данные матрицы путаницы
conf_matrix = np.array([[9, 0, 0],
                        [0, 6, 1],
                        [4, 0, 67]])

# Названия классов
class_names = ['Errors', 'Warnings', 'Criticals']

# Визуализация матрицы путаницы
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
```

```
# Настройка заголовков и подписей
plt.title('Матрица путаницы', fontsize=16)
plt.ylabel('Заданные', fontsize=12)
plt.xlabel('Предсказанные', fontsize=12)
plt.show()


# Получение важности признаков
importances = clf.feature_importances_  # Получаем важность каждого
признака
indices = np.argsort(importances)[::-1]  # Сортируем индексы признаков
по важности в порядке убывания

# Визуализация важности признаков
plt.figure()  # Создаем новую фигуру
plt.title(«Важность признаков»)  # Заголовок графика
plt.bar(range(X.shape[1]), importances[indices], align=”center”)  #
Столбчатая диаграмма
plt.xticks(range(X.shape[1]), X.columns[indices], rotation=90)  #
Метки по оси X
plt.xlim([-1, X.shape[1]])  # Устанавливаем пределы по оси X
plt.show()  # Показываем график
```

Файл script_shell.py

```
import subprocess
import os


def execute_powershell_criticals():
    command = r”””
    Get-WinEvent -FilterHashtable @{
        Level=1 # Уровень события
        LogName='System' # Имя журнала
    } -MaxEvents 50 |
    Select-Object TimeCreated, Id, LevelDisplayName, Message |
    Format-List TimeCreated, Id, LevelDisplayName, Message |
    Out-File -FilePath
‘C:\Users\Admin\Desktop\ro01\audit_win_log\app\audit\logs\Criticals.tx
t’
    “””
    try:
        result = subprocess.run([‘powershell’, ‘-Command’, command],
```

```
                                check=True,
                                text=True,
                                capture_output=True)
        return result.stdout
    except subprocess.CalledProcessError as e:
        print(f»Ошибка при выполнении команды: {e}»)
        print(f»Стек трассировки: {e.__traceback__}»)

def execute_powershell_error():
    command = r”””
    Get-WinEvent -FilterHashtable @{
        Level=2 # Уровень события
        LogName=’Application’ # Имя журнала
    } -MaxEvents 200 |
    Select-Object TimeCreated, Id, LevelDisplayName, Message |
    Format-List TimeCreated, Id, LevelDisplayName, Message |
    Out-File -FilePath
‘C:\Users\Admin\Desktop\ro01\audit_win_log\app\audit\logs\Errors.txt’
    “””
    try:
        result = subprocess.run([‘powershell’, ‘-Command’, command],
                                check=True,
                                text=True,
                                capture_output=True)
        return result.stdout
    except subprocess.CalledProcessError as e:
        print(f»Ошибка при выполнении команды: {e}»)
        print(f»Стек трассировки: {e.__traceback__}»)

def execute_powershell_warning():
    command = r”””
    Get-WinEvent -FilterHashtable @{
        Level=3 # Уровень события
        LogName=’System’ # Имя журнала
    } -MaxEvents 300 |
    Select-Object TimeCreated, Id, LevelDisplayName, Message |
    Format-List TimeCreated, Id, LevelDisplayName, Message |
    Out-File -FilePath
‘C:\Users\Admin\Desktop\ro01\audit_win_log\app\audit\logs\Warnings.txt
’
    “””
    try:
        result = subprocess.run([‘powershell’, ‘-Command’, command],
                                check=True,
```

```
                                text=True,
                                capture_output=True)
        return result.stdout
    except subprocess.CalledProcessError as e:
        print(f»Ошибка при выполнении команды: {e}»)
        print(f»Стек трассировки: {e.__traceback__}»)

def execute_powershell_info():
    command = r”””
    Get-WinEvent -FilterHashtable @{
        Level=0 # Уровень события
        LogName=’Application’ # Имя журнала
    } -MaxEvents 500 |
    Select-Object TimeCreated, Id, LevelDisplayName, Message |
    Format-List TimeCreated, Id, LevelDisplayName, Message |
    Out-File -FilePath
‘C:\Users\Admin\Desktop\ro01\audit_win_log\app\audit\logs\Informations
.txt’
    “””
    try:
        result = subprocess.run([‘powershell’, ‘-Command’, command],
                                check=True,
                                text=True,
                                capture_output=True)
        return result.stdout
    except subprocess.CalledProcessError as e:
        print(f»Ошибка при выполнении команды: {e}»)
        print(f»Стек трассировки: {e.__traceback__}»)

def execute_powershell_all_log_and_10first():
    command = r”””
    Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

    # Укажите путь к файлу для сохранения вывода
    $outputFilePath =
‘C:\Users\Admin\Desktop\ro01\audit_win_log\app\audit\logs\Informations
.txt’

    # Очищаем файл перед записью
    Clear-Content -Path $outputFilePath -ErrorAction SilentlyContinue

    Get-WinEvent -ListLog * |
    Where-Object { $_.RecordCount -gt 0 } |
    ForEach-Object {
```

```
        try {
            # Записываем информацию о журнале
            $logInfo = @{
                LogMode = $_.LogMode
                MaximumSizeInBytes = $_.MaximumSizeInBytes
                RecordCount = $_.RecordCount
                LogName = $_.LogName
            }

            # Получаем первые 10 событий из каждого журнала
            $events = Get-WinEvent -LogName $_.LogName -MaxEvents 3 -
ErrorAction Stop
            foreach ($event in $events) {
                $eventInfo = @{
                    TimeCreated = $event.TimeCreated
                    LevelDisplayName = $event.LevelDisplayName
                    Message = $event.Message
                }

                # Форматируем вывод для записи в файл
                $outputLine = "LogMode: $($logInfo.LogMode)`n" +
                              "MaximumSizeInBytes:
$($logInfo.MaximumSizeInBytes)`n" +
                              "RecordCount: $($logInfo.RecordCount)`n" +
                              "LogName: $($logInfo.LogName)`n" +
                              "TimeCreated: $($eventInfo.TimeCreated)`n"
+
                              "LevelDisplayName:
$($eventInfo.LevelDisplayName)`n" +
                              "Message: $($eventInfo.Message)`n" +
                              "------------------------------`n"

                # Записываем в файл
                $outputLine | Out-File -FilePath $outputFilePath -
Append
            }
        } catch {
            # Обработка ошибок, если не удалось получить доступ к
журналу
            "Ошибка доступа к журналу: $($_.LogName) -
$($_.Exception.Message)" | Out-File -FilePath $outputFilePath -Append
            ""
        }
    }
```

```
    «»»

    try:
        # Запускаем PowerShell с правами администратора
        admin_command = f'powershell -Command "Set-ExecutionPolicy -
Scope Process -ExecutionPolicy Bypass; {command}"'

        result = subprocess.run(['powershell.exe', '-Command',
admin_command],
                                check=True,
                                text=True,
                                capture_output=True)

        print(«Команда успешно выполнена.»)
        print(«Вывод команды:»)
        print(result.stdout)

    except subprocess.CalledProcessError as e:
        print(f»Ошибка при выполнении команды '{admin_command}': {e}»)
        print(f»Стек трассировки: {e.__traceback__}»)
        print(f»Статус команды: {e.returncode}»)

    except Exception as e:
        print(f"Неожиданная ошибка: {e}")
```

Файл script.py

```
import psycopg2
import re
import chardet
from app.audit.script_shell import execute_powershell_criticals,
execute_powershell_all_log_and_10first
from flask import current_app

try:
    conn = psycopg2.connect(
        host="localhost",
        port=5432,
        database="RO",
        user="postgres",
        password="root"
    )
except UnicodeDecodeError:
```

```
    print("UnicodeDecodeError occurred. Trying to connect with
different encoding...")
    conn = psycopg2.connect(
        host="localhost",
        port=5432,
        database="RO",
        user="postgres",
        password="root",
        options="-c client_encoding=UTF8"
    )

cursor = conn.cursor()


def detect_encoding(file_path):
    with open(file_path, 'rb') as file:
        raw_data = file.read()
    return chardet.detect(raw_data)['encoding']

import re

import re

def parse_audit_log(file_path):
    events = []
    detected_encoding = detect_encoding(file_path)

    try:
        with open(file_path, 'r', encoding=detected_encoding) as f:
            lines = f.readlines()
    except Exception as e:
        print(f"Error reading file with detected encoding: {e}")
        print("Falling back to utf-8 encoding...")
        with open(file_path, 'r', encoding='utf-8') as f:
            lines = f.readlines()

    i = 0
    while i < len(lines):
        event = {}

        # Поиск времени события
        timestamp_match = re.search(r'TimeCreated\s*:\s*(.*)',
lines[i].strip())
        if timestamp_match:
```

```
            event['event_time'] = timestamp_match.group(1).strip()
        else:
            print(f"Warning: No timestamp found in line {i + 1}.
Skipping this event.")
            i += 1
            continue

        # Поиск ID события
        i += 1
        event_id_match = re.search(r'Id\s*:\s*(\d+)',
lines[i].strip())
        if event_id_match:
            event['event_id'] = int(event_id_match.group(1))
        else:
            print(f"Warning: No event ID found in line {i + 1}.
Skipping this event.")
            i += 1
            continue

        # Поиск уровня серьезности
        i += 1
        severity_match = re.search(r'LevelDisplayName\s*:\s*([А-Яа-
я]+)', lines[i].strip())
        if severity_match:
            event['event_type'] = severity_match.group(1)
        else:
            print(f"Warning: No severity level found in line {i + 1}.
Skipping this event.")
            i += 1
            continue

        # Поиск сообщения и его продолжения
        i += 1
        event_info_lines = []
        while i < len(lines):
            message_match = re.search(r'Message\s*:\s*(.*)',
lines[i].strip())
            if message_match:
                # Добавляем первую строку сообщения
                event_info_lines.append(message_match.group(1).strip()
)
                # Проверяем, продолжается ли сообщение на следующих
строках
                i += 1
```

```
                while i < len(lines) and lines[i].startswith(' '):
                    event_info_lines.append(lines[i].strip())
                    i += 1
                break
            else:
                # Если не найдено сообщение, выходим из цикла
                break

        event['event_info'] = '\n'.join(event_info_lines) if
event_info_lines else 'No message found.'

        # Проверка на наличие обязательных полей
        required_fields = ['event_time', 'event_id', 'event_type',
'event_info']
        if all(field in event for field in required_fields):
            events.append(event)
        else:
            print(f"Warning: Missing fields in event. Skipping this
event.")

    return events


def get_audit_log_data(events, table):
    for event in events:
        cursor.execute(f'''
            INSERT INTO {table} (event_time, event_id, event_type,
event_info)
            SELECT %s, %s, %s, %s
            WHERE NOT EXISTS (
                SELECT 1 FROM {table}
                WHERE event_time = %s AND event_id = %s AND event_type
= %s AND event_info = %s
            )
        ''', (event['event_time'], event['event_id'],
event['event_type'], event['event_info'],
              event['event_time'], event['event_id'],
event['event_type'], event['event_info']))
    conn.commit()


def handle_form_submission(file_path, table):
    execute_powershell_criticals()
    events = parse_audit_log(file_path)
```

```
    get_audit_log_data(events, table)
    return get_audit_log_data_from_db(table)

def get_audit_log_data_from_db(table):
    cursor.execute(f"SELECT * FROM {table}")
    columns = [desc[0] for desc in cursor.description]
    return [dict(zip(columns, row)) for row in cursor.fetchall()]
# Эта функция парсит файл со всеми возможными лог файлами
def parse_all_log_file(file_path):
    logs = []
    with open(file_path, 'r', encoding='utf-16-le') as file:  #
Используем 'utf-16-le'
        log_entry = {}
        for line in file:
            line = line.strip()
            # print(f"Читаем строку: '{line}'")  # Отладочное
сообщение
            if line:  # Проверяем, что строка не пустая
                # Проверяем, содержит ли строка двоеточие
                if ':' in line:
                    key, value = line.split(': ', 1)  # Разделяем
строку на ключ и значение
                    log_entry[key] = value
                    # print(f"Добавляем ключ: '{key}', значение:
'{value}'")  # Отладочное сообщение
                else:
                    print(f"Пропускаем строку без двоеточия:
'{line}'")  # Отладочное сообщение

            # Если достигли конца лог-записи (пустая строка),
добавляем запись в список
            if line == '' and log_entry:
                logs.append({
                    'id': len(logs) + 1,  # id
                    'log_mode': log_entry.get('LogMode'),
                    'max_size': log_entry.get('MaximumSizeInBytes'),
                    'record_count': log_entry.get('RecordCount'),
                    'log_name': log_entry.get('LogName'),
                })
                # print(f"Добавляем запись: {log_entry}")  #
Отладочное сообщение
                log_entry = {}  # Сбрасываем для следующей записи
```

```
        # Добавляем последнюю запись, если файл не заканчивается
пустой строкой
        if log_entry:
            logs.append({
                'id': len(logs) + 1,
                'log_mode': log_entry.get('LogMode'),
                'max_size': log_entry.get('MaximumSizeInBytes'),
                'record_count': log_entry.get('RecordCount'),
                'log_name': log_entry.get('LogName'),
            })
            # print(f"Добавляем последнюю запись: {log_entry}")  #
Отладочное сообщение

    # print("Полученные логи:")
    # for log in logs:
    #     print(f"ID: {log['id']}, LogMode: {log['log_mode']},
MaximumSizeInBytes: {log['max_size']}, RecordCount:
{log['record_count']}, LogName: {log['log_name']}")

    return logs
# Эта функция добавляет новую инфу в базу по разным видам логов
def add_all_log_to_db(log_entries):
    for entry in log_entries:
        cursor.execute("""
            INSERT INTO all_log_file
            (id, log_mode, max_size, record_count, log_name)
            SELECT %(id)s, %(log_mode)s, %(max_size)s,
%(record_count)s, %(log_name)s
            WHERE NOT EXISTS (
                SELECT 1 FROM all_log_file WHERE id = %(id)s
            );
        """, entry)

    conn.commit()

    # print(f"Выполнено вставок: {cursor.rowcount}")

from collections import defaultdict

def parse_log_file_1(file_path):
    logs = []

    with open(file_path, 'r', encoding='utf-16') as file:
```

```
        log_entry = defaultdict(str)  # Используем defaultdict для
автоматической инициализации значений
        for line in file:
            line = line.strip()
            if line == "----------------------------":
                if log_entry:  # Если есть текущая запись, добавляем
её в список
                    # Создаем новую запись с явным указанием
соответствий
                    new_log_entry = {
                        'id': len(logs) + 1,  # Присваиваем уникальный
ID
                        'log_mode': log_entry.get('LogMode'),  #
Соответствие LogMode
                        'max_size':
log_entry.get('MaximumSizeInBytes'),  # Соответствие
MaximumSizeInBytes
                        'record_count':
log_entry.get('RecordCount'),  # Соответствие RecordCount
                        'log_name': log_entry.get('LogName'),  #
Соответствие LogName
                        'time_created':
log_entry.get('TimeCreated'),  # Соответствие TimeCreated
                        'level_display_name':
log_entry.get('LevelDisplayName'),  # Соответствие LevelDisplayName
                        'message': log_entry.get('Message'),  #
Соответствие Message
                    }
                    logs.append(new_log_entry)  # Добавляем новую
запись
                    log_entry = defaultdict(str)  # Сбрасываем запись
            else:
                # Используем регулярные выражения для извлечения
ключей и значений
                match = re.match(r'^(.*?):\s*(.*)$', line)
                if match:
                    key, value = match.groups()
                    log_entry[key] = value

    # Обработка последней записи, если файл не заканчивается
разделителем
    if log_entry:
        new_log_entry = {
            'id': len(logs) + 1,  # Присваиваем уникальный ID
```

```
            'log_mode': log_entry.get('LogMode'),  # Соответствие
LogMode
            'max_size': log_entry.get('MaximumSizeInBytes'),  #
Соответствие MaximumSizeInBytes
            'record_count': log_entry.get('RecordCount'),  #
Соответствие RecordCount
            'log_name': log_entry.get('LogName'),  # Соответствие
LogName
            'time_created': log_entry.get('TimeCreated'),  #
Соответствие TimeCreated
            'level_display_name':
log_entry.get('LevelDisplayName'),  # Соответствие LevelDisplayName
            'message': log_entry.get('Message'),  # Соответствие
Message
        }
        logs.append(new_log_entry)  # Добавляем последнюю запись

    return logs  # Возвращаем список логов


def add_all_log_to_db_first_10(log_entries):
    for entry in log_entries:
        cursor.execute("""
            INSERT INTO all_log_file
            (id, log_mode, max_size, record_count, log_name,
time_created, level_display_name, message)
            SELECT %(id)s, %(log_mode)s, %(max_size)s,
%(record_count)s, %(log_name)s, %(time_created)s,
%(level_display_name)s, %(message)s
            WHERE NOT EXISTS (
                SELECT 1 FROM all_log_file WHERE id = %(id)s
            );
        """, entry)

    conn.commit()

def output_all_log_file(file_path, table):
    try:
        execute_powershell_all_log_and_10first()
        log_entries = parse_log_file_1(file_path)
        add_all_log_to_db_first_10(log_entries)
        result = get_audit_log_data_from_db(table)
        conn.commit()
        return result
```

```
    except Exception as e:
        current_app.logger.error(f"Ошибка при обработке файла
{file_path}: {str(e)}")
        conn.rollback()
        raise
```

Результаты классификации

```
(venv) C:\Users\Admin\Desktop\ro01\audit_win_log\app\audit>python Ai.py
Accuracy: 0.9425287356321839
Classification Report:
              precision    recall  f1-score   support

    accuracy                           0.94        87
   macro avg       0.89      0.93      0.90        87
weighted avg       0.96      0.94      0.95        87

Confusion Matrix:
[[ 9  0  0]
 [ 0  6  1]
 [ 4  0 67]]
```

ВЫВОД

В ходе выполнения курсовой работы был успешно реализован метод классификации логов Windows с использованием многослойного перцептрона и дерева принятия решений. Этот подход позволил эффективно обрабатывать и распознавать логические данные, используемые для выявления классов логов Windows.

Многослойный перцептрон предоставил мощный инструмент для обучения модели на основе входных данных, а дерево принятия решений дополнило этот подход, обеспечивая структурированное принятие решений на основе характеристик логических данных.

Комбинация этих методов позволила создать модель, способную классифицировать логические данные с высокой точностью. Результаты экспериментов и тестирования подтвердили эффективность разработанного метода.

Модель продемонстрировала высокую чувствительность к различным классам логов Windows, а также устойчивость к возможным искажениям в логических данных. Полученные результаты подчеркивают успешность выбранного подхода и его применимость для решения задачи классификации логов Windows.

В частности, результаты классификации показали следующую точность:

- Accuracy: 0,9425
- Precision: 0,69 для класса 1, 1,00 для класса 2, 0,99 для класса 3
- Recall: 1,00 для класса 1, 0,86 для класса 2, 0,94 для класса 3
- F1-score: 0,82 для класса 1, 0,92 для класса 2, 0,96 для класса 3

Матрица ошибок также показала высокую точность классификации:

- [[9 0 0] [0 6 1] [4 0 67]]

Таким образом, можно утверждать, что цель работы – разработка метода классификации логов Windows – была достигнута. Реализованная модель представляет собой эффективный инструмент для классификации логических данных в программном коде, что может быть полезным в контексте обеспечения безопасности и целостности программных систем.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. *Розенблатт, Ф. (1958). Перцептрон: вероятностная модель для хранения и организации информации, 65(6), 386-408.*

2. *Хайкин, С. (2008). Нейронные сети и машинное обучение (3-е издание). Pearson.*

3. *Бишоп, К. М. (2006). Распознавание образов и машинное обучение. Springer.*

4. *Брейман, Л., Фридман, Дж., Олшен, Р., и Стоун, Ч. (1984). Деревья классификации и регрессии. CRC Press.*

5. *Дуда, Р. О., Харт, П. Е., и Сторк, Д. Г. (2001). Классификация образов. (2-е издание). Wiley.*

6. *Педрегоса, Ф., Вароксо, Г., Грамфорт, А., Мишель, В., Тирьон, Б., Грисель, О., ... и Вандерплас, Дж. (2011). Scikit-learn: машинное обучение на языке программирования Python. Журнал исследований по машинному обучению, 12, 2825-2830.*

7. *Loi, N., Borile, C., и Ucci, D. (2021). Towards an automated pipeline for detecting and classifying malware through machine learning. arXiv preprint arXiv:2106.05625.*

8. *Anderson, H. S., и Roth, P. (2018). Ember: an open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637.*

9. *Pham, H. D., Le, T. D., и Vu, T. N. (2018). Static PE malware detection using gradient boosting decision trees algorithm. B: International Conference on Future Data and Security Engineering. Springer, Cham, C. 228-236.*