DOI 10.37882/2223-2966.2025.06-2.18

ОБНАРУЖЕНИЕ СЕТЕВЫХ АТАК С ИСПОЛЬЗОВАНИЕМ МЕТОДА СЛУЧАЙНОГО ЛЕСА В УСЛОВИЯХ НЕСБАЛАНСИРОВАННЫХ ДАННЫХ

DETECTION OF NETWORK ATTACKS USING THE RANDOM FOREST METHOD IN CONDITIONS OF UNBALANCED DATA

O. Kulikova
A. Pinigin

Summary. The article is devoted to the topic of detecting attacks using the random forest method in conditions of unbalanced data.
This article presents the application of the Random Forest machine learning method for detecting and classifying network attacks based on network traffic analysis.
The stages of preparing the selected CICIDS2017 dataset are described, including removing correlating features, balancing classes, and selecting the most significant features.
Special attention is paid to modifying the model by adjusting the class weights and optimizing the hyperparameters of the model, which made it possible to increase the accuracy of detecting rare types of attacks.
The results of the study showed the effectiveness of using ensemble methods in the task of classifying network attacks, the possibility of using a random forest in real intrusion detection systems (IDS).

Keywords: machine learning, random forest, attack classification, network traffic analysis, class balancing.

Куликова Ольга Витальевна
Доцент, Донской государственный технический университет, г. Ростов-на-Дону
Пинигин Андрей Сергеевич
Донской Государственный Технический Университет, г. Ростов-на-Дону
AndreyIAMN1Pinigin@mail.ru

Аннотация. Статья посвящена теме обнаружения атак с использованием метода случайного леса в условиях несбалансированных данных.
В данной статье представлено применение метода машинного обучения «случайный лес» (Random Forest) для обнаружения и классификации сетевых атак на основе анализа сетевого трафика.
Описаны этапы подготовки выбранного набора данных CICIDS2017, включая удаление коррелирующих признаков, балансировку классов и отбор наиболее значимых признаков.
Особое внимание уделяется модификации модели с настройкой весов классов и оптимизацией гиперпараметров модели, что позволило повысить точность распознавания редких видов атак.
Результаты исследования показали эффективность использования ансамблевых методов в задачи классификации сетевого атак, возможность применения случайного леса в реальных системах обнаружения вторжений (IDS).

Ключевые слова: машинное обучение, случайный лес, классификация атак, анализ сетевого трафика, балансировка классов.

Введение

С развитием цифровых технологий и увеличением объема передаваемой информации возрастает угроза кибератак на корпоративные и государственные информационные системы.

Одним из перспективных направлений обеспечения безопасности является использование методов машинного обучения для анализа сетевого трафика и автоматического выявления аномального поведения [1].

Особый интерес представляет применение ансамблевых алгоритмов, одним из которых является Random Forest. Ансамблевые методы демонстрируют высокую точность и устойчивость к переобучению при работе с многомерными данными [2]. Как показано в обзоре, такие подходы все чаще применяются в системах обнаружения вторжений.[6] Реализация моделей на практике часто осуществляется с помощью библиотеки scikit-learn [10].

Однако, в задачах обнаружения атак часто возникает проблема с балансом данных, которая может существенно снизить эффективность модели по редким классам атак [7].

Данная работа посвящена построению модели обнаружения атак с использованием метода случайного леса, включающая этапы препроцессинга данных, настройки весов классов и оптимизации гиперпараметров модели.

1. Набор данных

Для исследования использовался набор данных CICIDS2017 — один из наиболее популярных, открытых наборов данных, содержащий как нормальный трафик, так и вредоносный. Данные содержат метки классов атак, включая: Dos/DDos, Web Attack (SQLi, XSS, Brute Force), Botnet, PortScan, FTP/SSH Patator и другие.

На этапе объединения в единый датасет было решено исключить файл «Monday-WorkingHours.csv» содер-

жащий в себе только нормальный трафик, чтобы избежать избыточности выборки.

2. Препроцессинг данных

Перед построением модели необходимо произвести препроцессинг данных, с целью подготовить данные к процессу обучения, выявить и удалить различные выбросы, ошибки в данных, что позволяет повысить эффективность обучаемой на этих данных модели.

2.1. Общая подготовка данных

Данный этап препроцессинга данных производил их общую подготовку и решал проблему размерности датасета. Данные проверялись на наличие бесконечных и нулевых значений и удалял последние.

Так же были исключены признаки, не влияющие на классификацию:
- IP-адреса,
- Порты,
- Временная метка,
- Протоколы.

Эти параметры могут быть легко изменены злоумышленниками вследствие чего не должны влиять на обучение модели.

2.2. Балансировка классов

Классы сетевого трафика были сильно не сбалансированы: количество нормального трафика превышало 1 миллион, тогда как некоторые виды атак имели менее 5 тысяч записей.

Классы атак Web Attack — XSS, Infiltration, «Web Attack — Sql Injection и Heartbleed были исключены из набора данных в связи с слишком малым количеством примеров, что означает, что модель обучить на этих данных будет невозможно, либо она будет часто ошибаться.

Для устранения дисбаланса данных использовались следующие методы:
- RandomUnderSampler — снижение числа экземпляров больших классов до 90 тысяч примеров;
- BorderLineSMOTE — искусственное увеличение малочисленных классов до 10 тысяч примеров.

Балансировка проводилась только на тренировочной выборке, чтобы избежать утечки данных и переобучения модели.

2.3. Удаление коррелирующих признаков

Построена матрица корреляции, и признаки с коэффициентом корреляции выше 0.8 были удалены. Это позволило снизить размерность входных данных и повысить обобщающую способность модели.

3. Построение модели

3.1. Выбор метода машинного обучения

Для решения задачи классификации был выбран метод случайного леса (Random Forest), который имеет ряд преимуществ:

```
#проверяем датсет на inf и nan значения

has_inf = np.isinf(worker_df.select_dtypes(include='number').values).any()
if has_inf:
    print("В датасете есть бесконечные значения, удаляем их")
    worker_df.replace([np.inf, -np.inf], np.nan, inplace=True)
else:
    print("Датасет не имеет бесконечных значений")

has_nan = np.isnan(worker_df.select_dtypes(include='number').values).any()
if has_nan:
    print("Датасет имеет NAN значения, удаляем")
    worker_df.dropna(inplace=True)

# До преобразования было (2589427, 84)
display(worker_df.shape)
```

Python

```
В датасете есть бесконечные значения, удаляем их
Датасет имеет NAN значения, удаляем

(2298395, 84)
```

Рис. 1. Проверка на наличие бесконечных и пустых значений

```
#Определение количества строк с нормальным трафиком и содержащим атаку
normal_trafic_total = len(worker_df[worker_df["Label"] == "BENIGN"])
attack_trafic_total = len(worker_df[worker_df["Label"] != "BENIGN"])
display("Нормальный трафик: {}".format(normal_trafic_total))
display("Аномальный трафик: {}".format(attack_trafic_total))

#вывод всех возможных значений метки Label
display(worker_df['Label'].value_counts())
```

```
'Нормальный трафик: 1741839'

'Аномальный трафик: 556556'

Label
BENIGN                        1741839
DoS Hulk                       230124
PortScan                       158804
DDoS                           128025
DoS GoldenEye                   10293
FTP-Patator                      7935
SSH-Patator                      5897
DoS slowloris                    5796
DoS Slowhttptest                 5499
Bot                              1956
Web Attack – Brute Force         1507
Web Attack – XSS                  652
Infiltration                       36
Web Attack – Sql Injection         21
Heartbleed                         11
Name: count, dtype: int64
```

Рис. 2. Виды атак и их количество в наборе данных

```
#функция автоматической стратегии балансировки
def balancing_strategy(y_train, min_samples=1000, max_samples=100000):
    class_counts = dict(Counter(y_train))
    over_strategy = {}
    under_strategy = {}

    for cls, count in class_counts.items():
        if count < min_samples:
            over_strategy[cls] = min_samples
        elif count > max_samples:
            under_strategy[cls] = max_samples
        else:
            pass  # оставляем как есть

    return over_strategy, under_strategy


over_strategy, under_strategy = balancing_strategy(y_train, min_samples=10000, max_samples=90000)

print("Стратегия Oversampling:", over_strategy)
print("Стратегия Undersampling:", under_strategy)

# пайплайн, производящий преобразование над классами
pipeline = Pipeline([
    ('over', BorderlineSMOTE(sampling_strategy=over_strategy, n_jobs=-1)),
    ('under', RandomUnderSampler(sampling_strategy=under_strategy))
])

#применяем пайплайн
x_res, y_res = pipeline.fit_resample(x_train, y_train)

print('\nПосле балансировки:', sorted(Counter(y_res).items()))
```

```
До балансировки: [(0, 1219287), (1, 1369), (2, 89617), (3, 7205), (4, 161087), (5, 3849), (6, 4057), (7, 5555), (8, 111163), (9, 4128), (10, 1055)]
Стратегия Oversampling: {7: 10000, 5: 10000, 9: 10000, 3: 10000, 6: 10000, 1: 10000, 10: 10000}
Стратегия Undersampling: {0: 90000, 8: 90000, 4: 90000}
C:\Users\MeizekiN\anaconda3\Lib\site-packages\imblearn\over_sampling\_smote\filter.py:197: FutureWarning: The parameter `n_jobs` has been deprecated in 0.
  warnings.warn(

После балансировки: [(0, 90000), (1, 10000), (2, 89617), (3, 10000), (4, 90000), (5, 10000), (6, 10000), (7, 10000), (8, 90000), (9, 10000), (10, 10000)]
```

Рис. 3. Функция балансировки тренировочной выборки

```python
#создаем функцию и удаляем наиболее коррелирующие признаки
def remove_correlated_features(input_df, threshold=0.9):
    #вычисляем матрицу корреляции
    corr_matrix = input_df.corr().abs()

    #верхний треугольник матрицы корреляции
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

    #поиск столбцов с корреляцией = выше заданного порога
    to_drop = [column for column in upper.columns if any(upper[column]>= threshold)]

    return input_df.drop(columns=to_drop), to_drop

#удаление для тренировочной выборки

x_train_clean_corr_df, dropped_corr_features=remove_correlated_features(x_train, threshold=0.9)
print(f'Удалено признаков: {len(dropped_corr_features)}')
print(f'Удаленные признаки: {dropped_corr_features}')

# удаляем точно такие же признаки из тестового набора
#  для согласованности данных и снижения риска утечки
x_test_clean_corr_df = x_test.drop(columns=dropped_corr_features, errors='ignore')
```

Python

```
Удалено признаков: 32
Удаленные признаки: ['Total_Backward_Packets', 'Total_Length_of_Bwd_Packets', 'Fwd_Packet_Length_Std', 'Bwd_P
```

Рис. 4. Удаление коррелирующих признаков из выборок

```python
from collections import Counter
counts = Counter(y_train)
total = sum(counts.values())
class_weights = {cls: round((total / count), 3) for cls, count in counts.items()}

print(f"Веса для классов: {class_weights}")
```

```
Веса для классов: {0: 4.774, 1: 42.962, 2: 4.794, 3: 42.962, 4: 4.774, 5: 42.962, 6: 42.962, 7: 42.962,
```

Рис. 5. Расчет весов для каждого класса

— Работать с категориальными признаками и числовыми признаками,
— Хорошая устойчивость к переобучению,
— Позволяет оценить важность признаков.

3.2. Настройка весов классов

Для повышения чувствительности модели к редким видам атак была проведена настройка весов классов с учетом их относительной частоты.

3.3. Оптимизация гиперпараметров

Поиск оптимальных параметров производился в два этапа:

1. RandomizeSearchCV — Поиск приблизительного диапазона.
2. GridSearchCV — Финальная точная настройка.

Настройке подвергались следующие параметры:
— n_estimators,
— max_features,
— min_samples_leaf.
— max_depth.
— min_samples_split.
— bootstrap.

3.4. Отбор важных признаков

Для уменьшения размерности и повышения интерпретируемости модели использовался метод

SelectFromModel (SFM) на основе обученного дерева решений. Метод SelectFromModel позволил повысить обобщающую способность модели, что подтверждается успешным применением аналогичных подходов в реальных задачах детектирования атак [9]

Результаты

Финальная модель показала улучшение качества по сравнению с базовой версией, что согласуется с результатами исследований, в которых были продемонстрированы преимущества использования ансамблевых методов в задачах классификации сетевого трафика [4][5]:

Таблица 1.

метрика	Первая модель	Финальная модель
Accuracy	95 %	97 %
F1-score (в среднем)	0.82	0.91

Особенно улучшилось качество по редким классам, но для использования в реальной системе обнаружения вторжений, все же стоит увеличить количество примеров редких классов.

Заключение

В работе продемонстрирована эффективность применения ансамблевого метода Random Forest для обнаружения сетевых атак. Предложенный подход, включающий препроцессинг, балансировку классов и подбор гиперпараметров, позволил достичь высокой точности модели.

Результаты могут быть использованы при разработке современных систем обнаружения вторжений, основанных на методах машинного обучения.

ЛИТЕРАТУРА

1. Шелухин О.И., Сакалема Д.Ж., Филинова А.С. Обнаружение вторжений в компьютерной сети // Горячая линия-телеком. — 2013. — № 4. — С. 45–52.
2. Скарфон К., Мелл П. Руководство по системам обнаружения и предотвращения вторжений (IDPS). Специальная публикация NIST 800–94, 2007.
3. Сомместад Т. и др. Машинное обучение для обнаружения вторжений в компьютерные сети: Обзор // IEEE Communications Surveys & Tutorials. — 2020. — Том 22, № 2. — С. 1296–1322.
4. Красильников Н.Н. Анализ сетевого трафика с использованием методов машинного обучения // Информационные технологии и защита информации. — 2021. — № 2. — С. 112–120.
5. Васильев А.А. Обнаружение атак в сетях на основе случайного леса // Вестник новых информационных технологий. — 2022. — № 3. — С. 78–85.
6. Ганем Р. и др. Обнаружение сетевых вторжений с использованием методов машинного обучения: всесторонний обзор // Компьютеры и безопасность. — 2021. — Том 107. — DOI: 10.1016/j.cose.2021.102311
7. Романов М.А., Коваленко А.С. Методы борьбы с дисбалансом классов в задачах анализа киберугроз // Защита информации. Инновационные технологии. — 2020. — № 3. — С. 45–50.
8. Педрегоса Ф. и др. Scikit-learn: Машинное обучение на Python // Journal of Machine Learning Research. — 2011. — Том 12. — С. 2825–2830.
9. Ван дер Уолт, К.М., Элофф, Дж. Х.П., Лабушань, В.А. Методы машинного обучения для обнаружения сетевых аномалий // Материалы конференции по информационной безопасности Южной Африки 2011 года. — 2011. — С. 1–7.
10. Ласков П., Шрндич Н. Практическая оценка детекторов вредоносных программ на основе машинного обучения // Препринт arXiv arXiv: 1801.08318, 2018.

Журнал «Современная наука: актуальные проблемы теории и практики»