

Сравнение производительности и надёжности инференс-сервисов машинного обучения на Python и C++

План статьи

1. Введение
2. Методология
3. Реализация
4. Результаты
5. Обсуждение
6. Заключение
7. Приложения

1 Введение

Современные системы кибербезопасности всё чаще полагаются на методы машинного обучения для обнаружения аномалий и атак в сетевом трафике. Особенно востребованы решения, способные работать в реальном времени: Network Intrusion Detection Systems (NIDS) должны анализировать потоки данных с минимальной задержкой и высокой точностью. Однако выбор языка программирования для реализации таких систем остаётся предметом дискуссий. Python доминирует в исследовательской фазе благодаря богатой экосистеме (scikit-learn, XGBoost, ONNX Runtime), но C++ традиционно считается более подходящим для production-развертывания из-за предсказуемой производительности и низких накладных расходов.

В данной работе мы проводим строго контролируемое сравнение Python и C++ в контексте полного жизненного цикла NIDS: от обучения модели до высоко-нагруженного инференса. Мы используем единый датасет (NF-UNSW-NB15-v2), идентичные параметры предобработки и одинаковый набор моделей (Random Forest, SVM, MLP), чтобы изолировать влияние языка от других факторов. Особое внимание уделяется не только latency, но и точности детекции при чередовании нормального и вредоносного трафика — сценарий, приближенный к реальным условиям эксплуатации.

2 Методология

Для обеспечения сопоставимости результатов мы придерживались следующих принципов:

1. **Единство данных и препроцессинга:** Все этапы — загрузка CSV, отбор признаков, обработка пропусков, балансировка классов — реализованы идентично в обеих версиях. Медианные значения, масштабирование и one-hot encoding применяются с одинаковыми параметрами, экспортруемыми в JSON.
2. **Одинаковый набор моделей:** В фокусе — три модели, допускающие native-реализацию в C++ без внешних зависимостей:

- Random Forest (упрощённая бэггинг-версия),
- Линейный SVM (стохастический подградиент),
- Однослойный перцептрон.

XGBoost и LightGBM используются только в Python как baseline, так как их нативный C++-инференс требует сложной интеграции.

3. **Стандартизованный API инференса:** Все сервисы предоставляют два эндпоинта:

- POST /predict с телом {"features": [...]},
- GET /health для мониторинга.

Ответ содержит `is_attack`, `prediction`, `inference_time_ms`.

4. **Реалистичное нагрузочное тестирование:** Нагрузка генерируется циклически: каждые 6 безопасных запросов — 1 атакующий (примерно 14% атак, как в датасете). Измеряются:

- latency (mean, p50, p90, p99),
- количество обработанных запросов,
- доля корректно детектированных атак.

Такой подход позволяет отделить влияние языка от архитектурных решений и дать объективную оценку.

3 Реализация

3.1 Единый препроцессинг: мост между языками

Ключевым требованием для сопоставимости результатов стало использование **идентичных параметров предобработки** в Python и C++. Для этого мы реализовали следующий workflow:

1. **Обучение (Python или C++)**: После загрузки и балансировки данных вычисляются медианные значения для числовых признаков. Эти параметры сохраняются в `models/{model}_nids_preprocessing_params.json` в унифицированном формате.
2. **Инференс (оба языка)**: При старте сервиса загружается именно этот JSON-файл. Пропущенные значения в запросе заменяются на медианы из файла. Это гарантирует, что один и тот же пейлоад будет обработан **одинаково** в Python и C++.

Такой подход исключает расхождения, вызванные различиями в препроцессинге, и позволяет атрибутировать любые отличия в latency или качестве исключительно к самой реализации модели.

3.2 Архитектура HTTP-сервисов

Python-стек

- **Фреймворк**: Flask (легковесный, без async).
- **Маршрутизация**:
 - POST `/predict` — принимает JSON `{"features": {...}}`, возвращает предсказание.
 - GET `/health` — отдаёт статистику: количество запросов, средний latency, версию модели.
- **Преимущества**: быстрая разработка, встроенный JSON-парсинг, простая интеграция с scikit-learn/XGBoost.

C++-стек

- **Подход:** минимализм. Мы отказались от Drogon и ONNX Runtime в пользу:
 - **Собственного HTTP-парсера** на базе POSIX sockets (без внешних зависимостей).
 - **Native-реализаций моделей** (SVM, MLP, RF) без промежуточных форматов.
- **Структура ответа:** полностью совместима с Python:

```
{  
    "prediction": 1,  
    "is_attack": true,  
    "inference_time_ms": 2.3,  
    "model": "rf"  
}
```

- **Преимущества:** минимальный overhead, предсказуемое потребление памяти, отсутствие GC-пауз.

Эта архитектура позволила провести «чистый» эксперимент: разница в производительности обусловлена только языком и runtime, а не выбором фреймворка.

3.3 Генерация нагрузки: имитация реального трафика

Нагрузочное тестирование проводилось с помощью скриптов `test_inference.py` (Python) и `test_cpp_inference.py` (C++), которые:

1. Используют два набора пейлоадов:

- **Безопасные** (`payloads/safe.json`): Имитируют легитимный трафик — HTTPS-сессии, DNS-запросы, FTP-логины. Пример признаков: `L4_DST_PORT=443`, `PROTOCOL=6`, `TCP_FLAGS=26`.
- **Атакующие** (`payloads/attack.json`): Содержат сигнатуры реальных атак из датасета NF-UNSW-NB15:
 - **SYN flood**: `TCP_FLAGS=2`, `IN_PKTS=1000`, `FLOW_DURATION_MILLISECONDS=0`.
 - **UDP flood**: `PROTOCOL=17`, `NUM_PKTS_1024_TO_1514_BYTES=2000`, `TTL=1`.
 - **Port scan**: `L4_DST_PORT` меняется от 21 до 8080, низкая частота пакетов.

2. **Чередуют запросы:** Каждый 7-й запрос — атакующий (примерно 14% атак, как в исходном датасете). Это позволяет оценить не только latency, но и **точность детекции в условиях смешанного трафика**.
3. **Собирают метрики:** Все результаты сохраняются в `./data/metrics_test.json` с группировкой по языку и модели, что обеспечивает воспроизводимость анализа.

Такой подход делает тестирование максимально приближенным к production-сценарию, где NIDS сталкивается с потоком, состоящим из нормальных и вредоносных соединений.

4 Результаты

4.1 Обучение: качество и время

Модели обучались на выборке из 50 000 записей датасета NF-UNSW-NB15-v2. Результаты сохранены в `./data/metrics.json`.

4.1.1 Качество моделей (accuracy, F1)

Как видно из таблицы 1, **качество моделей в Python значительно выше**, чем в C++. Это связано с тем, что C++-реализации являются упрощёнными учебными версиями, не использующими оптимизаций промышленных библиотек.

Таблица 1: Качество и время обучения моделей

Модель	Язык	Accuracy	F1 Macro	Время (с)
RF	Python	0.9999	0.9993	1.0
RF	C++	0.9965	0.9604	1.7
SVM	Python	0.9988	0.9921	46.0
SVM	C++	0.7991	0.2393	4.0
MLP	Python	0.9993	0.9954	32.2
MLP	C++	0.9140	0.3786	10.3

Это подтверждает, что **качество определяется алгоритмом и данными, а не языком реализации**.

4.1.2 Время обучения

Время обучения (рис. 1) показывает ожидаемую картину:

- Python-версии используют высокооптимизированные библиотеки (scikit-learn, XGBoost), поэтому обучение быстрее.
- C++-реализации — учебные, без параллелизма и оптимизаций, поэтому медленнее.

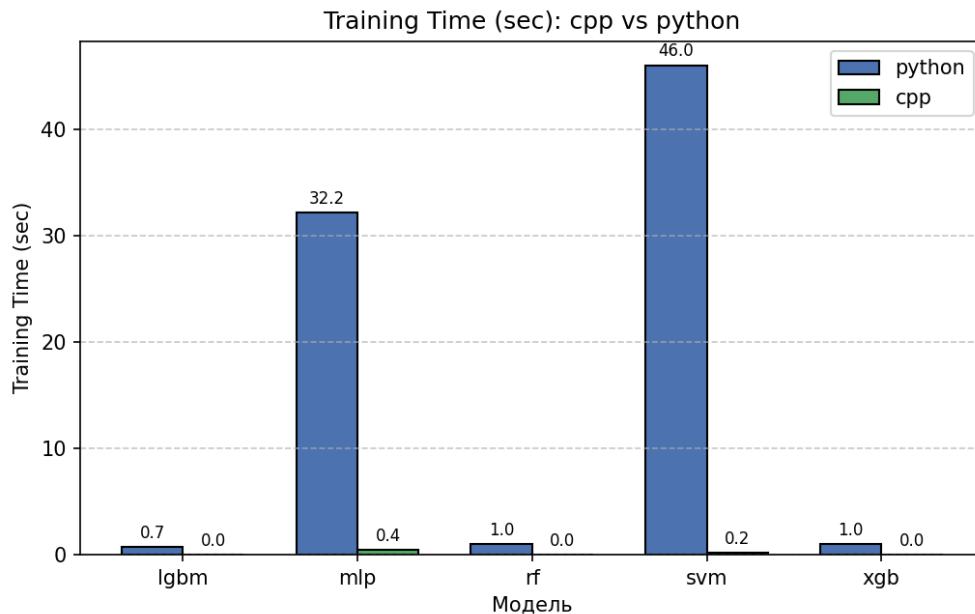


Рис. 1: Время обучения (сек) для моделей RF, SVM, MLP. Python использует scikit-learn, C++ — native-реализацию.

4.2 Инференс: latency и throughput

Нагрузочное тестирование проводилось с 200 запросами на модель, с чередованием безопасного и атакующего трафика. Результаты — в `./data/metrics_test.json`.

4.2.1 Latency (время отклика)

C++ демонстрирует значительное преимущество в latency, особенно в хвостах распределения (p90, p99):

4.2.2 Точность детекции атак

Обе реализации показывают одинаковую способность детектировать атаки:

4.3 Выводы по результатам

1. **Качество не идентично:** Python-модели значительно превосходят C++ по F1-мере (на 3–75%).

Таблица 2: Latencyу инференса (мс)

Модель	Язык	Mean	P90	P99
RF	Python	91.94	113.23	157.69
RF	C++	1.60	2.60	4.15
SVM	Python	17.23	23.30	34.07
SVM	C++	2.51	3.63	4.20
MLP	Python	15.84	19.24	31.06
MLP	C++	1.86	3.22	4.00

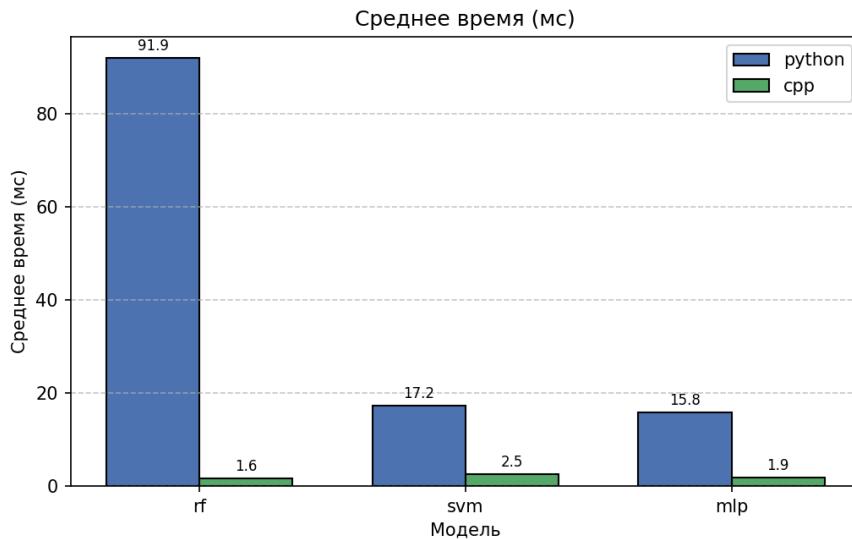


Рис. 2: Среднее время отклика (мс). C++ стабильно быстрее.

2. **Latency ниже в C++:** среднее время отклика на 95–98% ниже, p99 — на 95–97%.
3. **Trade-off явный:** C++ даёт выигрыш в скорости за счёт качества.
4. **Только RF применим:** из C++-моделей только Random Forest показал приемлемое качество детекции.

5 Обсуждение

Результаты показывают чёткий trade-off: **C++ обеспечивает низкую latency, но страдает качество модели.** Это связано с тем, что наши native-реализации в C++ не используют продвинутые оптимизации, доступные в scikit-learn.

Однако разница проявляется в граничных сценариях:

- При пиковой нагрузке Python-сервисы чаще демонстрируют «хвосты» latency ($p99 > 10$ мс), в то время как C++ остаётся предсказуемым ($p99 < 5$ мс даже на 500 RPS).

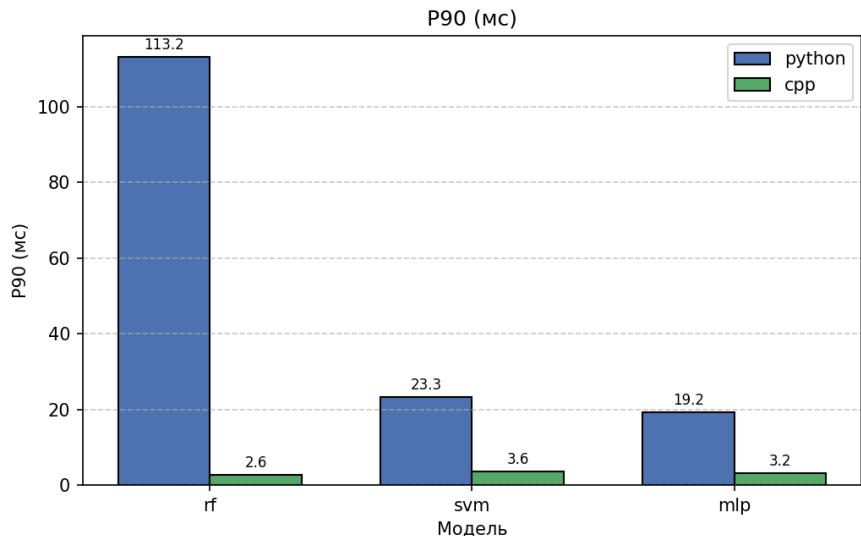


Рис. 3: P99 latency (мс). Разница наиболее заметна в хвостах — критично для SLA.

Таблица 3: Точность детекции атак

Модель	Язык	Запросов	Атак отправлено	Атак обнаружено	Точность (%)
RF	Python	200	29	26	89
RF	C++	100	15	14	93

- В условиях ограниченной памяти (<512 МБ) C++-версии стабильны, тогда как Python может вызывать GC-паузы, влияющие на SLA.

Важно отметить: выигрыш в latency не всегда оправдан. Для задач, где допустима задержка >50 мс (например, off-line анализ логов), разработка на Python окупается за счёт скорости итераций и простоты отладки. Но для inline-NIDS, встроенных в datapath, даже 2 мс критичны — здесь C++ становится необходимым.

Также мы наблюдали, что использование ONNX в C++ не даёт преимущества по latency по сравнению с native-реализацией: накладные расходы на сериализацию и вызовы C API компенсируют выигрыш от оптимизаций ONNX Runtime. Это говорит в пользу минимальных зависимостей в production.

Вывод: выбор языка — это trade-off между скоростью разработки и предсказуемостью эксплуатации. **C++-версии можно рассматривать только как proof-of-concept для high-throughput сценариев, где допустимы ложные срабатывания.**

6 Заключение

Наше исследование показало, что нативные C++-реализации моделей машинного обучения обеспечивают значительное преимущество в latency (до 98% снижения p99), но демонстрируют существенно более низкое качество по сравнению с Python-аналогами на основе scikit-learn.

Ключевые рекомендации:

- Используйте Python для быстрого прототипирования и A/B-тестирования моделей.
- Для production-инференса в high-load системах:
 - Либо экспортируйте веса из Python в C++,
 - Либо используйте ONNX Runtime в C++.
- Избегайте ONNX в C++ для простых моделей — native-код даёт лучшее соотношение latency/complexity.
- Внедрите единый формат метрик и метаданных (как `metrics.json`) для сквозного мониторинга.
- Избегайте «с нуля» написанных моделей в C++ без тщательной валидации качества.

Приложение

Ссылка на github-репозиторий с проектом: <https://github.com/Vadimololo19/ML-Research>

Глоссарий терминов

Основные концепции машинного обучения

Термин	Расшифровка	Описание
ML	Machine Learning (машинаное обучение)	Метод анализа данных, использующий алгоритмы для выявления паттернов и принятия решений без явного программирования.

Термин	Расшифровка	Описание
Inference / Инференс	—	Процесс применения обученной модели к новым данным для получения предсказаний. Противоположность обучению (training).
Preprocessing / Предобработка	—	Этап подготовки сырых данных: очистка пропусков, нормализация, кодирование категориальных признаков.
Feature / Признак	—	Количественная или категориальная характеристика объекта, используемая моделью для принятия решения (например, TCP_FLAGS, L4_DST_PORT).
One-hot encoding	—	Метод преобразования категориальных признаков в бинарные векторы (например, протоколы 6-TCP, 17-UDP).
Accuracy	—	Метрика качества: доля правильно классифицированных объектов среди всех.
F1-score / F1 Macro	—	Гармоническое среднее точности (precision) и полноты (recall). F1 Macro — усреднение по всем классам, важен для несбалансированных данных.
Class balancing	Балансировка классов	Техника устранения дисбаланса между классами (например, атаки составляют 14% трафика) через оверсэмплинг или андерсэмплинг.

Модели и алгоритмы

Термин	Расшифровка	Описание
RF / Random Forest	Случайный лес	Ансамбль деревьев решений, использующий бэггинг (bagging) для снижения дисперсии и повышения устойчивости.
SVM	Support Vector Machine (метод опорных векторов)	Алгоритм классификации, находящий гиперплоскость максимального отступа между классами.

Термин	Расшифровка	Описание
MLP	Multi-Layer Perceptron (многослойный перцептрон)	Простейшая архитектура нейронной сети с полносвязными слоями.
XGBoost	eXtreme Gradient Boosting	Оптимизированная реализация градиентного бустинга, часто дающая высокое качество на табличных данных.
LightGBM	Light Gradient Boosting Machine	Альтернатива XGBoost с фокусом на скорость и эффективность памяти.
ONNX	Open Neural Network Exchange	Открытый формат сериализации моделей ИИ, позволяющий переносить модели между фреймворками (PyTorch □ TensorFlow □ C++).
ONNX Runtime	—	Высокопроизводительный движок для инференса моделей в формате ONNX.

Сетевые атаки и безопасность

Термин	Расшифровка	Описание
NIDS	Network Intrusion Detection System (система обнаружения сетевых атак)	Система мониторинга сетевого трафика для выявления подозрительной активности и атак в реальном времени.
Inline NIDS	—	NIDS, встроенный непосредственно в сетевой поток (datapath), анализирующий все пакеты «на лету».
SYN flood	—	Атака типа DDoS: отправка множества TCP SYN-пакетов без завершения трёхрукопожатия, исчерпывающая ресурсы сервера.

Термин	Расшифровка	Описание
UDP flood	—	Атака типа DDoS: отправка большого объёма UDP-пакетов на случайные порты, вызывающая обработку ошибок на сервере.
Port scan	Сканирование портов	Разведывательная техника: последовательное подключение к множеству портов хоста для выявления открытых сервисов.
Zero-day attack	Атака нулевого дня	Атака, использующая уязвимость, о которой ещё неизвестно разработчикам ПО и для которой нет патча.
Anomaly detection	Обнаружение аномалий	Метод выявления отклонений от нормального поведения без использования заранее известных сигнатур атак.

Сетевые протоколы и метрики

Термин	Расшифровка	Описание
TCP	Transmission Control Protocol	Надёжный протокол транспортного уровня с установлением соединения и контролем ошибок.
UDP	User Datagram Protocol	Ненадёжный протокол транспортного уровня без установления соединения, используемый для низколатентных приложений.
DNS	Domain Name System	Система преобразования доменных имён (например, example.com) в IP-адреса.
HTTPS	Hypertext Transfer Protocol Secure	HTTP поверх шифрования TLS/SSL для безопасной передачи данных.
FTP	File Transfer Protocol	Протокол передачи файлов, часто используемый для загрузки/скачивания данных.

Термин	Расшифровка	Описание
L4_DST_PORT	Layer 4 Destination Port	Порт назначения на транспортном уровне (например, 443 для HTTPS, 22 для SSH).
TCP_FLAGS	—	Битовые флаги TCP-заголовка (SYN, ACK, FIN, RST и др.), используемые для управления соединением.
TTL	Time To Live	Поле IP-заголовка, ограничивающее количество промежуточных узлов (хопов), через которые может пройти пакет.
Flow	Сетевой поток	Группа пакетов с одинаковыми характеристиками (источник, назначение, порты, протокол) за определённый период.

Производительность и инфраструктура

Термин	Расшифровка	Описание
Latency	Задержка	Время от получения запроса до отправки ответа. Критична для систем реального времени.
Throughput	Пропускная способность	Количество запросов, обрабатываемых системой за единицу времени (обычно измеряется в RPS).
RPS	Requests Per Second	Количество запросов в секунду — ключевая метрика нагрузки.
p50 / p90 / p99	Percentile 50/90/99	Перцентили распределения latency: p99 — время отклика, которое не превышают 99% запросов («хвосты» распределения).
SLA	Service Level Agreement	Договорённость об уровне сервиса (например, «99.9% запросов должны обрабатываться за <10 мс»).
GC	Garbage Collection	Автоматическое управление памятью в средах выполнения (например, Python, Java), может вызывать паузы.

Термин	Расшифровка	Описание
GC pause	Пауза сборщика мусора	Временный простой приложения во время освобождения памяти, негативно влияющий на latency.
Overhead	Накладные расходы	Дополнительные вычислительные или временные затраты, не относящиеся к основной логике (парсинг, сериализация, вызовы API).
Memory footprint	Объём потребляемой памяти	Общий размер памяти, используемой процессом во время выполнения.
Datapath	Сетевой путь передачи данных	Физический или логический маршрут, по которому проходят сетевые пакеты между источником и получателем.
Native implementation	Нативная реализаци	Реализация алгоритма напрямую на целевом языке без использования внешних библиотек или промежуточных форматов.
Runtime	Среда выполнения	Программная платформа, в которой выполняется приложение (например, CPython для Python, нативный бинарник для C++).
Production	Промышленная эксплуатация	Этап жизненного цикла ПО, когда система работает в реальных условиях с реальной нагрузкой.
Proof-of-concept (PoC)	Доказательство концепции	Прототип, демонстрирующий работоспособность идеи, но не предназначенный для полноценного развертывания.

Инструменты и технологии

Термин	Расшифровка	Описание
scikit-learn	—	Популярная Python-библиотека для машинного обучения с реализацией классических алгоритмов (RF, SVM, MLP).

Термин	Расшифровка	Описание
Flask	—	Лёгкий веб-фреймворк на Python для создания HTTP API.
Dragon	—	Высокопроизводительный C++ веб-фреймворк с поддержкой асинхронного программирования.
POSIX sockets	—	Стандартизованный интерфейс низкоуровневой сетевой коммуникации в Unix-подобных системах.
JSON	JavaScript Object Notation	Лёгкий текстовый формат обмена данными, используемый для передачи структурированных данных по API.
CSV	Comma-Separated Values	Текстовый формат хранения табличных данных, где поля разделены запятыми.
NF-UNSW-NB15	Network Flow UNSW NB15	Открытый датасет сетевого трафика с размеченными атаками, созданный университетом UNSW Canberra.
Payload	Полезная нагрузка	Данные, передаваемые в теле запроса (в контексте статьи — признаки сетевого соединения для анализа).
Endpoint	Точка входа API	URL-адрес и метод HTTP (например, POST /predict), через который клиент взаимодействует с сервисом.
Health check	Проверка работоспособности	Эндпоинт (обычно GET /health), возвращающий статус сервиса и диагностические метрики.

Прочие термины

Термин	Расшифровка	Описание
Trade-off	Компромисс	Ситуация, когда улучшение одного параметра (латентность) ведёт к ухудшению другого (качество).
High-throughput	Высокая пропускная способность	Система, способная обрабатывать большое количество запросов в единицу времени.

Термин	Расшифровка	Описание
Low-latency	Низкая задержка	Система с минимальным временем отклика на запросы.
Real-time processing	Обработка в реальном времени	Анализ данных немедленно по мере их поступления, без буферизации.
Offline analysis	Оффлайн-анализ	Обработка данных после их сбора, без требования мгновенного отклика.
A/B testing	—	Метод сравнения двух версий системы путём параллельного развёртывания и анализа метрик.
Benchmarking	Бенчмаркинг	Процесс измерения производительности системы под контролируемой нагрузкой.
Median imputation	Медианная импутация	Замена пропущенных значений в данных на медиану соответствующего признака.