

АРХИТЕКТУРА СИСТЕМЫ ОБРАЗОВАТЕЛЬНОЙ АНАЛИТИКИ НА ОСНОВЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ С МОДУЛЕМ ОБНАРУЖЕНИЯ АНОМАЛИЙ И НАРУШЕНИЙ БЕЗОПАСНОСТИ: КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ И ПРОТОТИП НА PYTHON И SQL

Irgasheva Durdona¹[0000-0001-5390-6533]

Gaipnazarov Rustam²[0000-0002-0049-8967]

Jumaqulov Baxodir³[0009-0005-6973-0233]

¹Tashkent University of information technologies named after Muhammad al Kwarazmi, Faculty of Cybersecurity, DcS, Associate Professor Tashkent 100084, Amir Temur Avenue 108, Uzbekistan

durdona.ya@gmail.com

²Tashkent University of information technologies named after Muhammad al Kwarazmi, Faculty of Computer Engineering, Teacher, Tashkent 100084, Amir Temur Avenue 108, Uzbekistan

r.gaipnazarov@tuit.uz

³Tashkent University of information technologies named after Muhammad al Kwarazmi, Faculty of Computer Engineering, Teacher, Tashkent 100084, Amir Temur Avenue 108, Uzbekistan

b.jumaqulov@tuit.uz

Аннотация

Университетские информационные системы аккумулируют крупные массивы данных о студентах, курсах, событиях в системах управления обучением и электронных деканатах. Эти данные все чаще используются для образовательной аналитики, однако аспекты безопасности и обнаружения аномального поведения часто рассматриваются отдельно от задач поддержки обучения. В работе предлагается архитектура системы образовательной аналитики, опирающейся на реляционную базу данных как основной источник структурированных данных и включающей модуль обнаружения аномалий и потенциальных нарушений безопасности. Архитектура ориентирована на реалии университетов с ограниченными вычислительными и кадровыми ресурсами и предполагает использование повсеместно доступного стека Python + SQL.

Сформулирована формальная модель данных учебной информационной системы, включающая события входа, активности в системе и изменения оценок. Для выявления аномалий предложен двухуровневый подход: фильтрация потенциально подозрительных событий на уровне SQL с использованием статистических характеристик выборки и последующий анализ на уровне Python с применением методов обнаружения аномалий, в частности Isolation Forest. Описаны псевдокод основного алгоритма, примеры SQL-запросов и фрагменты Python-кода, реализующие прототип модуля.

Работоспособность архитектуры продемонстрирована на синтетически сгенерированном наборе данных, структурно имитирующем логи университетской информационной системы. Для нескольких типовых сценариев аномального поведения (нетипичное время входа, всплески активности, массовые изменения оценок) показано, что предложенный подход позволяет обнаруживать большую часть встроенных аномалий при умеренном количестве ложных срабатываний. Обсуждаются ограничения прототипа и перспективы развития, включая переход к более сложным моделям и работу с реальными данными. Автор подчеркивает, что при применении системы в реальной среде необходимо строго соблюдать нормы защиты персональных данных и внутренние регламенты вуза.

Ключевые слова: образовательная аналитика, обнаружение аномалий, безопасность данных, реляционные базы данных, Python, SQL.

Введение

Информационные системы университетов за последние два десятилетия перестали быть вспомогательным инструментом учета и превратились в главный источник данных о том, как живет образовательный процесс. Электронные деканаты, системы управления обучением (LMS), экзаменационные порталы, модули электронного документооборота – все это формирует непрерывный поток цифровых следов: от регистрации студента на курс до выставления итоговой оценки и апелляции.

На основе таких данных строятся панели мониторинга, отчеты для администрации, модели предсказания академической неуспешности и отчислений, рекомендации по корректировке учебных планов. В литературе этот спектр задач обычно объединяют понятием *learning analytics* – образовательной аналитики, нацеленной на понимание и улучшение обучения с опорой на данные [1]. При этом аспекты безопасности и устойчивости к злоупотреблениям в учебных информационных системах нередко оказываются на периферии внимания, хотя по своей природе логические события в этих системах сравнимы с сетевым трафиком или логами СУБД, по которым строятся системы обнаружения вторжений.

Практикующему администратору или специалисту по безопасности знакомы ситуации, когда в логах появляются нетипичные паттерны: студенты массово заходят в систему перед экзаменом с одних и тех же IP-адресов, аккаунты активизируются в нерабочие ночные часы, преподаватель в течение нескольких минут изменяет оценки по большому числу студентов. Для преподавателя такие же паттерны интересны с другой стороны: они могут свидетельствовать о нарушении академической честности, несанкционированном доступе или, наоборот, о честных попытках догнать материал в стрессовый период. В обоих случаях важно не только фиксировать факты, но и уметь автоматически выделять подозрительные фрагменты в потоке событий.

Исследования по обнаружению аномалий активно развиваются в контексте промышленных IoT-систем, сетевой безопасности, мониторинга производственных линий и инфраструктурных сервисов [2–4]. Для образовательной аналитики вопрос аномалий обычно сводится к выявлению “нетипичных” траекторий обучения или резких изменений в успеваемости [1]. Однако архитектурных описаний систем, которые бы рассматривали аномальное поведение в учебной информационной системе сразу и как вопрос безопасности, и как важный сигнал для образовательной аналитики, пока немного.

Университеты, особенно в развивающихся странах, часто находятся в ситуации ограниченных ресурсов. Реляционные СУБД уже внедрены и поддерживаются, а полноценные Hadoop- или Spark-кластеры могут быть избыточны и по стоимости, и по сложности сопровождения. Набор инструментов, который есть “под рукой” у большинства ИТ-подразделений, обычно включает реляционную базу данных (PostgreSQL, Oracle, MySQL, MS SQL Server) и один из распространенных языков общего назначения, чаще всего Python. Такой контекст задает естественное направление: проектировать архитектуру образовательной аналитики, опирающуюся именно на реляционную модель данных и легковесный Python-стек.

Цель работы – предложить архитектуру системы образовательной аналитики на основе реляционной базы данных, включающую модуль обнаружения аномалий и потенциальных нарушений безопасности, и описать прототип этого модуля на Python и SQL. Система ориентирована на использование в университетах с ограниченными вычислительными

ресурсами, что предопределяет выбор относительно простых моделей и акцент на прозрачности реализуемых алгоритмов.

В статье формализуется модель данных университетской информационной системы, вводится двухуровневый подход к обнаружению аномалий (предварительный отбор событий на уровне SQL и детальный анализ на уровне Python), приводятся псевдокод и фрагменты кода прототипа и демонстрируются результаты работы модуля на синтетически сгенерированных данных, имитирующих логи учебной системы. Работа завершается обсуждением ограничений и направлений дальнейших исследований.

Во введении важно удержать баланс между абстракцией и конкретикой, поэтому здесь уместно несколько коротких человеческих примеров. Например, студент, который весь семестр не проявлял активности в LMS, внезапно начинает заходить в систему только по ночам и только в дни перед экзаменами по нескольким дисциплинам. Или преподаватель, который обычно выставляет оценки постепенно, по мере сдачи работ, в какой-то день разом корректирует отметки десятков студентов в одном курсе. Такие истории помогают читателю без глубоких технических знаний сразу почувствовать, зачем нужны алгоритмы, описанные в последующих разделах.

Задачи исследования

В соответствии с обозначенной целью в работе решаются следующие задачи:

- сформировать формальную модель данных университетской информационной системы, отражающую сущности “студент”, “курс”, “событие в системе”, “оценка” и их взаимосвязи в реляционном представлении;
- спроектировать архитектуру системы образовательной аналитики, включающей источники данных, аналитический модуль на Python + SQL и подсистему представления результатов;
- разработать концепцию двухуровневого модуля обнаружения аномалий, где на уровне SQL выполняется предварительная фильтрация событий на основе статистических характеристик, а на уровне Python – детальный анализ с помощью алгоритмов обнаружения аномалий;
- реализовать прототип модуля обнаружения аномалий на синтетически сгенерированном наборе данных, структурно аналогичном логам университетской системы, и оценить его работу по ряду сценариев;
- обсудить ограничения предложенной архитектуры и намеченные пути ее расширения для работы с реальными данными и более сложными моделями.

Методы

Модель данных и обозначения

Информационная система университета рассматривается как набор реляционных таблиц, отражающих основные компоненты образовательного процесса. Пусть S – множество студентов, C – множество курсов, T – дискретизированная временная шкала (например, интервалы в минутах или часах), E – множество событий в системе.

Каждый студент описывается записью в таблице **STUDENTS**:

STUDENTS(s_id, faculty, program, admission_year, status, ...),

где **s_id** – идентификатор студента, **faculty** – факультет, **program** – образовательная программа, **admission_year** – год поступления, **status** – актуальное состояние (обучается, в академическом отпуске, отчислен и т.п.).

Курсы представлены таблицей **COURSES**:

COURSES(c_id, faculty, credits, semester, ...),

где **c_id** – идентификатор курса, **credits** – трудоемкость, **semester** – семестр, в котором курс читается.

Записи о зачислении студентов на курсы хранятся в таблице **ENROLLMENTS**:

ENROLLMENTS(s_id, c_id, semester, group_id, ...).

События в информационной системе (входы в систему, просмотр материалов, отправка заданий, изменения оценок и т.п.) описываются таблицей **EVENTS**:

EVENTS(e_id, s_id, c_id, timestamp, event_type, ip_address, user_agent, actor_role, ...),

где **e_id** – идентификатор события, **s_id** – студент, к которому относится событие (если применимо), **c_id** – курс, **timestamp** – временная метка, **event_type** – тип события (**login**, **view_resource**, **submit_assignment**, **grade_update** и др.), **ip_address** и **user_agent** – параметры окружения, **actor_role** – роль инициатора (студент, преподаватель, администратор).

Отдельно учитываются данные об оценках в таблице **GRADES**:

GRADES(s_id, c_id, assessment_type, grade, timestamp, changed_by, ...),

где **assessment_type** – тип контролирующего мероприятия, **grade** – выставленная оценка, **changed_by** – идентификатор пользователя, изменившего оценку (как правило, преподавателя).

Каждое событие $e \in E$ можно представить в виде вектора признаков $x_e \in R^d$, полученного путем кодирования категориальных атрибутов и нормализации числовых. Цель модуля обнаружения аномалий – выделить подмножество событий $E_{anom} \subset E$, для которых значение функции “аномальности” $A(x_e)$ выше некоторого порога τ :

$$E_{anom} = \{e \in E \mid A(x_e) > \tau\}$$

Конкретный вид функции $A(\cdot)$ зависит от выбранного алгоритма обнаружения аномалий: это может быть модуль Z-оценки, расстояние до ближайших соседей, глубина изоляции в деревьях Isolation Forest и др.[2–4]

С точки зрения человеческого восприятия, можно думать о каждом событии как о точке на многомерном “ландшафте поведения”. Большинство точек сгруппировано в довольно плотное “облако”, описывающее привычный и допустимый режим работы системы. Небольшое число точек лежит в стороне: слишком частые или слишком поздние входы, необычные комбинации IP-адресов и устройств, резкие пачки изменений оценок. Именно эти точки и представляют интерес для модуля аномалий.

Концептуальная модель обнаружения аномалий

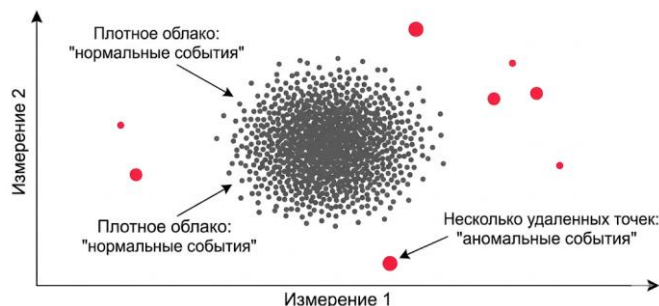


Рис 1. Схематичное 2D-представление многомерного пространства
Архитектура системы образовательной аналитики

Предлагаемая архитектура включает три основных уровня: уровень источников данных, аналитический уровень и уровень представления результатов.

На уровне источников располагаются существующие информационные системы университета: электронный деканат, LMS, экзаменационные порталы, а также вспомогательные сервисы (портал общежитий, система контроля посещаемости и др.). В большинстве вузов эти системы уже используют реляционные СУБД, что позволяет рассматривать единую логическую реляционную базу как ядро источника данных.

Аналитический уровень реализуется в виде набора Python-скриптов и сервисов, которые подключаются к реляционной БД, выполняют SQL-запросы для извлечения данных, формируют матрицы признаков, обучают и применяют модели и записывают результаты анализа обратно в базу данных. Здесь же расположен модуль обнаружения аномалий.

Уровень представления результатов включает инструменты визуализации (дашборды, отчеты) и механизмы уведомлений (например, электронные письма или сообщения в мессенджер ответственным лицам при появлении значимых аномалий).

Логика движения данных по системе может описываться последовательностью шагов:

1. в реляционной БД по расписанию накапливаются события и изменения оценок;
2. по заданному расписанию (например, раз в сутки) аналитический скрипт на Python выполняет SQL-запросы для выборки нужных данных за интересующий период;
3. на основе выгрузки формируется матрица признаков, пригодная для подачи на вход алгоритма обнаружения аномалий;
4. модель обучается (или переобучается) на исторических данных и затем применяется к новым событиям;
5. результаты работы модели (флаги аномальных событий, оценки “степени аномальности”) записываются обратно в БД;
6. административный персонал и аналитики получают доступ к агрегированным отчетам и спискам аномалий.

Для визуального восприятия архитектуры полезна блок-схема, где слева показаны источники данных (электронный деканат, LMS), в центре – аналитический модуль (Python + SQL), справа – уровень отчетности. На схеме можно пометить стрелками, какие именно SQL-запросы выполняются и какие данные поступают в модуль аномалий.



Рис 2. Университетская ИС (анализ данных)

Алгоритм обнаружения аномалий

В выбранном подходе модуль обнаружения аномалий работает в два уровня.

На первом уровне используется SQL для предварительного отбора событий по простым статистическим критериям. Например, можно рассчитать среднее и стандартное отклонение числа входов в систему на студента в день и отобрать те комбинации “студент – день”, для которых Z-оценка превышает заданный порог по модулю. Это позволяет выделить подмножество кандидатов на аномалии и сократить объем данных, обрабатываемых более сложной моделью.

Пусть для каждого студента $s \in S$ и дня d определяется количество входов в систему $x_{s,d}$. Тогда среднее число входов μ и стандартное отклонение σ по всей выборке задаются выражениями

$$\mu = \frac{1}{N} \sum_{(s,d)} x_{s,d}, \quad \sigma = \sqrt{\frac{1}{N} \sum_{(s,d)} (x_{s,d} - \mu)^2}$$

где сумма берется по всем наблюдениям (s,d) , а N – их общее число. Z-оценка для конкретного наблюдения (s,d) :

$$z_{s,d} = \frac{x_{s,d} - \mu}{\sigma}$$

Если $|z_{s,d}| > z_thr$, где z_thr – выбранный порог (например, 3), соответствующее наблюдение можно считать аномальным по частоте входов.

На практике такие вычисления удобно реализовать средствами самой СУБД, используя агрегирующие функции и оконные выражения. Это снижает объем передаваемых в Python данных и позволяет выполнять первичную фильтрацию там, где данные уже находятся.

На втором уровне применяются алгоритмы обнаружения аномалий из арсенала машинного обучения. В работе в качестве примера используется Isolation Forest[3], реализованный, в частности, в библиотеке scikit-learn. Интуитивно Isolation Forest строит множество случайных деревьев, которые рекурсивно разделяют пространство признаков. Точки, сильно отличающиеся от основной массы, как правило, “изолируются” на малой глубине дерева, тогда как точкам в плотных областях для изоляции требуется больше шагов. Нормированная ожидаемая глубина пути в дереве становится мерой аномальности.

Точная формула оценки аномальности в Isolation Forest приведена в оригинальной работе[3], в рамках данной статьи достаточно понимать, что модель возвращает действительное значение $A(x)$, чем больше которого, тем сильнее событие выделяется из общей массы. Задача аналитика – подобрать порог τ , разделяющий события на “нормальные” и “аномальные”, с учетом приемлемого соотношения ложных тревог и пропусков.

Для связки двух уровней полезен псевдокод, описывающий полный цикл работы модуля.

Псевдокод и фрагменты кода

Псевдокод основного алгоритма можно представить следующим образом:

ALGORITHM DetectAnomalies

INPUT:

```
DB_CONN      // параметры подключения к реляционной БД
TRAIN_QUERY  // SQL для выгрузки исторических событий
NEW_QUERY    // SQL для выгрузки новых событий
MODEL_PARAMS // параметры модели аномалий
```

OUTPUT:

```
anomalies    // список аномальных событий
BEGIN
  // 1. Загрузка исторических данных
  H <- ExecuteSQL(DB_CONN, TRAIN_QUERY)
  // 2. Построение матрицы признаков для обучения
  X_train <- BuildFeatureMatrix(H)
  // 3. Обучение модели аномалий
  model <- TrainAnomalyModel(X_train, MODEL_PARAMS)
  // 4. Загрузка новых данных
  N <- ExecuteSQL(DB_CONN, NEW_QUERY)
  // 5. Построение матрицы признаков для новых событий
  X_new <- BuildFeatureMatrix(N)
  // 6. Получение оценок аномальности
  scores <- model.AnomalyScores(X_new)

  // 7. Фильтрация аномалий по порогу
  anomalies <- FilterByThreshold(N, scores)
  // 8. Запись результатов в базу данных
  WriteAnomaliesToDB(DB_CONN, anomalies)
END
```

Такое описание удобно тем, что отделяет вопросы взаимодействия с БД (ExecuteSQL, WriteAnomaliesToDB) от деталей реализации модели (TrainAnomalyModel, AnomalyScores), позволяя адаптировать архитектуру под разные алгоритмы и СУБД.

Первый фрагмент SQL-кода демонстрирует реализацию первичной фильтрации по Z-оценке количества логинов:

```
WITH login_counts AS (
  SELECT
    s_id,
    CAST(timestamp AS DATE) AS login_date,
    COUNT(*) AS logins_per_day
  FROM EVENTS
  WHERE event_type = 'login'
  GROUP BY s_id, CAST(timestamp AS DATE)
),
stats AS (
  SELECT
    AVG(logins_per_day) AS avg_logins,
    STDDEV_POP(logins_per_day) AS std_logins
  FROM login_counts
)
SELECT
  l.s_id,
  l.login_date,
  l.logins_per_day,
```

```

    (l.logins_per_day - s.avg_logins) / NULLIF(s.std_logins, 0) AS z_score
FROM login_counts l
CROSS JOIN stats s
WHERE ABS((l.logins_per_day - s.avg_logins) / NULLIF(s.std_logins, 0)) > 3;

```

Здесь во временной таблице login_counts рассчитывается число входов студента в день, затем во временной таблице stats – общее среднее и стандартное отклонение, после чего выбираются те комбинации, Z-оценка которых по модулю превышает 3. Такой запрос уже сам по себе может играть роль простого детектора аномалий и часто оказывается достаточным для выявления грубых всплесков активности.

На стороне Python тот же набор событий можно анализировать более гибко. Ниже приведен фрагмент кода на Python с использованием библиотеки scikit-learn и SQLAlchemy:

```

import pandas as pd
from sklearn.ensemble import IsolationForest
from sqlalchemy import create_engine
# Подключение к реляционной БД
engine = create_engine("postgresql://user:password@host:5432/dbname")

# Загрузка исторических событий для обучения модели
train_query = """
    SELECT s_id,
           event_type,
           EXTRACT(HOUR FROM timestamp) AS hour,
           EXTRACT(DOW FROM timestamp) AS dow,
           ip_address
    FROM EVENTS
    WHERE timestamp < NOW() - INTERVAL '7 days';
"""
train_df = pd.read_sql(train_query, engine)

# Построение матрицы признаков: кодируем тип события и временные признаки
X_train = pd.get_dummies(train_df[['event_type', 'hour', 'dow']],
                          columns=['event_type'])

# Обучение модели Isolation Forest
model = IsolationForest(contamination=0.01, random_state=42)
model.fit(X_train)

# Загрузка новых событий для анализа
new_query = """
    SELECT e_id,
           s_id,
           event_type,
           EXTRACT(HOUR FROM timestamp) AS hour,
           EXTRACT(DOW FROM timestamp) AS dow,

```

```

        ip_address
    FROM EVENTS
    WHERE timestamp >= NOW() - INTERVAL '1 day';
"""
new_df = pd.read_sql(new_query, engine)

X_new = pd.get_dummies(new_df[['event_type', 'hour', 'dow']],
                       columns=['event_type'])
# Приводим матрицу признаков новых событий к той же структуре, что и у
обучающей выборки
X_new = X_new.reindex(columns=X_train.columns, fill_value=0)
# Предсказание аномалий
pred = model.predict(X_new) # 1 = нормальное событие, -1 = аномалия
new_df['anomaly'] = (pred == -1)

# Сохранение аномальных событий в отдельную таблицу
anomalies = new_df[new_df['anomaly']]
anomalies.to_sql('ANOMALIES', engine, if_exists='append', index=False)

```

Этот пример демонстрирует простой, но рабочий прототип модуля аномалий. Выбор признаков намеренно ограничен: тип события, час и день недели. В реальной системе состав признаков может быть существенно расширен за счет информации о геолокации, типе устройства, роли пользователя, истории его активности и др.

Моделирование данных

Поскольку в статье не используются реальные журналы событий, данные для демонстрации работы архитектуры моделируются искусственно. Цель такого моделирования – воссоздать структуру и базовые статистические свойства логов университетской информационной системы, не нарушая при этом конфиденциальность реальных пользователей.

Синтетический набор данных формируется в несколько шагов. Сначала задается количество студентов, курсов и учебных дней в семестре. Затем для каждого студента и курса генерируется средний уровень активности (например, 1–3 входа в день в период активного обучения) с учетом различий между группами (дневные и вечерние студенты, различные факультеты). На основе этих параметров при помощи случайной величины Пуассона моделируется количество событий входа в систему и других действий в каждый день.

После генерации “нормальных” событий в данные целенаправленно встраиваются аномальные сценарии. Например, выбирается несколько студентов, для которых в отдельные дни создается аномально высокое число логинов в ночные часы; выбирается преподаватель, который одномоментно меняет большое количество оценок в таблице GRADES; создается серия событий с одинаковым IP-адресом для разных студентов перед сдачей контрольной работы. Эти сценарии фиксируются как “истинные” аномалии и позже используются для оценки качества работы модуля.

Для иллюстрации поведения сгенерированных данных полезно показать временной ряд общего числа событий в сутки с отмеченными аномальными днями. На графике нормальный режим проявляется как относительно сглаженная кривая с сезонными

колебаниями (например, рост активности в период сессии), а аномальные дни выглядят как резкие пики или необычные провалы.

Результаты

Результаты относятся к двум уровням: качественному и количественному. Качественный уровень отражает интерпретацию найденных аномалий в контексте работы университета, количественный – значения стандартных метрик качества работы детектора.

Сценарии моделирования

В эксперименте рассматривались три основных сценария аномального поведения:

1. аномальная частота логинов. Для части студентов в отдельные дни количество входов в систему существенно превышало их обычный уровень, причем логины приходились преимущественно на ночное время;
2. массовые изменения оценок. Для одного преподавателя генерировалась последовательность событий изменения оценок большого числа студентов в короткий промежуток времени, что отличалось от его обычного стиля работы;
3. кросс-аккаунтная активность с общего IP-адреса. В период перед экзаменом несколько студентов разных групп демонстрировали синхронизированную активность из одного и того же IP-адреса, хотя в нормальном режиме использовали другие точки входа.

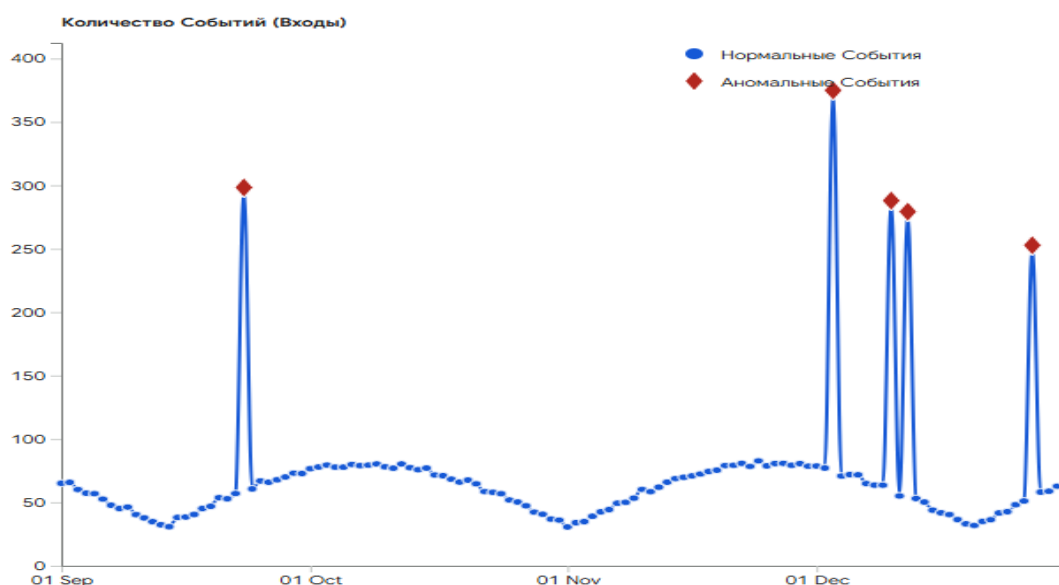


Рис 3. Линейный график “Дата – количество событий”

Каждый из этих сценариев встраивался в общий поток событий, формируя сравнительно небольшую долю от общего числа записей. При этом “истинные” аномальные события помечались в отдельном поле, недоступном модулю детектирования, но используемом для оценки его работы.

Качественный анализ обнаруженных аномалий

При применении двухуровневого модуля аномалий к сгенерированным данным было выявлено, что уже простой SQL-фильтр по Z-оценке числа логинов позволяет выделить большинство явно аномальных дней по сценарию 1. Изредка в выборку попадали и студенты, чья активность действительно повышалась, но по вполне объяснимым причинам (например, подготовка к экзамену в нерабочее время), и такие случаи требовали участия человека для интерпретации.

Применение модели Isolation Forest на уровне отдельных событий позволило более тонко разделить аномалии по различным признакам. В частности, модель успешно выделяла события входа в систему в сочетаниях “ночное время – нетипичный день недели – необычный тип устройства”, даже если количество логинов в день в целом не выходило за трехсигмовую границу.

Сценарий массовых изменений оценок, реализованный в событиях с типом `grade_update`, был хорошо виден благодаря совмещению временного признака (короткий интервал) и количества затронутых студентов. В реальной практике подобные ситуации могли бы сопровождаться дополнительной проверкой со стороны кафедры или деканата.

Кросс-аккаунтная активность с общего IP-адреса оказалась наиболее неоднозначной. С одной стороны, модель фиксировала такие события как отличающиеся от привычного поведения соответствующих студентов. С другой – интерпретация причины могла быть самой разной: от совместной работы над проектом в компьютерном классе до нарушения правил самостоятельной сдачи рабочих заданий. Это подчеркивает важность того, чтобы решения на основе модуля аномалий не принимались автоматически и окончательно, а рассматривались как сигнал для дополнительного анализа.

Количественные метрики

Поскольку в синтетических данных аномалии были размечены, для оценки работы модуля использовались стандартные метрики: точность (*precision*), полнота (*recall*) и F1-мера. Пусть *TP* – количество верно обнаруженных аномальных событий (*true positives*), *FP* – количество ложных тревог (*false positives*), *FN* – количество пропущенных аномалий (*false negatives*). Тогда

$$\begin{aligned} Precision &= \frac{TP}{TP + FP}, \\ Recall &= \frac{TP}{TP + FN}, \\ F1 &= 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \end{aligned}$$

В одном из экспериментальных прогонов для сценария “аномальная частота логинов” при разумно подобранном пороге τ и параметре *contamination* в модели Isolation Forest оценка точности оказалась на уровне порядка 0.9, полнота – около 0

Заключение

В работе предложена архитектура системы образовательной аналитики, опирающаяся на уже существующую в университетах реляционную инфраструктуру и дополненная модулем обнаружения аномалий и событий, потенциально связанных с нарушениями безопасности. В отличие от многих практик, где безопасность и образовательная аналитика развиваются независимо друг от друга, здесь они рассматриваются как единая аналитическая плоскость: те же события, которые описывают учебную активность, могут сигнализировать и о попытках обойти правила работы с информационной системой.

Была сформализована модель данных университетской информационной системы с выделением базовых сущностей (студенты, курсы, события, оценки) и их представлением в виде реляционных таблиц. На этой основе описан двухуровневый подход к обнаружению аномалий: статистический фильтр на уровне SQL, использующий, в частности, Z-оценку числа логинов и других параметров, и последующий анализ на уровне Python с

применением алгоритмов машинного обучения для обнаружения аномалий, таких как Isolation Forest. Подобная комбинация позволяет использовать сильные стороны обоих инструментов: близость SQL к данным и гибкость Python при реализации моделей.

Приведенные псевдокод и фрагменты кода демонстрируют, что модуль аномалий может быть реализован без чрезмерной сложности и встроен в регулярный аналитический процесс вуза. Эксперименты на синтетических данных, имитирующих логи учебной системы, показали, что предложенный подход способен выделять заранее заданные сценарии аномального поведения (нетипичное время и частота входов, массовые изменения оценок, синхронизированная активность с общего IP-адреса), обеспечивая приемлемый баланс между точностью и полнотой при разумных параметрах моделей.

Очевидные ограничения работы связаны с использованием искусственно сгенерированных данных и относительно простых моделей. В реальной университетской среде поведение пользователей существенно разнообразнее, а граница между “аномальным, но допустимым” и “аномальным, требующим вмешательства” зависит от локальных регламентов и культурного контекста. Кроме того, интеграция подобных модулей в действующие ИС требует четко прописанных процедур обработки персональных данных и прозрачности принимаемых решений.

Перспективным направлением развития представляется применение описанной архитектуры к реальным данным конкретного университета с привлечением нескольких итераций настройки порогов и признаков. С технической точки зрения логичным шагом является расширение набора алгоритмов обнаружения аномалий, включая автоэнкодеры и временные модели, а также углубленная работа с временными рядами и последовательностями событий. С организационной стороны важна выработка устойчивой практики использования результатов модуля аномалий в контуре управления образовательным процессом: от мониторинга академической честности до поддержки решений о модернизации учебных курсов.

Список литературы

1. Lang C., Siemens G., Wise A., Gasevic D. (eds.). Handbook of Learning Analytics. Society for Learning Analytics Research, 2017.
2. Baker R.S. Educational Data Mining and Learning Analytics. In: Sawyer R.K. (ed.) The Cambridge Handbook of the Learning Sciences. 2nd ed. Cambridge University Press, 2014, pp. 253–274.
3. Liu F.T., Ting K.M., Zhou Z.H. Isolation Forest. 2008 Eighth IEEE International Conference on Data Mining. IEEE, 2008, pp. 413–422.
4. Chandola V., Banerjee A., Kumar V. Anomaly Detection: A Survey. ACM Computing Surveys, 2009, vol. 41, no. 3, article 15.
5. Aggarwal C.C. (ed.). Outlier Analysis. 2nd ed. Springer, 2016.
6. Martynova M.D. Учебная аналитика в образовательном процессе: куда нас ведут большие данные. Открытое образование, 2024, № 3, с. 34–47.
7. Баранников К.А. Аналитика обучения как способ повышения эффективности системы управления образованием. Открытое образование, 2020, № 4, с. 20–32.
8. Siemens G., Long P. Penetrating the Fog: Analytics in Learning and Education. EDUCAUSE Review, 2011, vol. 46, no. 5, pp. 31–40.
9. Gasevic D., Dawson S., Siemens G. Let's Not Forget: Learning Analytics Are about Learning. TechTrends, 2015, vol. 59, no. 1, pp. 64–71.

10. Ahmed M., Mahmood A., Hu J. A Survey of Network Anomaly Detection Techniques. *Journal of Network and Computer Applications*, 2016, vol. 60, pp. 19–31.
11. Summerfield M. *Rapid SQL Server Security: AI-Driven Anomaly Detection*. DZone, 2022.
12. Pedregosa F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2011, vol. 12, pp. 2825–2830.
13. Долгова Е.Г., Кайсина А.Р. Образовательная аналитика как инструмент управления качеством образовательного процесса в вузе. *Университетское управление: практика и анализ*, 2019, № 3, с. 90–101.
14. Pimentel M.A.F., Clifton D.A., Clifton L., Tarassenko L. A Review of Novelty Detection. *Signal Processing*, 2014, vol. 99, pp. 215–249.
15. Johnson L., Adams Becker S., Cummins M., Estrada V. *NMC Horizon Report: 2016 Higher Education Edition*. The New Media Consortium, 2016.