# Лабораторная работа 6. Поиск аномалий

Поиск аномалий – набор приёмов и техник по выделению из набора данных отдельных экземпляров, которые не соответствуют привычному поведению по сравнению со всеми остальными экземплярами в наборе. В англоязычных источниках для аномальных экземпляров принято использовать термин «outlier» (посторонний). Поиск аномалий имеет множество различных областей применения, в том числе и в обеспечении информационной безопасности. Очень часто задача поиска аномалий применяется для выявления необычного поведения при мониторинге системы, что может служить неявным индикатором возможной атаки или мошеннической активности, поэтому анализ на поиск аномалий имеет широкое распространение в системах мониторинга сетевого трафика для фаерволлов, отслеживания поведения системных процессов для антивирусов, а также доменного анализа информационной безопасности в конкретной области.

Выделяют 3 различных вида аномалий, которые могут присутствовать в данных: точечные, контекстные и коллективные. Точечные аномалии наблюдаются в случае, если единственный экземпляр данных статистически в значительной степени отличается от остальных данных или слишком сильно удалён от них по некоторому признаку. Контекстные аномалии имеют место преимущественно в анализе временных рядов и наблюдаются в том случае, если отдельный экземпляр данных в целом распространён по сравнению со всем набором данных, но является аномальным при сравнении с непосредственной временной окрестностью – например, при мониторинге активности запущенных процессов использование процессом значительного объема оперативной памяти может быть ожидаемым в рабочие часы, но будет аномальным ночью. Контекстные аномалии в значительной степени зависят от высокоуровневых «сезонных» зависимостей (временных периодичностей) в наборе данных. Коллективные аномалии имеют место в том случае, если в отдельно взятой группе экземпляров каждый из них не является точечной аномалией относительно друг друга, но при этом вся группа проявляет аномальное поведение относительно остального набора данных.

Задача поиска аномалий в значительной степени перекликается со следующими задачами обработки данных:

1. Устранение шума. В отдельных случаях «шумом» можно считать те данные, которые являются аномальными относительно остального набора. Тем не менее, следует отметить, что, в зависимости от специфики задачи, аномальные экземпляры, наоборот, могут нести

наибольшую информацию, предоставляющую интерес, и их устранение может затруднить её решение.

2. Поиск новых закономерностей. В некоторых задачах аномальные данные могут являться новым, ранее не наблюдаемым, но ожидаемым поведением системы, которое может быть интересно для анализа (например, рост посещений интернет-ресурсов с новостной информацией о вирусных инфекциях из-за ухудшения эпидемиологической обстановки).

Наиболее известными и старыми методом поиска аномалий являются методы на основе фильтров низких частот. Такие методы подразумевают, что любое аномальное поведение, по сравнению с остальным набором данных, будет встречаться значительно реже, поэтому частота появления таких экземпляров или их окрестности будет очень низкой. Также зачастую, особенно при анализе временных рядов, фильтрация низких частот осуществляется не по всему набору, а по некоторому окну с усреднением.

Фильтрация низких частот применима к данным, в которых отдельные выбросы могут считаться аномальными. Тем не менее, у такого подхода есть ряд недостатков. Во-первых, сами данные могут содержать некоторый шум, и в этом случае при фильтрации низких частот отсутствует критерий, по которому шум можно отличить от аномального экземпляра. Во-вторых, определение того, какое именно поведение можно считать аномальным, может меняться в рамках набора данных с течением времени и проявлять некоторую «сезонность», для обнаружения которой требуется корректным образом подобрать ширину окна.

В последнее время наибольшее распространение для решения задачи поиска аномалий нашли подходы, основанные на машинном обучении. В частности, при поиске аномалий часто применяются методы машинного обучения без учителя, самостоятельно или в комбинации с методами машинного обучения с учителем.

В отличие от машинного обучения с учителем, задачей машинного обучения без учителя является анализ данных, для которых не определен выходной результат. Таким образом, машинное обучение без учителя предоставляет ряд методов для решения задачи кластеризации. Задача кластеризации в значительной степени отличается от задачи классификации. При классификации в исходном наборе (обучающей выборке) для каждого экземпляра входных параметров известен результат классификации – ожидаемый выходной класс, а сама задача заключается в переходе от частных, заранее размеченных данных к обобщенному классификатору, который способен аналогичным образом размечать ранее неизвестные данные. При кластеризации для каждого экземпляра известен лишь набор его входных

параметров, а задача заключается в введении некоторого разбиения этого набора на некоторое количество кластеров. При этом количество кластеров, в зависимости от метода, может задаваться вручную или определяться динамически, а интерпретация полученного разбиения, т.е. определение того, что означает тот или иной кластер, осуществляется отдельно.

При рассмотрении применимости различных методов машинного обучения без учителя для возможности «отделения» определенного скопления точек друг от друга часто используются различные механизмы понижения размерности. Наиболее распространенными являются метод главных компонент (PCA) и метод независимых компонент (ICA). В некоторых случаях для понижения размерности также можно использовать специализированные нейросетевые архитектуры, например, автоэнкодеры. Кроме того, снижение размерности до 2 является удобным инструментом для визуализации данных. Для визуализации наиболее распространен подход стохастического вложения соседей с t-распределением (t-distributed Stochastic Neighbor Embedding, t-SNE), который позволяет визуализировать точки на двумерной плоскости. Ниже приведен пример реализации функции `tsne_plot` для двумерной визуализации данных, размеченных на бинарную классификацию, с использованием класса `TSNE` модуля `sklearn.manifold` библиотеки scikit-learn, а также библиотеки matplotlib.

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np

def tsne_plot(X, y):
    tsne = TSNE(n_components=2)

    X_transformed = tsne.fit_transform(X)
    plt.figure(figsize=(12, 8))
    plt.scatter(
        X_transformed[np.where(y == 0), 0],
        X_transformed[np.where(y == 0), 1],
        marker='o',
        color='g',
        linewidth='1',
        alpha=0.8
    )
    plt.scatter(
        X_transformed[np.where(y == 1), 0],
        X_transformed[np.where(y == 1), 1],
        marker='o',
        color='r',
```

```
        linewidth='1',
        alpha=0.8
    )

    plt.show()
```

Представленная функция визуализирует набор данных X с размерностью, уменьшенной до 2, на плоскости в виде графика разброса точек (scatterplot), на котором точки, которым соответствует значение y=0, обозначены зеленым цветом, а точки, которым соответствует значение y=1 – красным. По внешнему виду полученной визуализации можно оценить, являются ли точки линейно разделимыми – например, если на полученной двумерной визуализации можно отделить зеленые точки от красных некоторой простой кривой, это значит, что для реализации классификации по этим точкам можно использовать линейный классификатор, например, логистическую регрессию.

Рассмотрим основные методы и приёмы, наиболее распространенные в задаче поиска аномалий.

**Isolation Forest**

Isolation Forest (изоляционный лес, в оригинальном исследовании обозначается как iForest) – метод машинного обучения без учителя для поиска аномалий, концептуально схожий с методом случайных лесов (Random Forest) машинного обучения с учителем. Как и случайный лес, изоляционный лес подразумевает создание набора деревьев принятия решений, после чего вдоль каждого дерева определяется глубина, которая необходима для изоляции отдельного элемента данных. Под «изоляцией» экземпляра подразумевается отделение последовательными разбиениями пространства признаков по осям таким образом, чтобы в одной части пространства оказался только этот экземпляр, а в другой – все остальные. После того, как лес обучен изолировать каждую точку, присутствующую в наборе, используется эвристическое предположение о том, что для изоляции обычных (неаномальных) экземпляров требуется намного больше разбиений, чем для изоляции аномальных экземпляров, т.е. глубина изоляционного леса для аномальных экземпляров будет значительно меньше, чем для обычных – поскольку неаномальные экземпляры будут находиться достаточно близко друг к другу, потребуется большое количество разбиений, чтобы отделить одну точку от плотно расположенных соседних точек; в то же время, аномальные экземпляры будут достаточно удалены от других данных, чтобы их можно было изолировать небольшим количеством разбиений. Как правило, для экономии памяти и повышения производительности при построении

изоляционного дерева его глубина искусственно ограничивается величиной порядка $\log_2 n$, где $n$ – количество элементов в исходной выборке, на основании предположения о том, что если для изоляции отдельного экземпляра требуется больше разбиений, он, скорее всего, уже не является аномальным, поэтому строить полное разбиение не имеет смысла. Таким образом, изоляционные леса считаются достаточно «легковесным» алгоритмом с точки зрения вычислительной эффективности и потребления памяти.

В python для работы с изоляционными лесами можно использовать класс `IsolationForest` из модуля `sklearn.ensemble` библиотеки scikit-learn (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html):

```
from sklearn.ensemble import IsolationForest

isolation_forest = IsolationForest(n_estimators=100,
max_samples=len(X))
isolation_forest.fit(X)
y_isolation_forest = isolation_forest.predict(X) == -1
```

Обратите внимание, что, будучи методом машинного обучения без учителя, изоляционный лес обучается только на входных данных. Работу `IsolationForest` можно регулировать параметрами `n_estimators` и `contamination`. Кроме того, для алгоритмов поиска аномалий в scikit-learn принято на выходе формировать набор из значений {-1; 1}, где -1 обозначает аномальный экземпляр, а 1 – обычный, поэтому для сведения результата к бинарной классификации (0 – обычный, 1 – аномалия) можно использовать логическое сравнение вектора с -1; при наличии ранее размеченной информации об аномалиях в дальнейшем для анализа качества поиска аномалий можно использовать стандартные метрики бинарной классификации (precision, recall, F1 и т.п.).

**Local Outlier Factor**

Local Outlier Factor (локальный уровень выброса, LOF) – алгоритм машинного обучения без учителя для поиска аномалий, концептуально схожий с методом kNN (k ближайших соседей). Для оценки «аномальности» отдельного экземпляра по сравнению с его окрестностями используется понятие локальной плотности относительно некоторой метрики (расстояния) в пространстве признаков (евклидова, Махаланобиса, Минковского и т.п.). Локальная плотность определяется как величина, обратная расстоянию вокруг точки, в пределах которого находятся как минимум $k$ других соседних точек.

Для точек, которые расположены достаточно плотно и близко друг к другу, расстояние, включающее в себя $k$ соседних точек, будет небольшим, соответственно значение локальной плотности будет высоким. Для точек, которые значительно удалены от скоплений других точек, расстояние до $k$ соседних точек будет высоким, поэтому локальная плотность будет низкой. После определения локальной плотности для всех точек полученные плотности могут быть усреднены или каким-либо другим образом сравнены друг с другом. Аномалиями можно считать те точки, для которых локальная плотность значительно ниже, чем средняя по всему набору данных.

Для работы с методом локального уровня выброса можно использовать класс `LocalOutlierFactor` из модуля `sklearn.neighbors` библиотеки scikit-learn (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html):

```
from sklearn.neighbors import LocalOutlierFactor

local_outlier_factor = LocalOutlierFactor(n_neighbors=20, leaf_size=30, metric='minkowski', p=2)

y_laf = local_outlier_factor.fit_predict(X) == -1
```

Основными параметрами, влияющими на работу `LocalOutlierFactor`, являются `n_neighbors`, `leaf_size`, `metric`, `contamination`, `algorithm` и другие. При этом для конкретного набора данных важно корректно подобрать количество соседних точек $k$ (`n_neighbors`) и выбрать правильное расстояние (`metric`). Как и в случае с изоляционными лесами, выходным значением является вектор значений {-1; 1}.

**One Class SVM**

One Class SVM (одноклассовый метод опорных векторов) – метод машинного обучения без учителя, основанный на построении классификатора с использованием метода опорных векторов, который аппроксимирует принадлежность к неизвестному статистическому распределению экземпляров в наборе данных. В отличие от традиционного метода опорных векторов, для исходных данных целевой «класс» не задаётся вручную, а вместо этого выводится по виду самих данным с использованием специализированной функции оценки плотности точек в окрестности. Функция оценки плотности концептуально схожа с оценкой локальной плотности, используемой в методе локального уровня выброса, и основывается на статистической оценке величины «расстояние до других точек». Аналогично с оценкой локальной плотности, вводится предположение

о том, что аномальные точки будут расположены дальше от всех остальных, поэтому величина расстояния до других точек будет статистически «скошена» по сравнению со всеми остальными значениями. Полученные отклонения центрируются и нормируются, что позволяет для каждой точки ввести значение [0;1], где значения, близкие к 0, означают, что точка находится в плотной окрестности других точек, а значения, близкие к 1, означают, что точка удалена от скоплений других точек. Эти значения используются в качестве «выходного» значения для обучения классификатора по методу опорных векторов с некоторым линейным или нелинейным ядром.

Для работы с одноклассовым методом опорных векторов можно использовать класс `OneClassSVM` модуля `sklearn.svm` библиотеки scikit-learn (https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html):

```
from sklearn.svm import OneClassSVM

svm = OneClassSVM(kernel='rbf', degree=3, gamma=0.1, nu=0.05)
svm.fit(X)
y_svm = svm.predict(X) == -1
```

Основными параметрами, влияющими на работу `OneClassSVM`, являются `kernel`, `degree`, `gamma`, `nu` и другие. Как и для других методов поиска аномалий, выходным значением является вектор значений {-1; 1}. Следует отметить, что для произвольного набора данных одноклассовый метод опорных векторов обладает всеми теми же недостатками, что и традиционный метод опорных векторов. В частности, существенным недостатком является как минимум квадратическая асимптотическая сложность обучения от объема входных данных, что делает данный метод трудно применимым для анализа большого количества экземпляров.

**Понижение размерности с помощью автоэнкодера и бинарная классификация**

Часто задача поиска аномалий может быть сформулирована как задача машинного обучения с учителем, т.е. в виде некоторого набора данных, для которого известно, какие экземпляры необходимо считать аномальными. Тем не менее, при такой постановке задачи значительную трудность представляет «скошенность» классов –аномальные экземпляры в размеченном наборе могут составлять доли процента от всех имеющихся экземпляров. В таких случаях удобно использовать выборку подмножеств, а также методы обучения без учителя и нелинейного понижения размерности в качестве этапа предварительной обработки перед использованием традиционного метода машинного обучения с учителем.

Для формирования подмножества в условиях «скошенных» классов удобно использовать все аномальные элементы, присутствующие в наборе, в совокупности с некоторой выборкой фиксированного размера из неаномальных элементов. Полученную выборку также можно визуализировать с использованием t-SNE:

```
anomaly_entries = dataset[dataset['Class'] == 1]
regular_entries = dataset[dataset['Class'] == 0]
subsample_size = 2000
regular_sample = regular_entries.sample(n=subsample_size)
dataset_subsample = anomaly_entries.append(regular_sample)
X_subsample = dataset_subsample.drop(columns=['Class']).values
y_subsample = dataset_subsample['Class'].values
tsne_plot(X_subsample, y_subsample)
```

Одним из наиболее распространенных подходов для комбинирования обучения без учителя и обучения с учителем является понижение размерности на основании автоэнкодера с последующей классификация некоторым простым методом, например, логистической регрессией.

Автоэнкодеры могут быть заданы естественным образом при помощи набора нейросетевых слоёв с «бутылочным горлышком» для формирования представлений исходных данных, которые в дальнейшем будет удобнее отделять, чем исходные данные. В случае поиска аномалий модель для формирования представлений обучают на неаномальных данных. Пример построения автоэнкодера с использованием функционального API Keras представлен ниже:

```
from tensorflow.keras import layers, losses, metrics, optimizers,
regularizers
from tensorflow.keras.models import Model

input_layer = layers.Input(shape=(X.shape[1],))
encoded = layers.Dense(100, activation='tanh')(input_layer)
encoded = layers.Dense(50, activation='relu')(encoded)

decoded = layers.Dense(50, activation='tanh')(encoded)
decoded = layers.Dense(100, activation='tanh')(decoded)

output_layer = layers.Dense(X.shape[1], activation='relu')(decoded)

autoencoder = Model(input_layer, output_layer)
autoencoder.compile(
  optimizer='adadelta', loss='mse', metrics=['accuracy']
```

```
)
autoencoder.summary()
```

Для обучения автоэнкодера в качестве ожидаемого выходного значения используются те же данные, которые подаются на вход. В случае поиска аномалий представления обучаются только на неаномальных экземплярах, поэтому в размеченных данных их необходимо предварительно извлечь:

```
autoencoder.fit(regular_sample, regular_sample,
                batch_size = 256, epochs = 10,
                shuffle = True, validation_split = 0.20)
```

Для поиска эффективного представления, как правило, не требуется большое количество итераций.

После обучения автоэнкодера можно создать дополнительную искусственную модель, содержащую все слои обученного автоэнкодера до «бутылочного горлышка», чтобы преобразовать исходные данные:

```
from tensorflow.keras.models import Sequential

autoencoder_encode = Sequential()
autoencoder_encode.add(autoencoder.layers[0])
autoencoder_encode.add(autoencoder.layers[1])
autoencoder_encode.add(autoencoder.layers[2])

X_encoded = autoencoder_encode.predict(X_subsample)
tsne_plot(X_encoded, y_subsample)
```

При удачном обучении автоэнкодера результат визуализации с помощью t-SNE должен показать, что аномальные точки в форме представления автоэнкодера являются намного лучше отделимыми от остальных, чем до преобразования. Полученные представления могут в дальнейшем использоваться в качестве входных для простого метода обучения с учителем, например, логистической регрессией. Кроме того, их также можно разделить на обучающую и тестовую выборку для проверки качества полученной модели.

Для классификации отдельного экземпляра данных в соответствии с разработанной моделью необходимо сначала преобразовать его в форму представления автоэнкодера с помощью autoencoder_encode.predict, а затем полученное представление подать на вход обученному классификатору.

В качестве примера задачи поиска аномалий в данной работе предлагается рассмотреть датасет с данными по мошенническим операциям с кредитными карточками, доступный по ссылке https://www.kaggle.com/mlg-ulb/creditcardfraud. В качестве входных параметров в датасете фигурирует время операции, сумма операции, а также 28 значений, полученных из исходных данных понижением размерности по методу главных компонент. Особенностью рассматриваемого датасета является крайне высокая «скошенность» классов – аномалии составляют лишь 0,17% от всего объема, поэтому точность является некорректной метрикой при анализе результатов поиска аномалий как бинарной классификации. Для упрощения обработки рекомендуется использовать все аномальные и подмножество неаномальных экземпляров из датасета.

**Задание к лабораторной 6.**

1. Загрузите исходный датасет и проанализируйте его. Опишите его характеристики. Предложите варианты для преобразования колонки Time.
2. Попробуйте применить методы бинарной классификации, рассмотренные в лабораторной работе 1, к решению задачи на всём наборе данных. Объясните полученные результаты.
3. Реализуйте для подмножества рассматриваемого датасета поиск аномалий с использованием метода в соответствии с вашим вариантом:

| № варианта | Используемый метод для решения |
|---|---|
| 1 | Изоляционный лес |
| 2 | Локальный уровень выброса |
| 3 | Одноклассовый метод опорных векторов |
| 4 | Изоляционный лес |
| 5 | Локальный уровень выброса |
| 6 | Одноклассовый метод опорных векторов |

4. Оцените полученные результаты с помощью метрик бинарной классификации.
5. Для рассмотренного датасета реализуйте извлечение представлений с помощью автоэнкодера и последующую классификацию по представлениям с использованием логистической регрессии.
6. Представьте ваше исследование в виде Блокнота JupyterLab, содержащего все пункты задания.