ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ГОРОДА МОСКВЫ
Юго-Западный административный округ

Государственное бюджетное общеобразовательное учреждение города Москвы

«Школа № 1533 «ЛИТ»

Техническое задание на выполнение выпускного проекта

Анализ финансовых аномалий с помощью нейросетевого автоэнкодера

по специальности *Программирование*

Исполнитель *Кармаев Александр Андреевич, Мукосей Филипп Ильич группа 10. 5*

Заказчик проекта *Суринов Андрей*

Руководитель *Нет*

Консультант(ы) *Нет*

Москва, 2020

Содержание

Введение

В наши дни деньги являются важнейшим ресурсом, ведь на них можно купить почти всё что угодно. И очень многие финансовые операции производятся с помощью безналичного расчета. Именно поэтому важно уметь защищать деньги от мошенников, способы кражи которых постоянно развиваются. Даже если не брать в расчет крупные операции, мошенничество в малых транзакциях достигает огромных размеров за небольшой промежуток времени. Поэтому создание способов борьбы с мошенничеством, их совершенствование и сравнение различных методик крайне полезно для программистов, работающих в банках и “оберегающих наш кошелёк”.

Постановка задачи, актуальность, целевая аудитория, аналоги

Актуальность и целевая аудитория

Данный проект будет весьма полезен для всей банковской системы по защите пользователей от мошенников. Он не только представляет из себя уже готовый продукт, который может быть применён для выявления мошенничества, но и дает советы: какой способ обнаружения мошенничеств лучше и как можно улучшить модели для его обнаружения.

Постановка задачи

1. Выявить наиболее эффективную модель классификации финансовых операций путем сравнения моделей и написать программу, реализующую функцию выявления мошенничества.
2. Реализовать определитель мошеннических транзакций с помощью наиболее эффективной модели классификации.
3. Исследование
    a. Реализация
    b. Улучшение
    c. Сравнение

Аналоги

Статья на habr:

habr.com/ru/company/nix/blog/478286/

Преимущества нашего продукта:

- Показание наглядной эффективности на графиках
- Улучшение моделей

Другая статья на habr:

habr.com/ru/company/ruvds/blog/488342/

Преимущества нашего продукта:

- Использование большего числа моделей
- Показание наглядной эффективности на графиках

Решение

Анализ предметной области

Транзакция — в общем случае, любая сделка с использованием банковского счета. Различают онлайн-транзакции, выполняющиеся в режиме реального времени между всеми заинтересованными сторонами, и офлайн-транзакции.

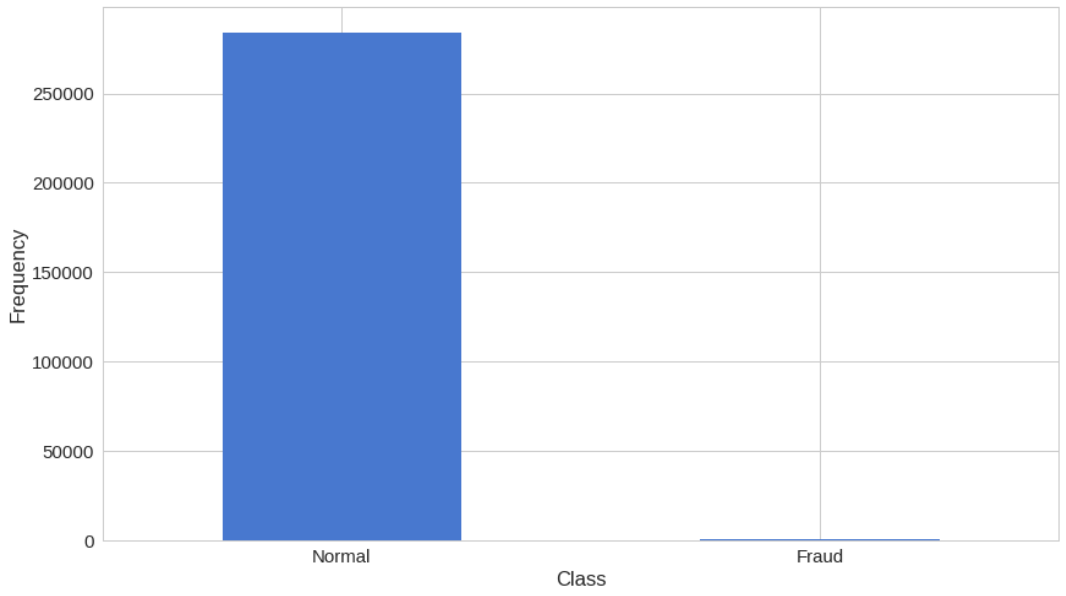
В таблице ниже показаны примеры транзакций из используемого датасета

Датасет — это обработанная и структурированная информация в табличном виде (выборка). Строки такой таблицы называются объектами, а столбцы – признаками.

Название столбца	не мошенничество	мошенничество
V1	1.017055	-3.043541
V2	-0.185049	-3.157307
V3	1.181666	1.088463
V4	1.073490	2.288644
V5	-0.550586	1.359805
V6	0.971827	-1.064823
V7	-0.806452	0.325574
V8	0.484792	-0.067794
V9	0.517268	-0.270953
V10	-0.060124	-0.838587
V11	1.446990	-0.414575
V12	1.198013	-0.503141
V13	-0.070891	0.676502
V14	0.006577	-1.692029

V15	0.432215	2.000635
V16	-0.149873	0.666780
V17	-0.090841	0.599717
V18	-0.195277	1.725321
V19	-0.870041	0.283345
V20	-0.164293	2.102339
V21	0.239287	0.661696
V22	0.875848	0.435477
V23	-0.053458	1.375966
V24	-0.269918	-0.293803
V25	0.328907	0.279798
V26	-0.217332	-0.145362
V27	0.096208	-0.252773
V28	0.020796	0.035764
Amount	17.570000	529.000000
Class	0.000000	1.000000
Time	472.000000	472.000000

Распределение классов



Обучение — процесс уточнения модели, основанный на обработке обучающей выборки.

Гиперпараметры — параметры, значения которых задается до начала обучения модели и не изменяется в процессе обучения. У модели может не быть гиперпараметров.

Оптимизация — модификация системы для повышения её эффективности. Хотя целью оптимизации является получение оптимальной системы, истинно оптимальная система в процессе оптимизации достигается далеко не всегда.

Методы обучения:

- ***с учителем*** — при обучении с учителем машина обучается на примерах. Оператор обеспечивает алгоритм машинного обучения набором известных данных, который содержит необходимые входные и выходные значения. Алгоритм должен установить, как получаются по данным входам данные выходы. Сам оператор знает решение поставленной задачи; алгоритм выявляет закономерности в данных учится на основе наблюдений и делает прогнозы. Эти прогнозы затем корректируются оператором. Процесс продолжается до тех пор, пока алгоритм не достигнет высокого уровня точности / производительности.

- ***без учителя*** — в этом случае алгоритм машинного обучения изучает данные с целью выявления закономерностей (паттернов). Не существует справочника с ответами или оператора, который мог бы обучить машину. Напротив, программа сама определяет корреляции и связи на основе анализа доступных данных. При обучении без учителя алгоритму машинного обучения позволено самостоятельно интерпретировать большие наборы данных и делать на их основе выводы. Алгоритм пытается каким-либо образом упорядочить данные и описать их структуру. Это может выглядеть как группировка данных в кластеры или это такое упорядочивание данных, при котором они начинают выглядеть систематизировано.

ROC-кривая (англ. receiver operating characteristic, рабочая характеристика приёмника) — график, позволяющий оценить качество бинарной классификации, отражает соотношение между долей объектов от общего количества носителей признака, верно классифицированных как несущие признак (англ. true positive rate, TPR, называемой чувствительностью алгоритма классификации), и долей объектов от общего количества объектов, не несущих признака, ошибочно классифицированных как несущие признак (англ. false positive rate, FPR, величина 1-FPR называется специфичностью алгоритма классификации) при варьировании порога решающего правила. Также известна как кривая ошибок. Анализ классификаций с применением ROC-кривых называется ROC-анализом.

Перекрёстная проверка (кросс-проверка, скользящий контроль, англ. *cross-validation*) — метод оценки аналитической модели и её поведения на независимых данных. При оценке модели имеющиеся в наличии данные разбиваются на k частей. Затем на $k-1$ частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз; в итоге каждая из k частей данных используется для тестирования. В результате получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных.

Методы решения

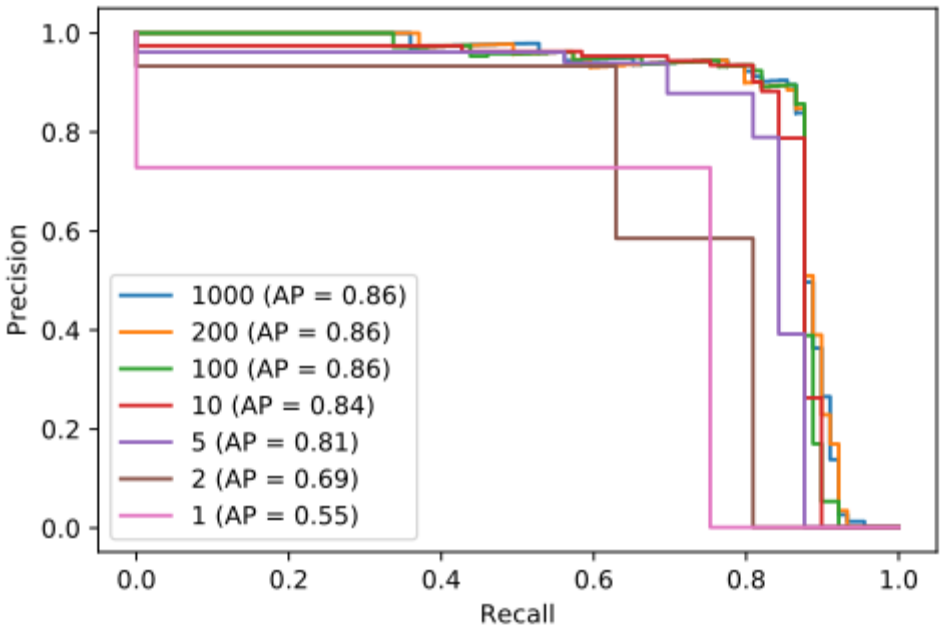
Случайный лес

Случайный лес — модель, состоящая из множества деревьев решений. Вместо того, чтобы просто усреднять прогнозы разных деревьев (такая концепция называется просто «лес»), эта модель использует две ключевые концепции, которые и делают этот лес случайным:

1. Случайная выборка образцов из набора данных при построении деревьев.
2. При разделении узлов выбираются случайные наборы параметров.

Дерево решений — интуитивно понятная базовая единица алгоритма случайный лес. Мы можем рассматривать его как серию вопросов да / нет о входных данных. В конечном итоге вопросы приводят к предсказанию определённого класса (или величины в случае регрессии). Это интерпретируемая модель, так как решения принимаются так же, как и человеком: мы задаём вопросы о доступных данных до тех пор, пока не приходим к определенному решению (в идеальном мире).
Базовая идея дерева решений заключается в формировании запросов, с которыми алгоритм обращается к данным. Дерево решений формирует узлы, содержащие большое количество образцов (из набора исходных данных), принадлежащих к одному классу. Алгоритм старается обнаружить параметры со сходными значениями.

Сравнительный график с разным количеством деревьев



Изолирующий лес

Изолирующий лес — это неконтролируемый алгоритм обучения для обнаружения аномалий, который работает по принципу изолирования аномалий , а не по наиболее распространенным методам профилирования нормальных точек. Вместо того, чтобы пытаться построить модель нормальных экземпляров, он явно изолирует аномальные

точки в наборе данных. Главным преимуществом такого подхода является возможность использования методов выборки до такой степени, которая не допускается профильными методами, создавая очень быстрый алгоритм с низкой потребностью в памяти.

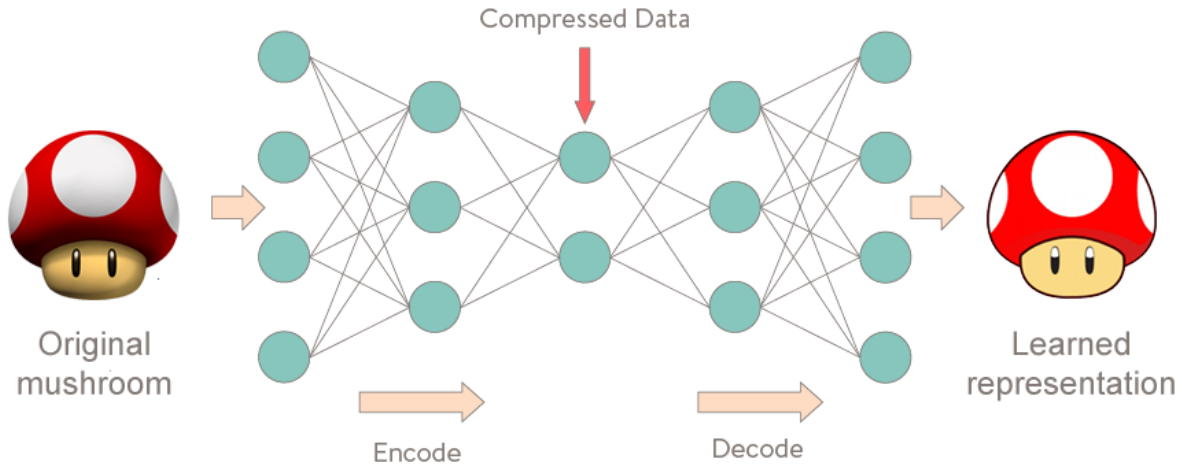
В основе алгоритма изолированного леса лежит тенденция к тому, что аномальные экземпляры в наборе данных легче отделить от остальной выборки (изолировать) по сравнению с нормальными точками. Чтобы изолировать точку данных, алгоритм рекурсивно создает секции на выборке, случайным образом выбирая атрибут, а затем случайным образом выбирая значение разделения для атрибута между минимальным и максимальным значениями, разрешенными для этого атрибута.

Автоэнкодер

Автоэнкодер *(автокодировщика)* — специальная архитектура искусственных нейронных сетей, позволяющая применять обучение без учителя при использовании метода обратного распространения ошибки. Простейшая архитектура автокодировщика — сеть прямого распространения, без обратных связей, содержащая входной слой, промежуточный слой и выходной слой. Выходной слой автокодировщика должен содержать столько же нейронов, сколько и входной слой.

Основной принцип работы и обучения сети автокодировщика — получить на выходном слое отклик, наиболее близкий к входному. Чтобы решение не оказалось тривиальным, на промежуточный слой автокодировщика накладывают ограничения: промежуточный слой должен быть или меньшей размерности, чем входной и выходной слои, или искусственно ограничивается количество одновременно активных нейронов промежуточного слоя — разрежённая активация. Эти ограничения заставляют нейросеть искать обобщения и корреляцию в поступающих на вход данных, выполнять их сжатие. Таким образом, нейросеть автоматически обучается выделять из входных данных общие признаки, которые кодируются в значениях весов искусственной нейронной сети. Так, при обучении сети на наборе различных входных изображений, нейросеть может самостоятельно обучиться распознавать линии и полосы под

различными углами.



Метрики

Метрики используются для оценки качества моделей и сравнения различных алгоритмов.

Перед переходом к самим метрикам необходимо ввести важную концепцию для описания этих метрик в терминах ошибок классификации — confusion matrix (матрица ошибок). Допустим, что у нас есть два класса и алгоритм, предсказывающий принадлежность каждого объекта одному из классов, тогда матрица ошибок классификации будет выглядеть следующим образом:

	y = 1	y = 0
y^ = 1	True Positive (TP)	False Positive (FP)

y^ = 0	False Negative (FN)	True Negative (TN)

Здесь **y^** — это ответ алгоритма на объект. а **y** — истинная метка класса на этом объекте.

ROC-AUC

При конвертации вещественного ответа алгоритма (как правило, вероятности принадлежности к классу) в бинарную метку, мы должны выбрать какой-либо порог, при котором 0 становится 1. Естественным и близким кажется порог, равный 0.5, но он не всегда оказывается оптимальным, например, при вышеупомянутом отсутствии баланса классов.

Одним из способов оценить модель в целом, не привязываясь к конкретному порогу, является AUC-ROC (или ROC AUC) — площадь (Area Under Curve) под кривой ошибок (Receiver Operating Characteristic curve). Данная кривая представляет из себя линию от (0; 0) до (1; 1) в координатах True Positive Rate (TPR) и False Positive Rate (FPR): значение менее 0,5 говорит, что классификатор действует с точностью до наоборот: если положительные назвать отрицательными и наоборот, классификатор будет работать лучше.

$$TPR = \frac{TP}{TP+FN}$$

$$FPR = \frac{FP}{FP+TN}$$

ROC-PR

Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота).

Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

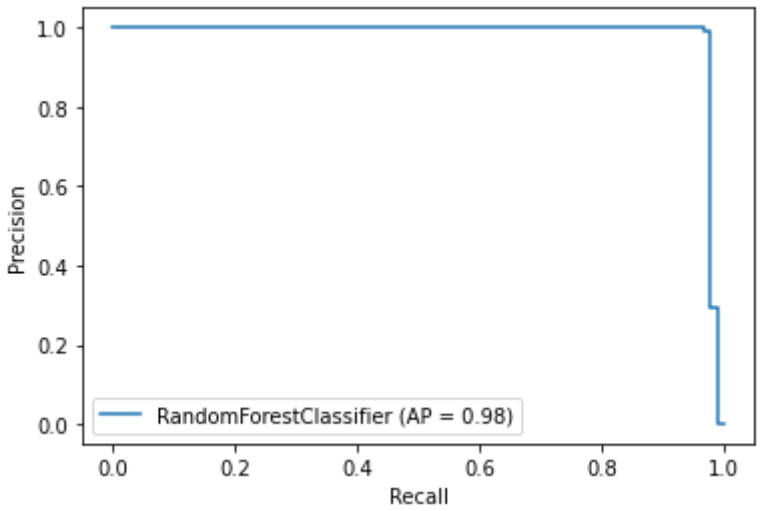
Именно введение precision не позволяет нам записывать все объекты в один класс, так как в этом случае мы получаем рост уровня False Positive. Recall демонстрирует способность алгоритма обнаруживать данный класс вообще, а precision — способность отличать этот класс от других классов.

$$recall = \frac{TP}{TP + FN}$$

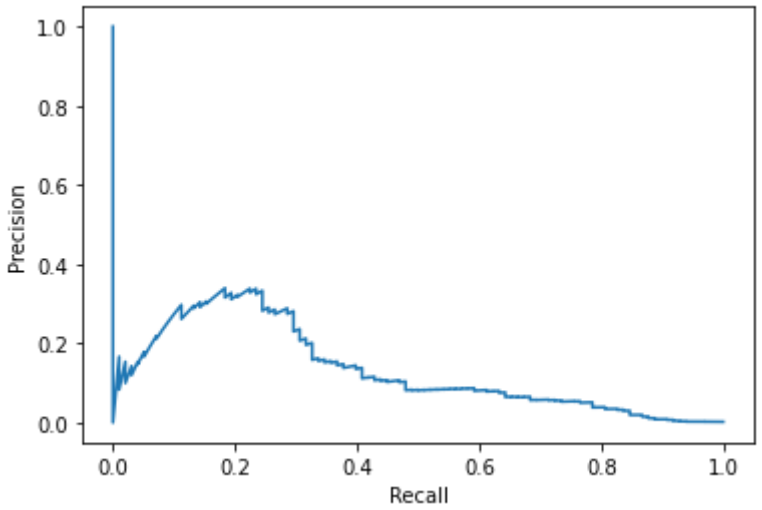
$$precision = \frac{TP}{TP + FP}$$

Результаты моделей

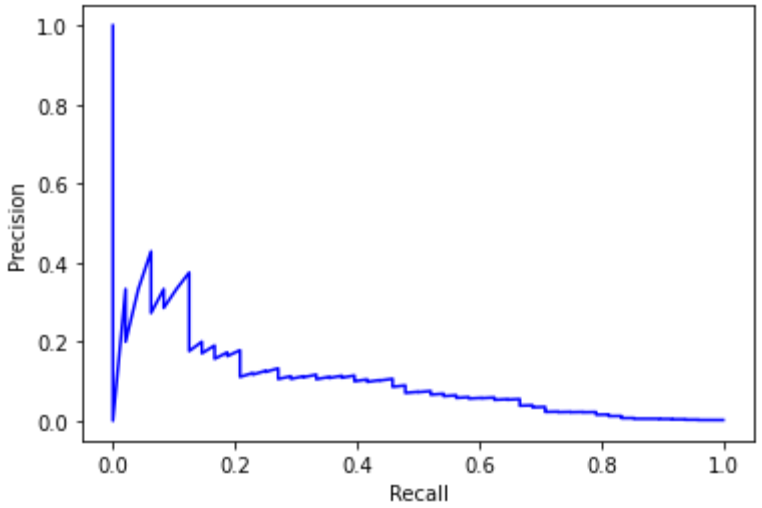
Random forest



Isolation forest



Autoencoder



Оптимизация моделей

Оптимизация гиперпараметров — задача машинного обучения по выбору набора оптимальных гиперпараметров для обучающего алгоритма.

Одни и те же виды моделей машинного обучения могут требовать различные предположения, веса или скорости обучения для различных видов данных. Эти параметры называются гиперпараметрами и их следует настраивать так, чтобы модель могла оптимально решить задачу обучения.

Поиск по решетке

Традиционным методом осуществления оптимизации гиперпарамеров является поиск по решётке (или вариация параметров), который просто делает полный перебор по заданному вручную подмножеству пространства гиперпараметров обучающего алгоритма. Поиск по решётке должен сопровождаться некоторым измерением производительности, обычно измеряемой посредством перекрёстной проверки на тренировочном множестве, или прогонкой алгоритма на устоявшемся проверочном наборе.

RandomizedSearch

RandomizedSearch — способ оптимизации гиперпараметров модели путём случайного их перебора. Особо эффективен при наличии малого числа гиперпараметров, оказывающих влияние на эффективность модели. Кроме того, позволяют использовать предварительные данные путём указания распределения для выборки случайных параметров.

Программная реализация

Используемые инструменты

- Язык реализации:
  - Python 3

- Библиотеки:
  - NumPy
  - Pandas
  - SciPy
  - TensorFlow
  - Keras
  - Matplotlib
  - Sklearn
  - joblib
  - RandomizedSearchCV
  - GridSearchCV

- Среды разработки:
    - Jupyter Notebook
    - PyCharm

Ход работы

Изначально был разработан бинарный классификатор - рандомный лес, и для него были построены графики ROC_AUC и ROC_PR. Эта часть проекта стала основой для последующего развития. Вслед за ним был реализован детектор аномалий - изолирующий лес, однако при попытке построения графиков возникли проблемы: т.к. для рандомного леса был использован модуль из библиотеки sklearn, применимый только для классификаторов. Поэтому мы залезли в исходники функций, строящих графики, и сделали свою версию по образу и подобию, подходящую под детектор аномалий. Далее был создан нейросетевой автоэнкодер и оценен на графиках. Также был применён один из методов оптимизации гиперпараметров - RandomizedSearch.

Результат

В конечном итоге мы имеем 3 реализованные модели: случайный лес, изолирующий лес и автоэнкодер. Использованы 2 типа метрик для оценки эффективности полученных моделей. Был прменён один из методов оптимизации гиперепараметров модели - RandomizedSearch.

Выводы

Данная тема крайне актуальна, так как в наши дни очень многие финансовые операции производятся с помощью безналичного расчета. Именно поэтому остро стоит проблема мошенничества и очень важно найти эффективные способы борьбы с ним. Даже если не брать в расчет крупные операции, размеры мошенничества в малых транзакциях достигают огромных размеров за небольшой промежуток времени. Для этой задачи лучше всего подходят детекторы аномалий, обучающиеся на одном из классов, без учителя. Это обусловлено тем, что количество мошеннических транзакций

сильно меньше количества легитимных, т.е. распределение классов в данных очень неравномерное, а получить размеченные данные сложно.

Перспективы дальнейшей разработки

Дальнейшая разработка может быть продолжена в следующих направлениях:

- Включение в сравнение других моделей
    - Данное исследование может быть расширено добавлением других моделей, таких, как one class SVM или логистическая регрессия.

- Применение альтернативных способов улучшение моделей
    - Помимо тех способов совершенствования моделей, которые применялись здесь, могут быть использованы и альтернативы. Также наилучшая эффективность может быть достигнута путём применения нескольких способов в связке. Более того, от последовательности их применения результаты могут меняться.

- усовершенствование способа просмотра сравнения (приложение)
    - Вместо нескольких файлов с исходным кодом и графиков с пояснениями этот проект может быть превращен в программу, где можно выбирать модели, способы улучшения и сравнивать их между собой. Графики буду строиться в режиме реального времени, а модели обучаться непосредственно на устройстве пользователя, или же на серверах.

- сравнение сетей в других задачах
    - Данное исследование можно распространить и на другие задачи, не

только финансовые аномалии. Главная особенность финансовых аномалий в том, что количество мошеннических транзакций сильно меньше количества легитимных, распределение классов в данных неравномерное. В любой задаче с похожим распределением результаты данного исследования будут полезны.

- работа с другим датасетом
    - В данной реализации используется единственный набор данных, но можно добавить и другие. Возможно, они будут другой структуры, возможно, с другим количеством столбцов в таблице. Этот пункт предполагает динамическое создание и обучение моделей на основе определенных датасетов, выбранных пользователем. Таким образом, если код из данной реализации будет использоваться в других проектах, можно будет легко получить модели, заточенные под конкретные входные параметры.

- работа базами данных
    - Помимо других датасетов, можно добавить поддержку баз данных, в том числе на удаленных серверах. Это позволит не скачивать датасеты, не переводить их в формат .csv, а использовать имеющиеся данные из базы. Если желаемая модель уже обучена и используется, подобное улучшение позволит сильно сократить использование ресурсов.

- обучение автоэнкодера на данных, классифицированных с помощью других моделей
    - В датасете, использованном в данном исследовании, уже имеется колонка, в которой определена принадлежность каждой транзакции к одному из классов (мошенническая транзакция или легитимная транзакция), однако в реальности достаточно сложно получить такое

количество наверняка достоверных данных, которых хватит для обучения продвинутой модели. Таким образом, для этого могут быть использованы более простые модели, на обучение которых нужно меньше данных. При правильно выставленном пороге можно получить почти гарантированно достоверные данные, на которых будет возможно обучить более сложную модель.

- интерпретация модели (важность признаков)
- создание приложения с использованием технологий SAP
    - На основе исходного кода данного исследования может быть создан модуль с использованием технологий SAP, который позволит тесно интегрировать данную функциональность с сервисами SAP.

Источники

- Создаем нейронную сеть | Рашид Тарик
- документации по используемым библиотекам и инструментам
- Статьи:
    - habr.com/ru/post/312450/
    - habr.com/ru/company/ods/blog/328372/
    - medium.com/@curiousily/credit-card-fraud-detection-using-autoencoders-in-keras-tensorflow-for-hackers-part-vii-20e0c85301bd
    - en.wikipedia.org/wiki/Receiver_operating_characteristic
    - en.wikipedia.org/wiki/Isolation_forest
- Исследования:
    - nilsonreport.com/mention/407/1link/
- Датасет:

- kaggle.com/dalpozz/creditcardfraud

Приложение

Загрузка данных

```
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path

# constants
LABELS = ['Normal', 'Fraud']
RANDOM_SEED = 105
UPDATE_SIZE = 0.3
TEST_SIZE = 0.2
DATASET_PATH = Path('../creditcard.csv')

# data reading
df = pd.read_csv(DATASET_NAME)

# describe dataset by classes
count_classes = pd.value_counts(df['Class'], sort=True)
count_classes.plot(kind='bar', rot=0)
plt.title('Transaction class distribution')
plt.xticks(range(2), LABELS)
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```

Подготовка и разделение данных

```
# data preparing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# преобразуем данные, убираем колонку 'время', т.к. она ни на что не влияет
```

```
data = df.drop(['Time'], axis=1)
data['Amount'] =
StandardScaler().fit_transform(data['Amount'].values.reshape(-1, 1))

# отделяем X от Y
Y = data['Class'].values
X = data.drop(['Class'], axis=1).values

# делим данные
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=split_size, random_state=RANDOM_SEED, shuffle=True)

X_test, X_comparison, Y_test, Y_comparison = train_test_split(
    X_test, Y_test, test_size=split_size, random_state=RANDOM_SEED,
shuffle=True)
```

Random forest

```
### Creating model
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=1000 n_jobs=-1)

### train model
model.fit(X_train, Y_train)
import joblib

file_name = 'model_1000.pkl'

### saving model
with open(file_name, 'wb') as f:
    joblib.dump(model, f, compress=1)

### loading model
with open(file_name, 'rb') as f:
    model = joblib.load(file_name)

predictions = model.predict_proba(X_test)

# подсчёт TPR и FPR, precision и recall
```

```
TP, FP, TN, FN = 0, 0, 0, 0

for prediction, true_class in zip(predictions, Y_test):
    TP += int(true_class and prediction)
    FN += int(true_class and not prediction)
    FP += int(not true_class and prediction)
    TN += int(not true_class and not prediction


print('TP:', TP)  # мошенничество, определено правильно
print('FP:', FP)  # нормальная транзакция, определена неправильно
print('TN:', TN)  # нормальная транзакция, определена правильно
print('FN:', FN)  # мошенничество, определено неправильно

precision = TP / (TP + FP)
recall = TP / (TP + FN)

print(precision)
print(recall)

tpr = TP / (TP + FN)  # True positive rate
fpr = FP / (FP + TN)  # False positive rate

print(tpr)
print(fpr)
```

Isolation forest

```
# Creating model
from sklearn.ensemble import IsolationForest
import numpy as np

model = IsolationForest(n_estimators=1000, n_jobs=-1)

predictions = model.decision_function(X_test)

# перевод значений в диапазон от 0 до 1
for i in range(len(predictions)):
    predictions[i] = (predictions[i] + 1) / -2

# проверка
```

```
print(min(predictions))
print(max(predictions))
```

Автоэнкодер

```
# Creating model
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers
from keras import metrics


input_dim = X_train.shape[1]
encoding_dim = 14

input_layer = Input(shape=(input_dim, ))

encoder = Dense(encoding_dim, activation="tanh",

activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoder = Dense(int(encoding_dim / 2), activation="relu")(encoder)

decoder = Dense(int(encoding_dim / 2), activation='tanh')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)

autoencoder = Model(inputs=input_layer, outputs=decoder)

nb_epoch = 40
batch_size = 32

autoencoder.compile(optimizer='adam',
                    loss='mean_squared_error',
                    metrics=['accuracy'])

checkpointer = ModelCheckpoint(filepath='model.h5',
                               verbose=0,
                               save_best_only=True)
tensorboard = TensorBoard(log_dir='./logs',
                          histogram_freq=0,
```

```
                          write_graph=True,
                          write_images=True)

history = autoencoder.fit(X_train, X_train,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(X_test, X_test),
                    verbose=1,
                    callbacks=[checkpointer, tensorboard]).history

model = load_model('model.h5')

# график эффективности по эпохам
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
```

Метрики

```
# отчёт
from sklearn.metrics import classification_report

report = classification_report(Y_test, predictions, target_names=LABELS)
print(report)

# график precision - recall
from sklearn.metrics import precision_recall_curve

precision, recall, thresholds = precision_recall_curve(Y_test, predictions)

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.plot(recall, precision)


# график ROC-AUC
```

```
from sklearn.metrics import RocCurveDisplay, roc_curve, auc

fpr, tpr, _ = roc_curve(Y_test, predictions)
roc_auc = auc(fpr, tpr)

viz = RocCurveDisplay(
    fpr=fpr, tpr=tpr, roc_auc=roc_auc, estimator_name=''
)
viz.plot(ax=None, name='test')
plt.plot(fpr, tpr, label='AUC = %0.4f'% roc_auc)
plt.show()
```

```
# считаем точность
from sklearn.metrics import average_precision_score
average_precision = average_precision_score(Y_test, predictions)
print(average_precision)
```

Поиск по сетке

```
from sklearn.model_selection import GridSearchCV

n_estimators = [600,400,700]
max_features = ['sqrt']
max_depth = [13, 14, 15]
min_samples_split = [12, 23, 50]
min_samples_leaf = [2, 18, 12]
bootstrap = [False]
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
gs = GridSearchCV(rs.best_estimator_, param_grid, cv = 1, verbose = 1,
n_jobs=-1)
gs.fit(X_train, Y_train)
gs.best_params_
```

RandomizedSearch

```
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1000,
num = 10)]
max_features = ['log2', 'sqrt']
max_depth = [int(x) for x in np.linspace(start = 1, stop = 15, num =
15)]
min_samples_split = [int(x) for x in np.linspace(start = 2, stop = 50,
num = 10)]
min_samples_leaf = [int(x) for x in np.linspace(start = 2, stop = 50,
num = 10)]
bootstrap = [True, False]
param_dist = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
rs = RandomizedSearchCV(RF_model,
                        param_dist,
                        n_iter = 50,
                        cv = 3,
                        verbose = 1,
                        n_jobs=-1,
                        random_state=0)
rs.fit(X_train, Y_train)
rs.best_params_


rs_df =
pd.DataFrame(rs.cv_results_).sort_values('rank_test_score').reset_index(
drop=True)
rs_df = rs_df.drop([
```

```
            'mean_fit_time',
            'std_fit_time',
            'mean_score_time',
            'std_score_time',
            'params',
            'split0_test_score',
            'split1_test_score',
            'split2_test_score',
            'std_test_score'],
            axis=1)
rs_df.head(10)
```