Geekbrains

**Прогнозирование аналитических данных бухгалтерского учета по текстовому описанию платежного документа**

Искусственный интеллект

Машинное обучение

Проторский А.В.

Москва

2024

**Содержание**

[Введение 2](#_Toc174439986)

[Теоретическая часть 3](#_Toc174439987)

[1. Введение 3](#_Toc174439988)

[2. Типы задач машинного обучения 3](#_Toc174439989)

[3. Основные виды машинного обучения 4](#_Toc174439990)

[4. Основные алгоритмы моделей машинного обучения 6](#_Toc174439991)

[5. Разделение данных 11](#_Toc174439992)

[6. Основные метрики оценки качества моделей машинного обучения 11](#_Toc174439993)

[7. Библиотека машинного обучения 15](#_Toc174439994)

[8. Системное окружение 15](#_Toc174439995)

[9. Архитектурный стиль 15](#_Toc174439996)

[Практическая часть 17](#_Toc174439997)

[Заключение 25](#_Toc174439998)

[Список используемой литературы 26](#_Toc174439999)

[Приложение 27](#_Toc174440000)

# Введение

**Тема проекта:** Обучить модель машинного обучения для предсказания аналитических параметров бухгалтерского учета по текстовому описанию платежного документа.

**Цель:** Автоматизировать формирования бухгалтерских документов при обработке банковских выписок, сократив труд программистов и сотрудников бухгалтерии при загрузке данных из системы клиент-банк в бухгалтерскую систему.

**Какую проблему решает:** При формировании бухгалтерских документов из платежных необходимо определять параметры аналитического учета, для чего требуется проанализировать текстовое назначение платежа, для этого используется разбор текста с помощью регулярных выражений и человеческий труд по корректировки ошибочно определенных данных с последующим изменением алгоритма их разбора. Поскольку текст платежного поручения заполняется сотрудником банка в произвольной форме процесс модификации алгоритма постоянно продолжается, чтобы минимизировать время сопровождения системы, было принято решения обучить модель для решения данной задачи.

**Задачи:**

1. Изучить литературу, касающуюся темы исследования.
2. Рассмотреть основные виды и методы моделей машинного обучения.
3. Разработать и обучить модель машинного обучения.
4. Создать WEB REST API сервис для использования обученной модели.

**Инструменты:** Python**,** Jupyter Notebook,VSCode, Sklearn, FLASK, SoupUI, Git, WSL, Docker, Gunicorn

**Состав команды**: Проторский А.В. (Разработчик)

# Теоретическая часть

# Введение

Благодаря машинному обучению программист не обязан писать инструкции, учитывающие все возможные проблемы и содержащие все решения. Вместо этого в программу закладывают алгоритм самостоятельного нахождения решений путём комплексного использования статистических данных, из которых выводятся закономерности и на основе которых делаются прогнозы.

Технология машинного обучения на основе анализа данных берёт начало в 1950 году, когда начали разрабатывать первые программы для игры в шашки. За прошедшие десятилетия общий принцип не изменился. Зато благодаря взрывному росту вычислительных мощностей компьютеров многократно усложнились закономерности и прогнозы, создаваемые ими, и расширился круг проблем и задач, решаемых с использованием машинного обучения.

Чтобы запустить процесс машинного обучение, для начала необходимо загрузить в компьютер набор данных, на которых алгоритм будет учиться обрабатывать запросы. Например, могут быть фотографии собак и котов, на которых уже есть метки, обозначающие к кому они относятся. После процесса обучения, программа уже сама сможет распознавать собак и котов на новых изображениях без содержания меток. Процесс обучения продолжается и после выданных прогнозов, чем больше данных мы проанализировали программой, тем более точно она распознает нужные изображения.

Благодаря машинному обучению компьютеры учатся распознавать на фотографиях и рисунках не только лица, но и пейзажи, предметы, текст и цифры. Что касается текста, то и здесь не обойтись без машинного обучения: функция проверки грамматики сейчас присутствует в любом текстовом редакторе и даже в телефонах. Причем учитывается не только написание слов, но и контекст, оттенки смысла и другие тонкие лингвистические аспекты.

# Типы задач машинного обучения

Все задачи, решаемые с помощью машинного обучения, относятся к одной из следующих категорий.

1. Задача регрессии – прогноз на основе выборки объектов с различными признаками. На выходе должно получиться вещественное число (2, 35, 76.454 и др.), к примеру цена квартиры, стоимость ценной бумаги по прошествии полугода, ожидаемый доход магазина на следующий месяц, качество вина при слепом тестировании.
2. Задача классификации – получение категориального ответа на основе набора признаков. Имеет конечное количество ответов (как правило, в формате «да» или «нет»): есть ли на фотографии кот, является ли изображение человеческим лицом..
3. Задача кластеризации – распределение данных на группы: разделение всех клиентов мобильного оператора по уровню платёжеспособности, отнесение космических объектов к той или иной категории (планета, звёзда, чёрная дыра и т. п.).
4. Задача уменьшения размерности – сведение большого числа признаков к меньшему (обычно 2–3) для удобства их последующей визуализации (например, сжатие данных).
5. Задача выявления аномалий – отделение аномалий от стандартных случаев. На первый взгляд она совпадает с задачей классификации, но есть одно существенное отличие: аномалии – явление редкое, и обучающих примеров, на которых можно обучить модель на выявление таких объектов, либо исчезающе мало, либо просто нет, поэтому методы классификации здесь не работают. На практике такой задачей является, например, выявление мошеннических действий с банковскими картами.

# Основные виды машинного обучения

Основная масса задач, решаемых при помощи методов машинного обучения, относится к двум разным видам: обучение с учителем (supervised learning) либо без него (unsupervised learning). Однако этим учителем вовсе не обязательно является сам программист, который стоит над компьютером и контролирует каждое действие в программе. «Учитель» в терминах машинного обучения – это само вмешательство человека в процесс обработки информации. В обоих видах обучения машине предоставляются исходные данные, которые ей предстоит проанализировать и найти закономерности. Различие лишь в том, что при обучении с учителем есть ряд гипотез, которые необходимо опровергнуть или подтвердить.

1. **Машинное обучение с учителем**

Предположим, в нашем распоряжении оказались сведения о десяти тысячах московских квартир: площадь, этаж, район, наличие или отсутствие парковки у дома, расстояние от метро, цена квартиры и т. п. Нам необходимо создать модель, предсказывающую рыночную стоимость квартиры по её параметрам. Это идеальный пример машинного обучения с учителем: у нас есть исходные данные (количество квартир и их свойства, которые называются признаками) и готовый ответ по каждой из квартир – её стоимость (целевая переменная). Программе предстоит решить задачу регрессии, предсказать цену квартиры в зависимости от ее признаков.

1. **Машинное обучение без учителя**

В случае обучения без учителя, когда готовых «правильных ответов» системе не предоставлено, можно сгруппировать объекты в кластеры, так чтобы внутри одного кластера объекты были более похожи друг на друга, чем на объекты из других кластеров. Например, у нас есть информация о весе и росте какого-то количества людей, и эти данные нужно распределить по трём группам, для каждой из которых предстоит пошить рубашки подходящих размеров. Это задача кластеризации. В этом случае предстоит разделить все данные на 3 кластера (но, как правило, такого строгого и единственно возможного деления нет). Увидеть расположение данных кластеров на графике позволит диаграмма рассеяния**.** Если взять другую ситуацию, когда каждый из объектов в выборке обладает сотней различных признаков, то основной трудностью будет графическое отображение такой выборки. Поэтому количество признаков уменьшают до двух или трёх, и становится возможным визуализировать их на плоскости или в 3D. Это – задача уменьшения размерности.

1. **Машинное обучение с подкреплением**

Обучение с подкреплением - это один из подходов в машинном обучении, который основывается на взаимодействии агента с окружающей средой. Агент принимает решения, выполняет действия и получает обратную связь в виде награды или штрафа. Цель агента - максимизировать суммарную награду за определенный период времени. Этот метод широко используется в играх, робототехнике и других областях, где требуется принятие решений в реальном времени. Например, в играх агент может быть игровым персонажем, который учится побеждать противников, а в робототехнике - роботом, который учится выполнять задачи, такие как уборка или сборка объектов. Агент учится на основе взаимодействия с окружающей средой, что делает этот метод особенно полезным для задач, где требуется адаптация и принятие решений в реальном времени. Агент: Субъект, который принимает решения и выполняет действия. Агент может быть программой, роботом или любым другим субъектом, который взаимодействует с окружающей средой. Основные концепции данного подхода:

* Среда (Environment): Контекст, в котором действует агент. Среда может быть физической (например, реальный мир) или виртуальной (например, компьютерная игра).
* Состояние (State): Текущее положение агента в среде. Состояние описывает все, что агент знает о среде в данный момент времени.
* Действие (Action): Выбор агента, который изменяет состояние. Действия могут быть дискретными (например, шаги влево или вправо) или непрерывными (например, изменение скорости).
* Награда (Reward): Обратная связь от среды, которая может быть положительной или отрицательной. Награда указывает агенту, насколько его действия были успешными.
* Политика (Policy): Стратегия, по которой агент выбирает действия. Политика может быть детерминированной (каждое состояние соответствует одному действию) или стохастической (каждое состояние соответствует вероятностному распределению действий).
* Функция ценности (Value Function): Оценка полезности состояния или действия. Функция ценности показывает, насколько выгодно для агента находиться в данном состоянии или выполнять данное действие.
* Q-значение (Q-Value): Оценка полезности конкретного действия в конкретном состоянии. Q-значение используется для выбора оптимальных действий.

# Основные алгоритмы моделей машинного обучения

Машинное обучение использует широкий спектр алгоритмов для перевода наборов данных в прогнозные модели. Какой алгоритм сработает лучше, зависит от решаемой задачи.

* + - 1. **Дерево принятия решений**

Это метод поддержки принятия решений, основанный на использовании древовидного графа: модели принятия решений, которая учитывает их потенциальные последствия (с расчётом вероятности наступления того или иного события), эффективность, ресурсозатратность. Для бизнес-процессов это дерево складывается из минимального числа вопросов, предполагающих однозначный ответ — «да» или «нет». Последовательно дав ответы на все эти вопросы, мы приходим к правильному выбору. Структурно дерево решений состоит из объектов двух типов - узлов (node) и листьев (leaf). В узлах расположены решающие правила и подмножества наблюдений, которые им удовлетворяют. В листьях содержатся классифицированные деревом наблюдения: каждый лист ассоциируется с одним из классов, и объекту, который распределяется в лист, присваивается соответствующая метка класса.

Визуально узлы и листья в дереве хорошо различимы: в узлах указываются правила, разбивающие содержащиеся в нем наблюдения, и производится дальнейшее ветвление. В листьях правил нет, они помечаются меткой класса, объекты которого попали в данный лист. Ветвление в листьях не производится, и они заканчивают собой ветвь дерева (поэтому их иногда называют терминальными узлами).

Если класс, присвоенный деревом, совпадает с целевым классом, то объект является распознанным, в противном случае - нераспознанным. Самый верхний узел дерева называется корневым (root node). В нем содержится весь обучающий или рабочий набор данных.

Дерево решений является линейным классификатором, т.е. производит разбиение объектов в многомерном пространстве плоскостями (в двумерном случае — линиями).

Методологические преимущества дерева принятия решений – в том, что оно структурирует и систематизирует проблему, а итоговое решение принимается на основе логических выводов.

* + - 1. **Наивная байесовская классификация**

Наивные байесовские классификаторы относятся к семейству простых вероятностных классификаторов и берут начало из теоремы Байеса, которая применительно к данному случаю рассматривает функции как независимые (это называется строгим, или наивным, предположением).

* + - 1. **Метод наименьших квадратов (линейная регрессия)**

Всем, кто хоть немного изучал статистику, знакомо понятие линейной регрессии. К вариантам её реализации относятся и наименьшие квадраты. Обычно с помощью линейной регрессии решают задачи по подгонке прямой, которая проходит через множество точек. Вот как это делается с помощью метода наименьших квадратов: провести прямую, измерить расстояние от неё до каждой из точек (точки и линию соединяют вертикальными отрезками), получившуюся сумму перенести наверх. В результате та кривая, в которой сумма расстояний будет наименьшей, и есть искомая (эта линия пройдёт через точки с нормально распределённым отклонением от истинного значения). Линейная функция обычно используется при подборе данных для машинного обучения, а метод наименьших квадратов – для сведения к минимуму погрешностей путем создания метрики ошибок.



где y - истинное значение целевой переменной, а Ϋ - предсказанное значение целевой переменной. Задача модели - в процессе обучения подобрать коэффициенты ω так, чтобы значение этой функции было как можно более близко к реальному целевому значению y объекта x.



* + - 1. **Логистическая регрессия**

Логистическая регрессия – это способ определения зависимости между переменными, одна из которых категориально зависима, а другие независимы. Для этого применяется логистическая функция (аккумулятивное логистическое распределение). Практическое значение логистической регрессии заключается в том, что она является мощным статистическим методом предсказания событий, который включает в себя одну или несколько независимых переменных, по которым предсказывается вероятность принадлежности данного объекта к каждому из классов.

Рассмотрим алгоритм логистической регрессии для бинарной классификации, т.е. случая, когда различных классов всего 2. В этом случае задача алгоритма логистической регрессии очень похожа на задачу алгоритма линейной регрессии, рассмотренного ранее: модель должна подобрать коэффициенты ω0, ω1,… ωn. , затем определяется значение:



, величина z помещается в сигмоидную функцию:



Сигмоидная функция используется для того, чтобы преобразовать число z из промежутка (−∞,∞) в число f(z) из промежутка (0,1). Причём, делается это так, что число f(z) можно теперь интерпретировать как вероятность. В нашем случае это будет вероятность принадлежности объекта x классу 1. Соответственно, вероятность принадлежности объекта классу 0 будет равна 1− f(z).

Алгоритм логистической регрессии минимизирует функцию потерь:



Здесь y - правильный ответ на объекте x (0 или 1), а p - вычисленная алгоритмом вероятность принадлежности объекта x классу 1.

* + - 1. **Метод опорных векторов (SVM)**

Это целый набор алгоритмов, необходимых для решения задач на классификацию и регрессионный анализ. Исходя из того что объект, находящийся в N-мерном пространстве, относится к одному из двух классов, метод опорных векторов строит гиперплоскость с мерностью (N – 1), чтобы все объекты оказались в одной из двух групп. На бумаге это можно изобразить так: есть точки двух разных видов, и их можно линейно разделить. Кроме сепарации точек, данный метод генерирует гиперплоскость таким образом, чтобы она была максимально удалена от самой близкой точки каждой группы.

* + - 1. **Метод ансамблей**

Он базируется на алгоритмах машинного обучения, генерирующих множество классификаторов и разделяющих все объекты из вновь поступающих данных на основе их усреднения или итогов голосования. Изначально метод ансамблей был частным случаем байесовского усреднения, но затем усложнился и оброс дополнительными алгоритмами:

* бустинг (boosting) – преобразует слабые модели в сильные посредством формирования ансамбля классификаторов (с математической точки зрения это является улучшающим пересечением);
* бэггинг (bagging) – собирает усложнённые классификаторы, при этом параллельно обучая базовые (улучшающее объединение);
* корректирование ошибок выходного кодирования.

Метод ансамблей – более мощный инструмент по сравнению с отдельно стоящими моделями прогнозирования, поскольку:

* он сводит к минимуму влияние случайностей, усредняя ошибки каждого базового классификатора;
* уменьшает дисперсию, поскольку несколько разных моделей, исходящих из разных гипотез, имеют больше шансов прийти к правильному результату, чем одна отдельно взятая;
* исключает выход за рамки множества: если агрегированная гипотеза оказывается вне множества базовых гипотез, то на этапе формирования комбинированной гипотезы оно расширяется при помощи того или иного способа, и гипотеза уже входит в него.
  + - 1. **Случайный лес**

В ансамбля базовые алгоритмы не коррелированы, за счёт этого уменьшается дисперсия относительно входящих в него базовых алгоритмов. Однако на практике добиться этого сложно: ведь базовые алгоритмы учили одну и ту же зависимость на пересекающихся выборках. В действительности оказывается, что строгое отсутствие взаимокорреляции не обязательно. Достаточно, чтобы алгоритмы были в некоторой степени не похожи друг на друга. На этом строится развитие идеи бэггинга для решающих деревьев - случайный лес.

* + - 1. **Градиентный бустинг**

В ходе обучения случайного леса каждый базовый алгоритм строится независимо от остальных. Бустинг, в свою очередь, воплощает идею последовательного построения линейной комбинации алгоритмов. Каждый следующий алгоритм старается уменьшить ошибку текущего ансамбля.

Бустинг, использующий деревья решений в качестве базовых алгоритмов, называется градиентным бустингом над решающими деревьями, (Gradient Boosting on Decision Trees, GBDT).

Он отлично работает на выборках с «табличными», неоднородными данными. Пример таких данных - описание человека через его возраст, пол, и так далее. Такой бустинг способен эффективно находить нелинейные зависимости в данных различной природы.

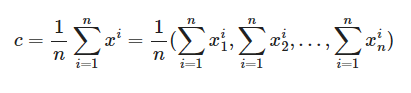
Этим свойством обладают все алгоритмы, которые используют деревья решений, однако именно GBDT обычно выигрывает в подавляющем большинстве задач. Не так хорошо бустинг проявляет себя на однородных данных: текстах, изображениях, звуке, видео. В таких задачах нейросетевые подходы почти всегда демонстрируют лучшее качество.

* + - 1. **Алгоритмы кластеризации**

Кластеризация заключается в распределении множества объектов по категориям так, чтобы в каждой категории – кластере – оказались наиболее схожие между собой элементы.  
  
Кластеризировать объекты можно по разным алгоритмам. Чаще всего используют следующие:

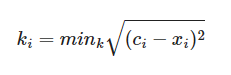
* на основе центра тяжести треугольника;
* на базе подключения;
* сокращения размерности;
* плотности (основанные на пространственной кластеризации);
* вероятностные;
* машинное обучение, в том числе нейронные сети.
  + - * 1. **K-means**

Распределяет данные на группы относя их к одному их кластеров. Центроид – центр кластера, вычисляется как среднее арифметическое векторов:

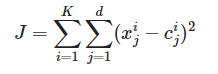
****

где n - количество векторов

Для определения кластера, необходимо найти расстояние от объекта до центроидов и выбрать тот, до которого расстояние минимально, это можно записать в виде формулы:



Во время итераций цель алгоритма - для каждого кластера минимизировать сумму расстояний от его центроида до всех элементов. Целевая функция примет вид:

****

где K - кол-во кластеров, d - все расстояния внутри кластера

* + - * 1. **Стохастическое вложение соседей с t-распределением**

Алгоритм t-SNE (t-distributed Stochastic Neighbor Embedding) позволяет понижать размерность данных до двух или трёх измерений, что позволяет визуализировать данные на двумерных и трёхмерных графиках. Изучая графики, можно, например, понять, на сколько кластеров адекватно разбивать данные, а также оценить уже выполненное разбиение на кластеры.

* + - * 1. **Сингулярное разложение**

В линейной алгебре сингулярное разложение, или SVD, определяется как разложение прямоугольной матрицы, состоящей из комплексных или вещественных чисел. Так, матрицу M размерностью [m\*n] можно разложить таким образом, что M = UΣV, где U и V будут унитарными матрицами, а Σ – диагональной. Одним из частных случаев сингулярного разложения является метод главных компонент PCA. Самые первые технологии компьютерного зрения разрабатывались на основе SVD и PCA и работали следующим образом: вначале лица (или другие паттерны, которые предстояло найти) представляли в виде суммы базисных компонент, затем уменьшали их размерность, после чего производили их сопоставление с изображениями из выборки.

Метод PCA представляет собой статистическую операцию по ортогональному преобразованию, которая имеет своей целью перевод наблюдений за переменными, которые могут быть как-то взаимосвязаны между собой, в набор главных компонент – значений, которые линейно не коррелированы. Практические задачи, в которых применяется PCA, – визуализация и большинство процедур сжатия, упрощения, минимизации данных для того, чтобы облегчить процесс обучения. Однако метод главных компонент не годится для ситуаций, когда исходные данные слабо упорядочены (то есть все компоненты метода характеризуются высокой дисперсией). Так что его применимость определяется тем, насколько хорошо изучена и описана предметная область.

Современные алгоритмы сингулярного разложения в машинном обучении, конечно, значительно сложнее, чем их предшественники, но суть их в целом нем изменилась.

* + - * 1. **Анализ независимых компонент (ICA)**

Это один из статистических методов, который выявляет скрытые факторы, оказывающие влияние на случайные величины, сигналы и пр. ICA формирует порождающую модель для баз многофакторных данных. Параметры в модели содержат некоторые скрытые переменные, причем нет никакой информации о правилах их смешивания. Эти скрытые переменные являются независимыми компонентами выборки и считаются не гауссовскими сигналами. В отличие от анализа главных компонент, который связан с данным методом, анализ независимых компонент более эффективен, особенно в тех случаях, когда классические подходы оказываются бессильны. Он обнаруживает скрытые причины явлений и благодаря этому нашёл широкое применение в самых различных областях – от астрономии и медицины до распознавания речи, автоматического тестирования и анализа динамики финансовых показателей

* + - 1. **Нейронные сети**

Искусственная нейронная сеть представляет собой [систему](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) соединённых и взаимодействующих между собой простых алгоритмов - перцептронов ([искусственных нейронов](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD)). Каждый перцептрон подобной сети имеет дело только с [сигналами](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB), которые он периодически получает, и сигналами, которые он периодически посылает другим перцептронам. И, тем не менее, будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие простые по отдельности алгоритмы вместе способны выполнять довольно сложные задачи.

Совокупность нейросетевых подходов и сама наука о нейросетях носит название глубинного обучения и является отдельной темой для исследования.

# Разделение данных

Необходимость в разделении данных на обучающую и проверочную выборки возникает вследствие проведения оценки качества обучения модели, нужно оценить ее работу на данных, которые она еще не видела, модель обучается на первой из выборок, а по второй проходит ее оценка согласно выбранной метрике.

# Основные метрики оценки качества моделей машинного обучения

Метрики оценки качества моделей разделяются для задач классификации и регрессии.

* + - 1. **Оценка качества модели в задачах классификации**

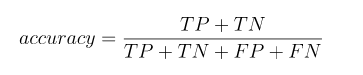
Перед переходом к самим метрикам необходимо ввести важную концепцию для описания этих метрик в терминах ошибок классификации — confusion matrix (матрица ошибок).

|  |  |  |
| --- | --- | --- |
|  | y = 1 | y = 0 |
| Ὗ = 1 | True Positive (TP) | False Positive (FP) |
| Ὗ = 0 | False Negative (FN) | True Negative (TN) |

Здесь Ὗ - это ответ алгоритма на объекте, а y - истинная метка класса на этом объекте.  
Таким образом, ошибки классификации бывают двух видов: False Negative (FN) и False Positive (FP).

#### **Accuracy**

Accuracy - это метрика, которая характеризует качество модели, агрегированное по всем классам. Доля правильных ответов модели рассчитывается по формуле:



Использование данной метрики не дает достоверную оценку качества модели, т.к при прогнозировании всех событий как TN будет получена более высокая оценка чем в иных случаях, следовательно модель не обладает ни какой предсказательной силой.

#### **Precision, recall и F-мера**

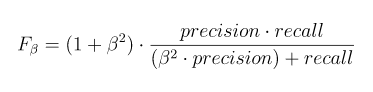
Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота).

|  |  |
| --- | --- |
|  |  |

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.

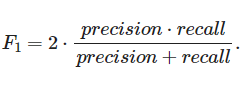
Именно введение precision не позволяет нам записывать все объекты в один класс, так как в этом случае мы получаем рост уровня False Positive. Recall демонстрирует способность алгоритма обнаруживать данный класс вообще, а precision — способность отличать этот класс от других классов. Precision и recall не зависят, в отличие от accuracy, от соотношения классов и потому применимы в условиях несбалансированных выборок.

Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. F-мера, среднее гармоническое precision и recall :



Β в данном случае определяет вес точности в метрике, при Β = 1 это среднее гармоническое (с множителем 2, чтобы в случае precision = 1 и recall = 1 иметь F = 1). F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю.

Если важность метрик Precision и Recall одинакова, можно использовать F1 – мерой:



Необходимо отметить, что в случае задач с несбалансированными классами, которые превалируют в реальной практике, часто приходится прибегать к техникам искусственной модификации набора данных для выравнивания соотношения классов, такие как oversampling (увеличение количество наблюдений с меньшим классом) или undersampling (уменьшение количества наблюдений доминирующего класса)

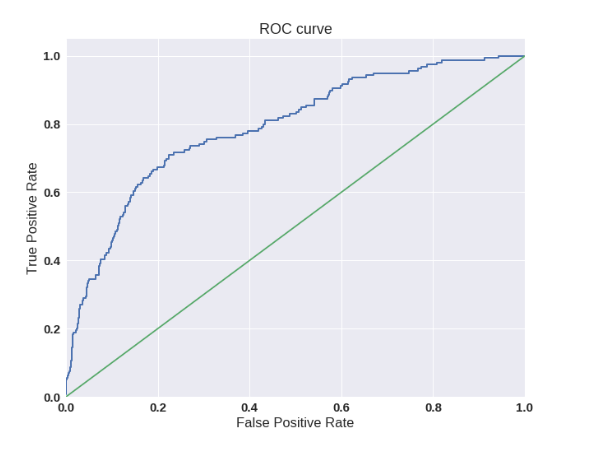
#### **AUC-ROC**

При конвертации вещественного ответа алгоритма (как правило, вероятности принадлежности к классу) в бинарное значение, 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 (*A*rea *U*nder *C*urve) под кривой ошибок (*R*eceiver *O*perating *C*haracteristic curve ). Данная кривая представляет из себя линию от (0,0) до (1,1) в координатах True Positive Rate (TPR) и False Positive Rate (FPR):

|  |  |
| --- | --- |
|  |  |

TPR это полнота, а FPR показывает, какую долю из объектов negative класса алгоритм предсказал неверно. В идеальном случае, когда классификатор не делает ошибок (FPR = 0, TPR = 1) мы получим площадь под кривой, равную единице; в противном случае, когда классификатор случайно выдает вероятности классов, AUC-ROC будет стремиться к 0.5, так как классификатор будет выдавать одинаковое количество TP и FP.



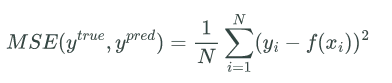
Недостатком данного метода является то, что AUC-ROC измеряет долю False Positive относительно True Negative и в задачах, где нам не так важен второй (больший) класс, может давать не совсем адекватную картину при сравнении алгоритмов.

* + - 1. **Оценка качества модели в задачах регрессии**

В задачах регрессии целевая переменная имеет потенциально бесконечное число значений. И природа этих значений, обычно, связана с каким-то процессом измерений: Задача регрессии: по обучающей выборке (xi,yi)Ni=1, где yi ∈ R построить модель f(x).

1. **MSE**

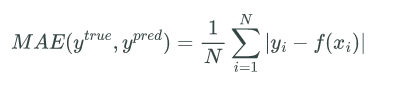
Одной из основных метрик для оценки качества моделей регрессии является *средняя квадратическая ошибка* или *mean squared error*, сокращённо *mse*. Она вычисляется как среднее от квадратов ошибок на каждом из объектов:



Иногда для того, чтобы показатель эффективности MSE имел размерность исходных данных, из него извлекают квадратный корень и получают показатель эффективности RMSE.

1. **MAE**

Также применяется *средняя абсолютная ошибка*. Она представляет из себя среднее не от квадратов ошибок, а от их модулей:

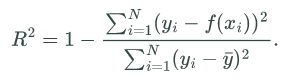


В отличается от MSE она не так сильно реагирует на выбросы в данных, например, на те ситуации, в которых у некоторых объектов недвижимости сильно завышена или сильно занижена цена.

1. **R2**

MSE неограничен сверху, и может быть нелегко понять, насколько «хорошим» или «плохим» является то или иное его значение. Чтобы появились какие-то ориентиры, делают следующее:

* Берут наилучшее константное предсказание с точки зрения MSE — среднее арифметическое целевых переменных Ὗ тестовой выборки
* Рассматривают в качестве показателя ошибки:



У идеального решающего правила **R2** равен 1, у наилучшего константного предсказания он равен 0 на обучающей выборке. Можно заметить, что **R2** показывает, какая доля дисперсии целевой переменной объяснена моделью.

# Библиотека машинного обучения

Функционал машинного обучения реализуется с использованием библиотеки scikit-learn, в которой представлены различные алгоритмы классификации, регрессии и кластеризации. Данная библиотека является одним из лучших варрантов разработки моделей машинного обучения за счет поддержки всех основных алгоритмов обучения и легкой интеграции с другими библиотеками, используемыми в процессе обучения, такими как: Pandas, Matplotlib, NumPy и других.

# Системное окружение

Docker - открытая программная платформа для быстрой разработки, тестирования, развертывания и эксплуатации приложений. Docker упаковывает программы в стандартизованные блоки, которые называются контейнерами. Каждый контейнер включает все необходимое для работы приложения: библиотеки, системные инструменты, код и среду исполнения. Контейнер Docker запускается без дополнительной нагрузки гипервизора, что позволяет добиваться большей производительности по сравнению с классической моделью виртуализации.

Windows Subsystem for Linux (WSL) - слой совместимости для запуска [Linux](https://ru.wikipedia.org/wiki/Linux)-приложений ([двоичных исполняемых файлов](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D0%BD%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB) в формате [ELF](https://ru.wikipedia.org/wiki/Executable_and_Linkable_Format)) в [ОС](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) [Windows](https://ru.wikipedia.org/wiki/Windows_10), стало возможным использовать образ [ОС](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) [Ubuntu](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9_Ubuntu) для непосредственного запуска поверх WSL множества инструментов и утилит из мира Linux без какой-либо [виртуализации](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F). WSL предоставляет интерфейсы, во многом совместимые с интерфейсами ядра [Linux](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux); однако подсистема WSL была полностью разработана корпорацией Microsoft и не содержит в себе каких-либо исходных кодов [ядра Linux](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux). WSL запускает многие не модифицированные приложения, работающие в пространстве пользователя, в частности, оболочку [bash](https://ru.wikipedia.org/wiki/Bash), утилиты sed, awk, интерпретаторы языков программирования [Ruby](https://ru.wikipedia.org/wiki/Ruby), [Python](https://ru.wikipedia.org/wiki/Python), и т. д.

Корпорация Microsoft позиционирует WSL в первую очередь как инструмент для разработчиков, веб-разработчиков и тех, кто работает над или с [приложениями с открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5).

# Архитектурный стиль

REpresentational State Transfer (REST) «передача репрезентативного состояния» -[архитектурный стиль](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) взаимодействия компонентов распределённого приложения в [сети](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C). REST - набор правил организации написания кода серверного приложения, для обеспечения простого обмена данными между системами и легкости масштабирования приложений. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой системы. В определённых случаях ([интернет-магазины](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BC%D0%B0%D0%B3%D0%B0%D0%B7%D0%B8%D0%BD), [поисковые системы](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0); прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов. REST является альтернативой [RPC](https://ru.wikipedia.org/wiki/%D0%A3%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80).

[Вызов удалённой процедуры](https://ru.wikipedia.org/wiki/%D0%A3%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80) может представлять собой обычный [HTTP](https://ru.wikipedia.org/wiki/HTTP)-запрос (обычно [GET](https://ru.wikipedia.org/wiki/HTTP) или [POST](https://ru.wikipedia.org/wiki/POST_(HTTP)); такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве [параметров](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80) запроса в формате JSON.

REST является архитектурным стилем, а не протоколом. Несмотря на то, что REST не является стандартом сам по себе, большинство REST API реализаций используют такие стандарты, как [HTTP](https://ru.wikipedia.org/wiki/HTTP), [URL](https://ru.wikipedia.org/wiki/URL), [JSON](https://ru.wikipedia.org/wiki/JSON) и, реже, [XML](https://ru.wikipedia.org/wiki/XML).

# Практическая часть

* + - 1. **EDA**

Загрузим и проанализируем обучающий набор данных:

df = pd.read\_csv('train.csv', sep=';', on\_bad\_lines='skip')

Из общего количество строк и колонок выделяем признаки:

|  |  |
| --- | --- |
| Признаки | описание |
| Person\_Id | идентификатор клиента |
| Contract\_Id | идентификатор договора клиента |
| Account\_Id | идентификатор счета клиента |
| AccountPart\_Id | идентификатор раздела счета клиента |
| Comments | назначение платежа |

, и целевые переменные

|  |  |
| --- | --- |
| целевые переменные | описание |
| ВидДокумента | вид документа – определяет производимое им действие (списание/зачисление) |
| ВидОперации | детализация действия до уровня проводимой операции |
| Контрагент | субъект отношений по документу |
| СчетУчетаРасчетовСКонтрагентом | счет субъекта |
| Субконто1…5 | аналитические признаки |
| СтатьяДвиженияДенежныхСредств | счет учета движения денежных средств |
| ЦеннаяБумага | объект, над которым совершаются действия |

*Полный список признаков приведен в приложении*

Поскольку целевых переменных много, обучаем несколько моделей для предсказания значений каждой из них.

* + - 1. **Обоснование выбора модели и метрики ее оценки**

Предсказание признаков аналитического учета является задачей классификации, в качестве baseline модели попробуем обучить логистическую регрессию и оценим результаты ее работы метрикой F1, поскольку она позволяет оценивать мульти-классовые модели, используем режим micro для вычисляется путем подсчета общего числа истинно положительных, ложноотрицательных и истинно отрицательных результатов. Это обеспечивает равный вклад каждого класса в общую метрику.

* + - 1. **Выбор инструментов и технологий разработки**

Для выполнения данной работы используется библиотека машинного обучения scikit-learn, данный выбор определяет используемый стек разработки языком Python, средой Visual Studio Code с оболочкой Jupyter Notebook, в которой будет проходить разработка и обучение модели. Работа с обученной моделью будет осуществляться с использование Web-службы по протоколу REST API.

* + - 1. **Разработка baseline модели**

Построение и обучение модели выполним в окружении Jupyter Notebook, он предоставляет удобную визуальною среду, специально созданную для работы с данными на языке Python, допускающую исполнение отдельных блоков кода и наглядную визуализацию данных. Созданная и обученная в нем модель будет сохранена в файл для последующего использования в Web-службе.

Процесс обучения модели предполагает предварительную подготовку данных, которые должны быть разделены на категориальные и количественные признаки. Категориальные признаки проходят процедуру кодирования – перевод в числовой форм. Количественные необходимо проверить на наличие «выбросов», отклонение их значения от математического ожидания более чем на 3ϭ, такие значения нужно заменить на среднее или медиану, в зависимости от «нормальности» их распределения, пропуски также необходимо обработать, для категориальных признаков можно использовать моду, значение пропущенных количественных признаков заполнит обученная для этой цели модель. После устранения «аномалий», данных проводится процедура их стандартизации, - все признаки должны принять соразмерный формат с нулевым средним и единичной дисперсией, для корректного обучения модели.

Процесс обработки данных с последующим обучением модели и оценкой его качества предпочтительно оформить в виде класса конвейера - Pipeline, это позволит выполнять одинаковый набор операций как на обучающем, так и на проверочном наборе данных. Краткое описание основных методов класса:

* prepare – выполняет формирование обучающего набора из загруженных данных, оставляя в нем только необходимы признаки и целевую переменную.
* transform – выполняет подготовку данных к передаче в модель: заполняет пропуски, обрабатывает выбросы, проводит кодирование номинальных признаков, отдельно стоит выделить проведение векторизации текстового признака «назначение платежа», в основном, именно по полученному в результате его кодирования вектору будет выполняться предсказание целевых переменных.
* sc\_data – данный метод реализован в виде декоратора Python, что дает возможность обучить и применить объект стандартизации данных внутри одного метода для обоих наборов, не передавая объект стандартизации за границы метода.
* split – делит входной набор на обучающий и проверочный
* search\_param – выполняет поиск оптимальных гиперпараметров модели по сетке
* fit – обучает модель и сохраняет ее в свойстве конвейере.
* predict – осуществляет предсказание целевой переменной
* scatterplot – позволяет графически оценить качество обучения модели
* \_resize\_ - метод понижения размерности методом PCA
* dunder методы \_\_setstate\_\_ и \_\_getstate\_\_ отвечают за сериализацию / десериализацию модели*.*

Обучение модели выполняется методом fit конвейера, для каждой целевой переменной обучается отдельную модель, сохраняемая в словарь, который затем записываем в файл.

1. # загрузка обучающего набора
2. train\_data **=** DataFrame(pd.read\_csv("train.csv", sep**=**';', on\_bad\_lines**=**'skip'))
3. pipelines **=** dict()
4. # обучения моделей для каждого целевой переменной
5. **for** target **in** ['ВидДокумента','ВидОперации','Контрагент','СчетУчетаРасчетовСКонтрагентом','СтатьяДвиженияДенежныхСредств','ЦеннаяБумага','НоменклатурнаяГруппа',\
6. 'СубконтоДт1','СубконтоДт2','СубконтоДт3','СубконтоДт4']:
7. pipeline **=** DataPipeline(train\_data**=**train\_data, features**=**['Person\_Id','Contract\_Id','Account\_Id','AccountPart\_Id','Comments'], target**=**target)
8. pipeline.transform(pipeline.train\_data)
9. pipeline.fit(LogisticRegression(), pipeline.sc\_train\_data)
10. pipeline.scatterplot(pipeline.model, pipeline.sc\_train\_data)
11. # сериализация модели
12. pipelines[target] **=** pickle.dumps(pipeline)
13. # сохранение конвейеров в файл
14. with open("pipelines.pkl", "wb") as f:
15. pickle.dump(pipelines, f)

*код класса приведен в приложении*

* + - 1. **Оценка качества модели**

Для оценки качества обучения модели применяется F-метрика, для ее расчета набор делится на обучающий и проверочный, первый использовался при обучении модели, второй при проверке качества обучения, сравнение предсказанных на нем значений с известными результатами даст реальное представление о качестве проведенного обучения.

Метод scatterplot визуализирует проверочные и предсказанные значения на плоскости, идеальным результатом обучения будет прямая восходящая из начала координат под углов в 450, в заголовке графика отображается значение рассчитанной метрики.

*Графики качества обучения моделей для каждой целевой переменной приведены в приложении.*

* + - 1. **Использование модели**

Работу с обученной моделью реализована через Web – сервис по протоколу REST API предоставляемому библиотекой flask. Создание объекта модели из предварительно загруженного словаря проходит в момент поступления запроса в методе post\_request\_all,

1. file **=** io.BytesIO(image)
2. # загрузка конвейера
3. pipeline **=** Unpickler(file).load()

При реализации данного подхода была обнаружена следующая проблема, т.к функционал был помещен в контейнер Docker, Python не может внутри него самостоятельно найти пользовательские типы данных в пространстве имен отличных от \_\_main\_\_, решением является добавление обертка над стандартным десериализатором pickle.

1. **class** Unpickler(pickle.Unpickler):
2. **def** find\_class(self, module, name):
3. **if** module **==** "\_\_main\_\_":
4. module **=** "pipeline"
6. **return** super().find\_class(module, name)

Значения признаков поступают Web-сервису в виде json данных запроса, перед использованием модели необходимо выполнить их преобразование в pandas DataFrame:

data = pd.DataFrame(pd.read\_json(request.data.decode("utf-8"), typ='series')).transpose()

, после этого можно выполнять предсказание целевой переменной

1. # подготовка признаков
2. pipeline.transform(pipeline.prepare(data))
3. # предсказание целевой переменной
4. result[target] **=** str().join(encoder.inverse\_transform(pipeline.predict(pipeline.model, pipeline.sc\_train\_data)))

*модуль использования модели приведен в приложении.*

* + - 1. **Развертывание системы**

Система должна обладать сервис-ориентированной архитектурой, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов. Хорошим вариантом для реализации данного подхода является использование Docker. В качестве хост-системы будем использовать windows 10 с развернутой службой WSL, это существенно облегчит процесс разработки и отладки, в случае необходимости в любой момент можно перенести сервис на выделенный ресурс в реальное Linux окружение.

Описание процесса развертывание системы:

* Устанавливаем службу с параметрами по умолчанию wsl –install.
* Установка Docker
  + Устанавливаем необходимые пакеты, которые позволяют apt получить доступ по HTTPS

$ sudo apt-get install apt-transport-https ca-certificates curl gnupg2 software-properties-common

* + Добавляем в систему ключ GPG официального репозитория Docker

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

* + Добавляем репозиторий Docker в список пакетов:

$ sudo add-apt-repository "deb [arch=amd64] <https://download.docker.com/linux/ubuntu> $(lsb\_release -cs) stable"

* + Обновляем данные о пакетах с учетом пакетов Docker из вновь добавленного репозитория

$ sudo apt-get update

* + Установка пакета Docker

$ sudo apt-get install docker-ce=17.09.0~ce-0~ubuntu

* + Создаем сценарий запуска службы

$ sudo nano /usr/local/sbin/start\_docker.sh

#!/usr/bin/env bash

sudo cgroups-mount

sudo service docker start

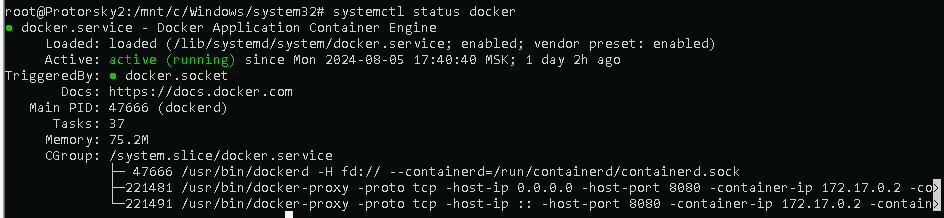
* + Права запускаемого скрипта:

$ sudo chmod +x /usr/local/sbin/start\_docker.sh

$ sudo chmod 755 /usr/local/sbin/start\_docker.sh

* запуска сервиса Docker

$ /bin/sh /usr/local/sbin/start\_docker.sh



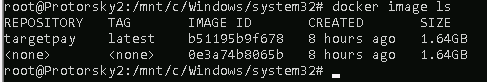
* + - 1. **Запуск системы**

Работа системы осуществляется в контейнере под управлением службы Docker. Запуск контейнера происходит из образа, который предварительно требуется собрать и поместить в Docker. Для создания образа необходимо подготовить его конфигурацию:

1. FROM python:3.12
3. RUN python **-**m pip install flask flask**-**cors chardet numpy pandas seaborn scipy scikit**-**learn matplotlib gunicorn pillow
5. WORKDIR **/**app
7. ADD pipeline.py pipeline.py
8. ADD targetpay.py targetpay.py
9. ADD pipelines.pkl pipelines.pkl
11. EXPOSE 8080
13. CMD [ "gunicorn", "--bind", "0.0.0.0:8080", "targetpay:app" ]

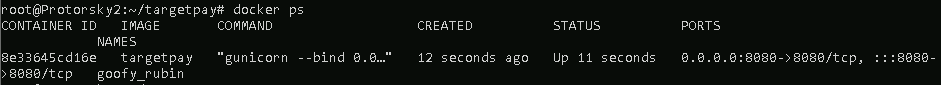
Образ собирается и помещается в рабочее пространство командой

docker build -t targetpay .



Запуск службы

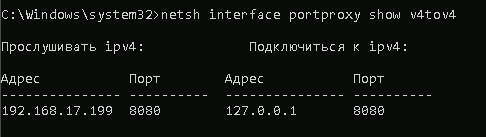
docker run -d -p 8080:8080 targetpay



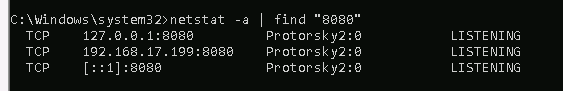
Добавляем правило firewall для трансляции сервиса наружу:

netsh interface portproxy add v4tov4 listenport=8080 listenaddress=192.168.17.199 connectport=8080 connectaddress=127.0.0.1

Данное правило перенаправляющее запрос с внешнего интерфейса на внутренний интерфейс Docker



Сервис готов к обработке запросов

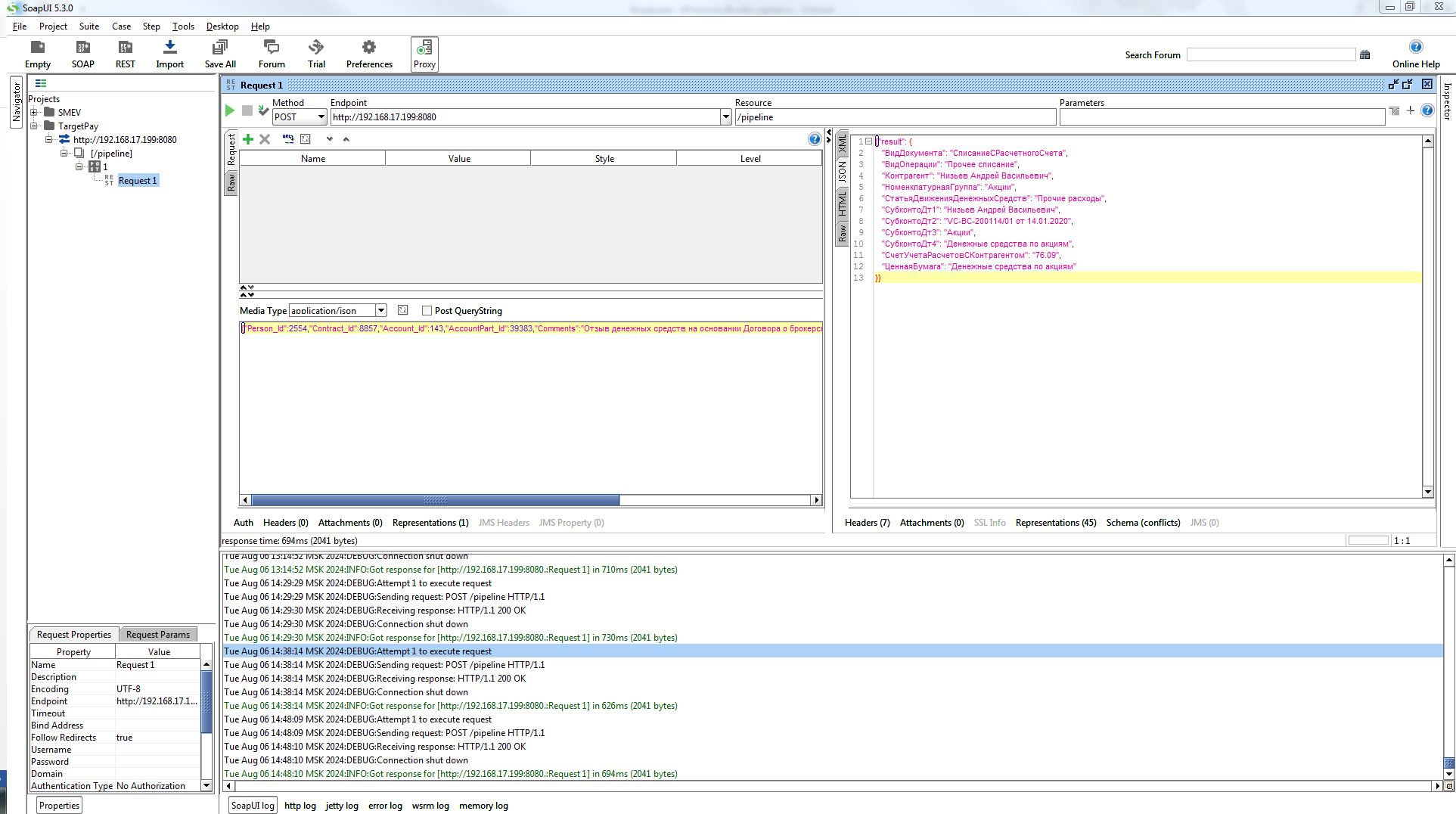


* + - 1. **Проверка работы системы**

Проверку работу службы проведём отправив HTTP запросом json данными, содержащими значения признаков, в ответ сервис должен вернуть json, содержащий значения целевых переменных

Для целей тестирования используем SoupUI, создадим в нем проект REST и сформируем запрос, содержащий json данные с признаками, укажем адрес сервиса наименование метода для работы с обученной моделью и отправим запрос на обработку. Получение ответа занимает менее одной секунды, с учетом работы системы внутри виртуального окружения на типовой рабочей станции, с предсказанием целевой переменной одновременно по одиннадцати моделям система справляется за вполне приемлемое время, такая задержка не вызовет недовольства оператора при формировании документа.

* + - * 1. **Пример 1**



Запрос

{"Person\_Id":**2554**,"Contract\_Id":**8857**,"Account\_Id":**143**,"AccountPart\_Id":**39383**,"Comments":"Отзыв денежных средств на основании Договора о брокерском обслуживании № VC-BC-200114/ 01 от 14.01.2020. НДС не облагается."}

Ответ:

{"result": {

"ВидДокумента": "СписаниеСРасчетногоСчета",

"ВидОперации": "Прочее списание",

"Контрагент": "Низьев Андрей Васильевич",

"НоменклатурнаяГруппа": "Акции",

"СтатьяДвиженияДенежныхСредств": "Прочие расходы",

"СубконтоДт1": "Низьев Андрей Васильевич",

"СубконтоДт2": "VC-BC-200114/01 от 14.01.2020",

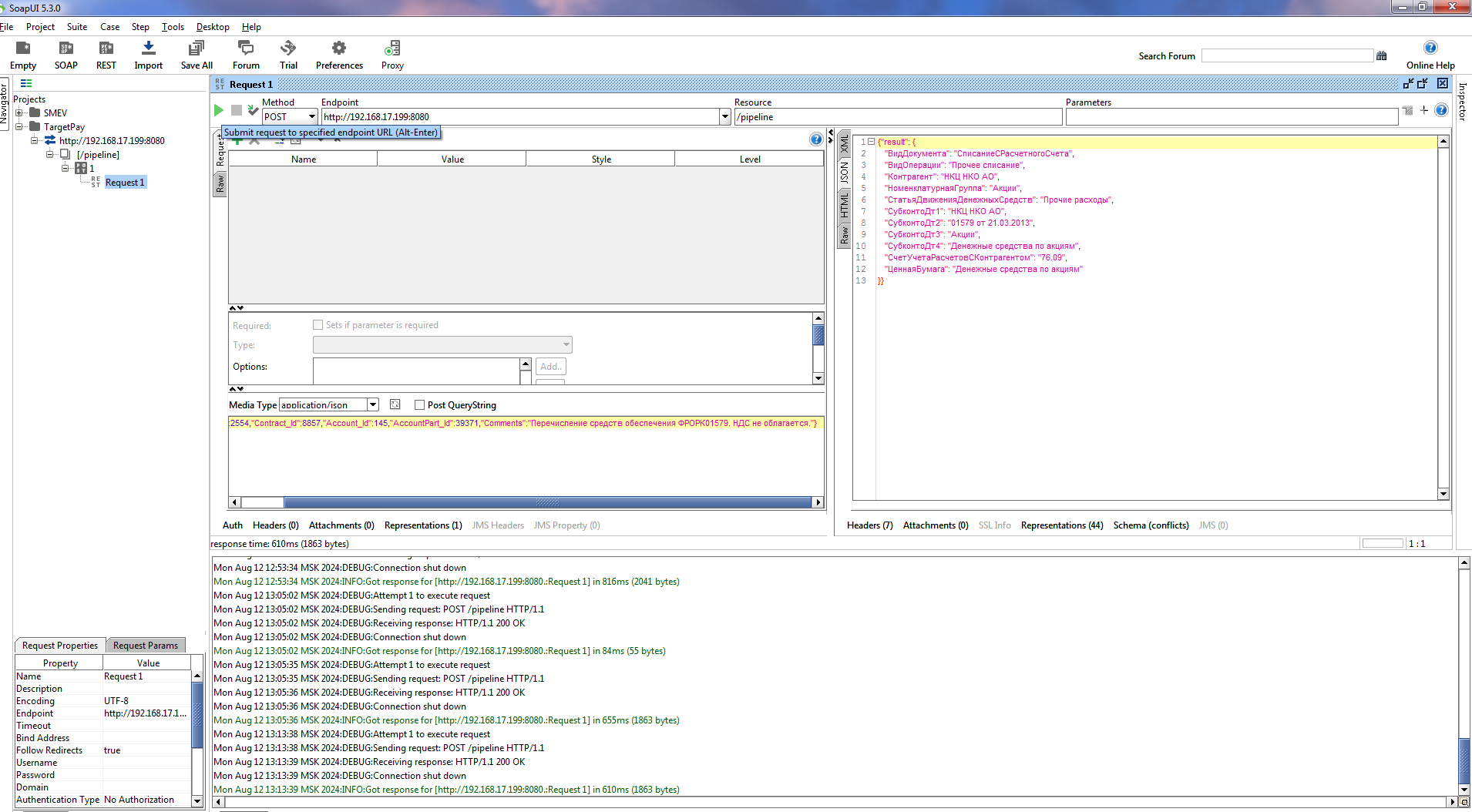
"СубконтоДт3": "Акции",

"СубконтоДт4": "Денежные средства по акциям",

"СчетУчетаРасчетовСКонтрагентом": "76.09",

"ЦеннаяБумага": "Денежные средства по акциям"}}

* + - * 1. **Пример 2**



Запрос

{"Person\_Id":**2554**,"Contract\_Id":**8857**,"Account\_Id":**145**,"AccountPart\_Id":**39371**,"Comments":"Перечисление средств обеспечения ФРОРК01579. НДС не облагается."}

Ответ:

{"result": {

"ВидДокумента": "СписаниеСРасчетногоСчета",

"ВидОперации": "Прочее списание",

"Контрагент": "НКЦ НКО АО",

"НоменклатурнаяГруппа": "Акции",

"СтатьяДвиженияДенежныхСредств": "Прочие расходы",

"СубконтоДт1": "НКЦ НКО АО",

"СубконтоДт2": "01579 от 21.03.2013",

"СубконтоДт3": "Акции",

"СубконтоДт4": "Денежные средства по акциям",

"СчетУчетаРасчетовСКонтрагентом": "76.09",

"ЦеннаяБумага": "Денежные средства по акциям"

}}

# Заключение

Система успешно прошла тестирование, возвращаемые ею данные позволяют бухгалтерской системе формировать документы с заполненными параметрами учета, что значительно уменьшает «ручной» труд оператора при их формировании.

Хотя сервис разработан и функционирует, есть существенная проблема – он способен одновременно обрабатывать только один запрос. Одним из способов решения данной проблемы, использование высокопроизводительного web – сервера Nginx в качестве балансировщика нагрузки. Можно добавить параметр запуска –workers, с указанием количества требуемых потоков, при запуске Gunicorn  в [Dockerfile](https://github.com/cdies/ML_microservice/blob/main/mnist-microservice/Dockerfile), но это привело бы к увеличению выполняемых процессов внутри контейнера, что не является верным подходом, лучшим решением будет использование [docker-compose](https://github.com/cdies/ML_microservice/blob/main/docker-compose.yml) запускающего требуемое количество копий контейнера с сервисом.

baseline модель показала хорошие результаты при тестирование на проверочной выборке, есть определенная уверенность в устойчивости работы модели в следствии невысокого изменения дисперсии входных данных, при увеличении их вариативности возможно придется прибегнуть к обучению более сложных ансамблевых моделей.

В текущей реализации системы требуется, с некоторой периодичностью, выполнять запуск Jupyter Notebook блокнота для обучения модели на обновленных данных, перезапуск Docker контейнера для обновления модели в кэше Web-сервиса, это также необходимо автоматизировать.

# Список используемой литературы

<https://gb.ru/lessons/430607>

<https://gb.ru/lessons/439262>

<https://habr.com/ru/companies/vdsina/articles/496804/>

<https://habr.com/ru/articles/548910/>

<https://learn.microsoft.com/ru-ru/windows/wsl/tutorials/wsl-containers>

<https://scikit-learn.org/stable/>

<https://www.dmosk.ru/miniinstruktions.php?mini=docker-self-image#newimage>

# Приложение

* + - 1. **Признаки набора данных**

<**class** '**pandas**.core.frame.DataFrame'>

RangeIndex: **15240** entries, **0** to **15239**

Data columns (total **47** columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

**0** DateTime **15240** non-null object

**1** Person\_Id **15240** non-null int64

**2** Contract\_Id **15240** non-null int64

**3** Account\_Id **15240** non-null int64

**4** AccountPart\_Id **15240** non-null int64

**5** Instrument\_Id **15240** non-null int64

**6** Quantity **15240** non-null object

**7** Number **15240** non-null object

**8** ExternalAccount **15107** non-null object

**9** Comments **15237** non-null object

**10** ДатаДокумента **8273** non-null object

**11** ВидДокумента **8273** non-null object

**12** ВидОперации **8273** non-null object

**13** НомерВходящегоДокумента **8273** non-null float64

**14** ДатаВходящегоДокумента **8273** non-null object

**15** СуммаДокумента **8273** non-null object

**16** ВалютаДокумента **8273** non-null object

**17** СчетБанк **8273** non-null float64

**18** СчетОрганизации **8273** non-null object

**19** СчетОрганизацииНомерСчета **8273** non-null object

**20** СчетОрганизацииБанк **8273** non-null object

**21** Контрагент **5166** non-null object

**22** КонтрагентИНН **5156** non-null float64

**23** СчетКонтрагента **3498** non-null object

**24** СчетКонтрагентаНомерСчета **3498** non-null object

**25** СчетКонтрагентаБанк **3498** non-null object

**26** СчетУчетаРасчетовСКонтрагентом **7885** non-null object

**27** СубконтоДт1 **5474** non-null object

**28** СубконтоДт2 **5323** non-null object

**29** СубконтоДт3 **4739** non-null object

**30** СубконтоДт4 **4717** non-null object

**31** СубконтоДт5 **0** non-null float64

**32** СтатьяДвиженияДенежныхСредств **8273** non-null object

**33** ЦеннаяБумага **163** non-null object

**34** НоменклатурнаяГруппа **4889** non-null object

**35** НазначениеПлатежа **8273** non-null object

**36** ДоговорКонтрагентаРасшифровкаПлатежа **410** non-null object

**37** СтатьяДвиженияДенежныхСредствРасшифровкаПлатежа **8098** non-null object

**38** СуммаПлатежаРасшифровкаПлатежа **8185** non-null object

**39** СтавкаНДСРасшифровкаПлатежа **5308** non-null object

**40** СуммаНДСРасшифровкаПлатежа **8185** non-null object

**41** СчетУчетаРасчетовСКонтрагентомРасшифровкаПлатежа **410** non-null float64

**42** СчетУчетаРасчетовПоАвансамРасшифровкаПлатежа **399** non-null float64

**43** СчетНаОплатуРасшифровкаПлатежа **36** non-null object

**44** НоменклатурнаяГруппаРасшифровкаПлатежа **4903** non-null object

**45** ЦеннаяБумагаРасшифровкаПлатежа **16** non-null object

**46** СтрокаНайдена **15240** non-null object

dtypes: float64(**6**), int64(**5**), object(**36**)

memory usage: **5.5**+ MB

* + - 1. **Графики оценки качества обучения моделей**

|  |
| --- |
| ВидДокумента |
| ВидОперации |
| Контрагент |
| СчетУчетаРасчетовСКонтрагентом |
| СтатьяДвиженияДенежныхСредств |
| ЦеннаяБумага |
| НоменклатурнаяГруппа |
| СубконтоДт1 |
| СубконтоДт2 |
| СубконтоДт3 |
| СубконтоДт4 |

* + - 1. **Класс конвейера**

1. **class** DataPipeline:
2. **def** \_\_init\_\_(self, **\*\***kwargs):
3. """Инициализация класса"""
4. # уровень стат значимости
5. self.alpha **=** 0.05
6. # срнднйи уровень корреляции
7. self.corr **=** 0.7
8. # объясненная дисперсия компонента
9. self.varatio **=** 0.8
10. # пороговое значение в три сигмы
11. self.threashold **=** 3
12. # пропорции деления набора
13. self.test\_size **=** 0.33
14. # кодировщики
15. self.\_\_encoders\_\_ **=** dict()
17. # параметры класса
18. **for** key, value **in** kwargs.items():
19. setattr(self, f"\_\_{key}\_\_", value)
21. **assert** self.\_\_target\_\_ **is** **not** None
22. **assert** self.\_\_features\_\_ **is** **not** None
23. **assert** self.\_\_train\_data\_\_ **is** **not** None
25. # подготовка обучения
26. self.\_\_train\_data\_\_  **=** self.prepare(self.\_\_train\_data\_\_)
28. **def** \_\_getstate\_\_(self) **-**> dict:
29. state **=** dict()
31. state["encoders"] **=** self.\_\_encoders\_\_
32. state["model"] **=** self.\_\_model\_\_
33. state["features"] **=** self.\_\_features\_\_
34. state["target"] **=** self.\_\_target\_\_
36. **return** state
38. **def** \_\_setstate\_\_(self, state: dict):
39. self.\_\_encoders\_\_ **=** state["encoders"]
40. self.\_\_model\_\_ **=** state["model"]
41. self.\_\_features\_\_ **=** state["features"]
42. self.\_\_target\_\_ **=** state["target"]
44. **def** prepare(self, data):
45. data **=** DataFrame(data[self.\_\_features\_\_ **+** [self.\_\_target\_\_] **if** self.\_\_target\_\_ **in** data.columns **else** []])
46. data **=** DataFrame(data[~data[self.\_\_target\_\_].isna()])
48. **return** data
50. **def** sc\_data(func: Callable):
51. """Декоратор стандартизации"""
52. **def** wrapper(self, data):
53. # стандартизация обучения
54. self.\_\_sc\_train\_data\_\_ **=** func(self, data)
55. # исключение целевой переменной из количественных признаков
56. column **=** self.\_\_sc\_train\_data\_\_.columns\_except(self.\_\_sc\_train\_data\_\_.columns\_num, self.\_\_target\_\_).columns
57. # создаем стандартизатор
58. **if** "scaler" **not** **in** self.\_\_encoders\_\_:
59. self.\_\_encoders\_\_["scaler"] **=** StandardScaler()
60. self.\_\_encoders\_\_["scaler"].fit(self.\_\_sc\_train\_data\_\_[column])
61. # стандартизация признаков
62. self.\_\_sc\_train\_data\_\_ **=** DataFrame(pd.concat([DataFrame(self.\_\_encoders\_\_["scaler"].transform(self.\_\_sc\_train\_data\_\_[column]), columns**=**column, index**=**self.\_\_sc\_train\_data\_\_.index),\
63. self.\_\_sc\_train\_data\_\_[self.\_\_sc\_train\_data\_\_.column\_cat]], axis**=**1))
64. #print("train resize")
65. # понижение размерности
66. #self.sc\_train\_data = DataFrame(self.\_resize\_(self.sc\_train\_data))
68. **return** wrapper
70. @sc\_data
71. **def** transform(self, data):
72. """Трансформация данных"""
73. data[self.\_\_target\_\_] **=** data[self.\_\_target\_\_].astype("category")
74. # заменяем inf на nan тогда моедль сможет их предсказать
75. data.replace([np.inf, **-**np.inf], np.nan, inplace **=** True)
77. # обработка количественных признаков
78. **for** item **in** data.columns\_num:
79. data **=** self.\_transform\_num\_(data, item)
81. # обработка текстовых признаков
82. **for** item **in** data.column\_cat:
83. data **=** self.\_transform\_vec\_(data, item)
85. # обработка номинальных признаков
86. **for** item **in** data.column\_cat:
87. data **=** self.\_transform\_cat\_(data, item)
89. **return** data
91. **def** \_transform\_vec\_(self, data, column):
92. """Векторизация текстовых признаков"""
93. **if** column **in** self.\_\_features\_\_:
94. data[column] **=** data[column].fillna(data[column].mode()[0])
96. **if** column **not** **in** self.\_\_encoders\_\_:
97. self.\_\_encoders\_\_[column] **=** TfidfVectorizer()
98. self.\_\_encoders\_\_[column].fit(data[column])
100. # векторизация текстового признака
101. vect **=** self.\_\_encoders\_\_[column].transform(data[column])
102. data **=** DataFrame(pd.concat([data.columns\_except(data.columns, column),\
103. DataFrame(vect.todense(), columns**=**self.\_\_encoders\_\_[column].get\_feature\_names\_out(), index**=**data.index)], axis**=**1))
105. **return** data
107. **def** \_transform\_cat\_(self, data, column):
108. """Трансформация номинальных признаков"""
109. # заменяем пропуски модой
110. data[column] **=** data[column].fillna(data[column].mode()[0])
111. # создаем кодировщик
112. **if** column **not** **in** self.\_\_encoders\_\_:
113. self.\_\_encoders\_\_[column] **=** LabelEncoder()
114. self.\_\_encoders\_\_[column].fit(data[column])
115. # кодируем признак
116. data[column] **=** self.\_\_encoders\_\_[column].transform(data[column])
117. # возвращаем тип признака из числового после кодирования
118. data[column] **=** data[column].astype("category")
120. **return** data
122. **def** \_transform\_num\_(self, data, column):
123. """Трансформация количественных признаков"""
124. **if** len(data) **==** 1:
125. **return** data
126. # пробуем предсказать пропуски
127. cdata **=** DataFrame(data.copy())
128. # запомнинание пропусков
129. cdata['isna'] **=** np.where(cdata[column].isna(), True, False)
130. # заполнение пропусков в копии, т.к моедель не работает с ними
131. **for** item **in** cdata.column\_cat:
132. **if** cdata[item].isna().values.any():
133. cdata[item] **=** cdata.fillna(cdata[item].mode())
134. **for** item **in** cdata.columns\_num:
135. **if** cdata[item].isna().values.any():
136. cdata[item] **=** cdata.fillna(cdata[item].mean())
137. # предсказание пропусков если можно сформировать обучающий и тестовый набор
138. **if** True **in** cdata['isna'].values **and** False **in** cdata['isna'].values:
139. # используем простую модель
140. model **=** LinearRegression()
141. # формируем тестовый и обучающий набор
142. X\_test **=** cdata.columns\_except(cdata.columns, column)[cdata['isna']**==**True]
143. X\_train **=** cdata.columns\_except(cdata.columns, column)[cdata['isna']**==**False]
144. y\_train **=** cdata[cdata['isna']**==**False][column]
145. # обучение модели
146. model.fit(X\_train, y\_train)
147. # выполняем предсказание пропущенного прзнака
148. data.loc[cdata['isna'], [column]] **=** model.predict(X\_test)
150. # винсоризацию уелевой переменной
151. **if** column **==** self.\_\_target\_\_:
152. data[column] **=** winsorize(data[column], limits**=**[0.1, 0.1])
154. # обработка выбрасов, вычисляем z-Score в цикле пока не подавим их все
155. **while** column !**=** self.\_\_target\_\_:
156. z **=** np.abs(stats.zscore(data[column]))
157. **if** **not** z[z > self.threashold].any():
158. **break**
159. # среднее если данные имеют нормальное распределение, иначе медиана
160. **if** stats.shapiro(data[column].values.reshape(**-**1))[1] > self.alpha:
161. data.loc[z > self.threashold, column] **=** data[column].mean()
162. **else**:
163. data.loc[z > self.threashold, column] **=** data[column].median()
165. **return** data
167. **def** \_resize\_(self, data, level **=** 0):
168. """понижение размерности"""
169. m\_corr, level **=** data.corr(), level **+** 1
170. **for** item **in** m\_corr.columns:
171. # поиск взаимокорреляции признаков по матрице
172. column **=** list(m\_corr[m\_corr[item] > self.corr].index)
173. # корреляция не с самим собой и не одни компоненты
174. **if** len(column) > 1 **and** **not** all('component' **in** item **for** item **in** column):
175. dim\_reducer **=** PCA(n\_components**=**1, random\_state**=**42)
176. components **=** dim\_reducer.fit\_transform(data[column])
177. # уровень дисперсии достаточен для свертывания
178. **if** dim\_reducer.explained\_variance\_ratio\_ >**=** self.varatio:
179. data **=** DataFrame(pd.concat([data.drop(column, axis**=**1),\
180. pd.DataFrame(data **=** components, columns **=** [f'component\_{level}'])], axis**=**1))
181. # результат свертки
182. print(f'{column} -> component\_{level} var:{dim\_reducer.explained\_variance\_ratio\_}')
183. # рекурсивный поиск компонент
184. **return** self.\_resize\_(data, level)
185. **return** data
187. @property
188. **def** train\_data(self):
189. **return** DataFrame(self.\_\_train\_data\_\_)
191. @property
192. **def** sc\_train\_data(self):
193. **return** DataFrame(self.\_\_sc\_train\_data\_\_)
195. @property
196. **def** encoders(self):
197. **return** self.\_\_encoders\_\_
199. @property
200. **def** model(self):
201. **return** self.\_\_model\_\_
203. **def** split(self, X, y):
204. """Разделение набора"""
205. **return** train\_test\_split(X, y, test\_size**=**self.test\_size, random\_state**=**42)
207. **def** search\_param(self, model, data):
208. """Поиск гиперпараметров"""
209. **if** isinstance(model, LogisticRegression):
210. params **=**{
211. 'penalty': ['none', 'elasticnet', 'l2'],
212. 'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
213. 'C': [item **/** 10 **for** item **in** range(1, 100, 10)],
214. }
216. clf **=** GridSearchCV(
217. estimator**=**LogisticRegression(random\_state**=**42),
218. param\_grid**=**params,
219. scoring**=**'f1',
220. n\_jobs**=-**1,
221. cv**=**5
222. )
223. **elif** isinstance(model, RandomForestClassifier):
224. params **=** {
225. 'criterion': ['squared\_error', 'absolute\_error', 'friedman\_mse', 'poisson', 'gini', 'entropy'],
226. 'n\_estimators': [item **for** item **in** range(10, 100, 10)],
227. 'max\_depth': [item **for** item **in** range(1, 10, 2)],
228. 'max\_features': [item **for** item **in** range(1, 10, 2)],
229. 'min\_samples\_leaf': [item **for** item **in** range(1, 10, 2)],
230. }
232. clf **=** GridSearchCV(
233. estimator**=**RandomForestClassifier(random\_state**=**42),
234. param\_grid**=**params,
235. scoring**=**'f1',
236. n\_jobs**=-**1,
237. cv**=**5
238. )
239. # формируем X и y из обучающего набора
240. X\_train, X\_valid, y\_train, y\_valid **=** self.split(data, data[self.\_\_target\_\_])
241. # искать праметры лучше на всеъ данных
242. clf.fit(pd.concat([X\_train, X\_valid]), pd.concat([y\_train, y\_valid]))
244. **return** clf.best\_params\_
246. **def** fit(self, model, data):
247. """Обучение модели"""
248. self.\_\_model\_\_ **=** model
249. # разделение набора на обучающий и проверочный
250. X\_train, X\_valid, y\_train, y\_valid **=** self.split(data.columns\_except(data.columns, self.\_\_target\_\_), data[self.\_\_target\_\_])
252. self.\_\_model\_\_.fit(X\_train, y\_train)
254. **return** self.\_\_model\_\_
256. **def** predict(self, model, data):
257. """Предсказание зависимой переменной"""
258. **return** model.predict(data.columns\_except(data.columns, self.\_\_target\_\_))
260. **def** importances(self, model, data):
261. """оценка признаков модели"""
262. **return** pd.DataFrame(zip(data.columns\_except(data.columns, self.\_\_target\_\_).columns, model.feature\_importances\_),\
263. columns **=** ['name', 'value']).sort\_values(by **=** 'value', ascending **=** False)
265. **def** scatterplot(self, model, data):
266. """график scatterplot качесива модели"""
267. X\_train, X\_valid, y\_train, y\_valid **=** self.split(data.columns\_except(data.columns, self.\_\_target\_\_), data[self.\_\_target\_\_])
269. y\_pred **=** self.predict(model, DataFrame(X\_train))
271. plt.subplot(121)
272. plt.xlabel('Predicted values')
273. plt.ylabel('True values')
274. plt.title(f"train f1:{round(f1\_score(y\_train, y\_pred, average='micro'), 3)}")
275. sns.scatterplot(x **=** y\_train, y **=** y\_pred)
277. y\_pred **=** self.predict(model, DataFrame(X\_valid))
279. plt.subplot(122)
280. plt.xlabel('Predicted values')
281. plt.ylabel('True values')
282. plt.title(f"valid f1:{round(f1\_score(y\_valid, y\_pred, average='micro'), 3)}")
283. sns.scatterplot(x **=** y\_valid, y **=** y\_pred)
285. plt.show()
287. **def** boxplot(self, data):
288. """график box-plot"""
289. data.boxplot(column **=** data.columns\_num, grid **=** False, rot **=** 45)
291. **def** corrplot(self, data):
292. """график корреляции"""
293. plt.title('Корреляционная матрица')
294. sns.heatmap(data.corr(), annot **=** True)
295. plt.show()
     * + 1. **Модуль доступа к данным модели**
296. **import** io
297. **import** pickle
298. **import** pandas as pd
300. **from** chardet **import** detect
301. **from** flask **import** Flask, jsonify, request
302. **from** flask\_cors **import** CORS
303. **from** pipeline **import** DataFrame, DataPipeline
305. app **=** Flask(\_\_name\_\_)
306. # загрузка конвейеров моделей
307. with open("pipelines.pkl", "rb") as f:
308. pipelines **=** pickle.load(f)
310. CORS(app, resources**=**{'/pipeline':{"origins":"<http://localhost:8080>"}})
312. @app.route('/pipeline', methods**=**['POST'])
313. **def** post\_request\_all():
314. **try**:
315. **assert** request.data, "no data"
316. **assert** detect(request.data)["encoding"] **==** "utf-8", "code page not utf-8"
318. result **=** dict()
319. **for** target, image **in** pipelines.items():
320. file **=** io.BytesIO(image)
321. # загрузка конвейера
322. pipeline **=** Unpickler(file).load()
323. #pipeline = pickle.loads(image)
324. # обученный кодировщик
325. encoder **=** pipeline.encoders[target]
326. # загрузка параметров запроса
327. data **=** DataFrame(pd.DataFrame(pd.read\_json(request.data.decode("utf-8"), typ**=**'series')).transpose())
328. # модель требует определение всех признаков при подготовки данных, пишем любое значение
329. data[target] **=** encoder.classes\_[0]
330. # подготовка признаков
331. pipeline.transform(pipeline.prepare(data))
332. # предсказание целевой переменной
333. result[target] **=** str().join(encoder.inverse\_transform(pipeline.predict(pipeline.model, pipeline.sc\_train\_data)))
334. # результат
335. **return** jsonify({'result': result})
336. **except** AssertionError as error:
337. **return** jsonify({'result': error.args})
338. **except** Exception as error:
339. **return** jsonify({'result': error.msg **if** hasattr(error, "msg") **else** error.args})
341. # Ћбеспечивает нахождение класса при загрузки из pickle в Docker
342. **class** Unpickler(pickle.Unpickler):
343. **def** find\_class(self, module, name):
344. **if** module **==** "\_\_main\_\_":
345. module **=** "pipeline"
347. **return** super().find\_class(module, name)
349. **if** \_\_name\_\_ **==** "\_\_main\_\_":
350. app.run(host **=** "0.0.0.0", port **=** 8080)
     * + 1. **Репозиторий проекта**

Исходный код проекта размещен в [GitHub](https://github.com/BlackStoneShadow/Python/tree/master/Diplom). (https://github.com/BlackStoneShadow/Python/tree/master/Diplom)