Работа защищена с оценкой ___

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.Ломоносова

Механико-математический факультет
Кафедра вычислительной математики

Дипломная работа
студента 610 группы
Сдобнова Евгения Павловича

Решение задачи «структура — свойство» на основе оптимизированной архитектуры нейро сети

Научный руководитель,
д.ф.-м.н. Кумсков М.И.

Москва, 2021 год.

Аннотация

В работе описаны основные подходы к решению задачи поиска зависимостей «структура-свойство» (QSAR). В качестве данных для тестирования моделей используются выборки различных химических соединений, таких как cox-2 inhibitor(нестероидные противовоспалительные препараты).

В данной работе решается задача ускорения алгоритмов машинного обучения с использованием облачных систем на основе GPU, а также задача подбора параметров внутреннего слоя нейронной сети на основе эволюционных алгоритмов. В расчетох используется облачная технология google colab.

Для написания вычислительных алгоритмов и нейронных сетей использовась библиотека pytorch.

При проведении численных экспериментов использовалась GPU NVIDIA Tesla t4 16gb.

Эволюционный алгоритм подбора параметров внутреннего слоя написан на pytorch с использованием архитектуры cuda

Все программы и данные к ним можно найти по ссылке https://github.com/SdobnovE/diplom

Содержание

1 Введение **5**

2 Глава 1 **7**
2.1 Постановка задачи структура-свойство 7
2.2 Этапы решения QSAR задачи 8
2.2.1 Формирование МД-матрицы 8
2.2.2 Основные модели, используемые при решение задачи структура-свойство 9
2.2.3 Использование кластерной структуры данных при решении задачи структура-свойство 10
2.2.4 Снижение размерности данных в задаче структура-свойство 10
2.3 Описание данных, используемых в работе 10
2.4 Описание средств, используемых в работе 11
2.5 Выводы главы 1 11

3 Глава 2 **12**
3.1 Основные методы понижения размерности данных 12
3.2 Метод группового учёта аргументов(МГУА) 12
3.3 Метод главных компонент(PCA) 15
3.4 TSNE 15
3.5 Методы снижения размерности на основе используемых моделей . . 16
3.5.1 Линейная регрессия с $l1$-регуляризацией 16
3.5.2 Случайные леса 17
3.6 Выводы из главы 2 18

4 Глава 3 **19**

5 Архитектура нейронной сети на основе МГУА **19**
5.1 Почему идея МГУА работает 19
5.2 Нейронная сеть на основе МГУА 22
5.3 Подбор оптимальных параметров нейронной сети 23
5.4 Алгоритм градиентного спуска 23
5.5 Стохастический градиентный спуск 24
5.6 Алгоритм стохастического градиентного спуска 24
5.7 Выводы из главы 3 25

6 Глава 4 **26**
6.1 Google colab 26
6.2 Cuda 26

6.3 Pytorch 27
6.4 Pytorch в применении к методу группового учета аргументов 28
6.5 Выводы из главы 4 29

7 Глава 5 **30**
7.1 Описание выборки 30
7.2 Сравнение реализаций МГУА на pytorch при запуске на cpu и gpu(cuda) 30
7.3 Сравнение классических методов 31
7.4 Беггинг на основе МГУА 32
7.5 Нейросеть на основе МГУА 32
7.6 Анализ таблицы точности для нейронной сети 33
7.7 Выводы из главы 5 35
7.8 Fish BCF Выборка 36
7.8.1 Сравнение реализаций МГУА на pytorch при запуске на cpu и gpu(cuda) 36
7.8.2 Сравнение классических методов 37
7.8.3 Нейронная сеть на основе МГУА 37

8 Заключение **38**

9 Список литературы **39**

1 Введение

Поиск зависимостей между структурами химических соединений и их свойствами — популярное направление для использования методов математической статистики и методов машинного обучения. Это направление получило название $QSAR$[12] (Quantitative Structure Activity Relationships). QSAR-модели нужны для анализа молекулярных баз данных и поиска в них потенциально активных соединений. Применение методов QSAR при создании новых соединений с заданными свойствами позволяет значительно сократить затраты на их поиск и осуществлять более целенаправленный синтез только соединений, потенциально обладающих заданным набором свойств.

Целями работы являются:

1) Исследование реализаций различных методов решения QSAR задачи с использованием машинного обучения и облачных технологий.

2) Сравнение результатов, полученных при помощи различных компьютерных архитектур(в том числе параллельных).

3) Поиск наиболее значимых групп признаков с помощью эволюционных алгоритмов в выборках с небольшим числом элементов, но большим числом признаков.

4) Построение глубокой нейронной сети на основе подобранных групп признаков эволюционного алгоритма.

В первой главе описаны основные подходы к векторизации признаков по молекулярному графу[21], а также возможные алгоритмы и подходы к решению задачи струкутра-свойство[22]. Помимо этого описаны основные средства, используемые в работе. И более подробно описаны данные, используемые в работе.

Вторая глава посвящена методам снижения размерности данных, для создания эффективных предиктивных моделей. Снижение размерности необходимо по причине ограниченности числа соединений в выборке и ограниченности вычислительных ресурсов. При этом, ввиду особенности дескрипторов, их число экспоненциально растет. На каждом этапе необходимо отбирать лучшие дескрипторы, а на их основе составлять более сложные.

В третьей главе описана архитектура нейронной сети, построенной на основе метода группового учёта аргументов. Также объяснено почему метод группового учета аргументов теоретически работает. Было описано как обучать нашу нейронную сеть на основе МГУА

Глава 4 посвящена современным инструментам для организации вычислений на удаленных серверах. Т.к. эти инструменты использовались при вычислительных экспериментах

Глава 5 полностью посвящена вычислительным экспериментам. С различными вариантами селекции признаков. И различными вариантами дескрипто-

ров(несколько вариантов длин цепочек и активных маркеров). А также произведено сравнение реализаций метода группового учета аргументов с остальными алгоритмами.

2 Глава 1

2.1 Постановка задачи структура-свойство

С развитием компьютерной техники появилась возможность использовать вычислительную технику для решения различных химических задач.

Появился термин хемоинформатика[4][5]. Хемоинформатика — это часть теоретической химии, базирующаяся на своей собственной молекулярной модели; в отличие от квантовой химии, в которой молекулы представлены как ансамбли электронов и ядер, и основанного на силовых полях молекулярного моделирования, имеющего дело с классическими «атомами» и «связями», хемоинформатика рассматривает молекулы как объекты в химическом пространстве. Согласно определению, данному А.Варнеком и И.Баскиным

Хемоинформатика находится на пересечении химии и информатики. В основе хемоинформатики лежит представление о химическом пространстве — совокупности всех доступных химических объектов (химических соединений, реакций, смесей, растворов, каталитических систем, материалов и др.). Отличительной особенностью хемоинформатики является то, что в её рамках прогнозирование свойств химических объектов осуществляется путём переноса (интерполяции) известных значений свойств от сходных химических объектов. В большинстве случаев химические объекты представимы в виде молекулярных графов, и поэтому методы теории графов находят широкое применение в хемоинформатике. Традиционный подход к обработке химической информации, однако, состоит в отображении химического пространства на дескрипторное пространство, образуемое вычисляемыми для каждого химического объекта векторами молекулярных дескрипторов[6] — числовых характеристик(инвариантных относительно перенумеровки вершин графа), описывающих химические объекты (в особенности, молекулярные графы). Это дает возможность применять методы математической статистики и машинного обучения (в том числе, интеллектуального анализа данных) для работы с химическими объектами.

За моделями, позволяющими прогнозировать количественных характеристики биологической активности, исторически закрепилось англоязычное название Quantitative Structure-Activity Relationship (QSAR). Аббревиатура QSAR часто трактуется расширенно для обозначения любых моделей структура-свойство.

Задача структура-свойство существенно распадается на две части: формирование МД-матрицы каким-либо способом и методы машинного обучения и анализа данных для улучшения качества выборки и уточнения результатов.

2.2 Этапы решения QSAR задачи

2.2.1 Формирование МД-матрицы

Пусть мы имеет N различных соединений(эталонная выборка). И количественную/классовую характеристику для каждого из них. Каждое соединение из этих N штук представляется помеченным графом $G = \{E, V\}$. Вершины этого графа интерпретируются как атомы, а рёбра как связи между ними. Рёбрам и вершинам также присваиваются различные характеристики. Например, характеристиками ребра могут являться: порядок связи, расстояние между атомами, соединёнными этим ребром и т.д. В качестве характеристик вершин можно выбрать: координату атома, заряд ядра, атомный вес или заряд и другие. Также пусть мы имеем какую-то количественную или классовую характеристику для каждого из этих соединений(например, точка кипения или точка плавления данного соединения, в случае решения регрессионной задачи, или «активное», «слабоактивное», «неактивное», в случае решения задачи классификации).

На основе этих данных необходимо построить функцию, которая получает в качестве аргумента новое соединение, а в качестве результата выдаёт либо класс этого соединения(при решении задачи классификации), либо численное значение свойства, которое мы исследуем(в случае решения задачи регрессии).

Таких функций можно построить бесчисленное множество, но при этом мы хотим, чтобы значения этой функции были как можно ближе к реальным значениям. Для этого вводятся различные метрики качества для данной функции. Например, в случае задачи классификации метрикой качества может служить отношения числа правильных ответов к неправильным, а также точность и полнота. В случае задачи регрессии метрикой качества может служить функция средней или среднеквадратичной ошибки.

Кажому молекулярному соединению мы будем ставить в соответствие вектор **дескрипторов**[3]. Где каждый дескриптор является инвариантом молекулярного графа.

Вектором признаков или вектором дескрипторов молекулярного графа G будем называть вектор $(x_1, \ldots, x_M)$, где x_j - значение j-ого дескриптора, вычисленное для того или иного соединения.

В итоге для выборки из N соединений мы получаем матрицу $N \times M$. Где по строкам расположены соединения. А в каждом столбце стоит тот или иной дескриптор. Таким образом $A_{i,j}$ элемент матрицы — это значение j-го признака для i-го соединения. В соответствии каждой строке признаков мы имеем ту или иную числовую характеристику/класс, для предсказание которой/которого мы хотим построить модель.

Как правило в матрице используются конкретные дескрипторы. Берётся вся выборка молекул по ней строится алфавит k-цепочек. **k-цепочка** - это набор из

k атомов, соединённых последовательно в молекулярном графе. Также к таким цепочкам можно добавлять и другие характеристики(включать различные маркеры), расширяя алфавит признаков. После составления такого алфавита, для каждой молекулы и для каждого элемента алфавита вычисляется сколько раз он содержится в молекуле. Полученная матрица используется в анализе данных. Такая матрица получается очень разреженной, т.к. размер алфавита получается намного больше числа цепочек в соединении

2.2.2 Основные модели, используемые при решение задачи структура-свойство

Линейная регрессия

Линейная регрессия (англ. Linear regression) — используемая в статистике регрессионная модель зависимости одной (объясняемой, зависимой) переменной y от другой или нескольких других переменных (факторов, регрессоров, независимых переменных) x с линейной функцией зависимости.

Модель линейной регрессии является часто используемой и наиболее изученной и многие вещи в ней объяснены с точки зрения математической статистики. Также могут использоваться различные вариации линейной регрессии с различными методами регуляризации(l2-регуляризация, l1-регуляризация одни из самых популярных вариантов). Регуляризация позволяет справится с линейной зависимостью столбцов матрицы и переобучением.

МГУА(Метод группового учёта аргументов)

Этот метод основан на линейной регрессии и позволяет отобрать несколько групп наиболее репрезентативных дескрипторов. Соответственно, существенно уменьшить размерность пространства дескрипторов. Заключается он в следующем:

На вход подается несколько параметров(число селекций, минимальный порог корелляции, размер буфера). На первом шаге отбираются лучшие пары векторов(строится линейная регрессия от всех возможных пар). Определние "лучших"основано на величине R^2 ошибки($R^2 = 1 - \frac{\|\varepsilon\|}{\|y - y_{mean}\|}$, чем ошибка ближе к 1, тем лучше, $\varepsilon = y_{\text{pred}} - y$). Соответственно, параметр размер буфера, говорит сколько максимум пар мы можем взять в буфер на каждой селекции. На последующих шагах мы также строим линейные двух переменных, но в качестве факторов используем вектора ошибкок с прошлого шага и вектор из МД-матрицы. Число селекций - это число таких шагов. Т.к. каждый раз в буфер мы брали вектор из МД-матрицы и из прошлого шага, то можно получить цепочку векторов. Векторы из таких цепочек как раз и будут наилучшим образом описывать выборку.

2.2.3 Использование кластерной структуры данных при решении задачи структура-свойство

Гипотеза компактности - гипотеза, согласно которой близким в содержательном смысле объектам в геометрическом пространстве признаков соответствуют обособленные множества точек, обладающие свойствами хорошей отделимости.

Согласно этой гипотезе близкие объекты похожи(имеют одинаковый класс или имеют близкие целевые значения). Отсюда следует довольно естественное предложение, строить прогноз не на всей выборке объектов, а строить прогноз на основе кластерной структуры. Т.е. для начала определяется, к какому кластеру мог бы быть отнесён объект. А уже в каждом кластере есть своя модель, которая совершает прогноз. Также можно отказываться от прогноза для тех соединений, которые попадают в определённый кластер.

Некоторые алгоритмы кластерного анализа такие как плотностные и агломеративные не только могут искать кластеры, но также позволяют определять объекты, являющиеся выбросами. Т.е. с помощью них мы можем удалить шумовые данные и не обучать наши модели на них.

2.2.4 Снижение размерности данных в задаче структура-свойство

В начале нужно определить какого размера цепочки мы будем использовать для построения матрицы молекула-дескриптор(МД-матрицы). А также определить какие маркеры нам необходимо использовать для каждой из вершин и связей в цепочках. Варьируя эти параметры мы получаем большое количество матриц. Причем эти матрицы очень широкие. Т.к. эти матрицы очень широкие, то нас настигает проклятье размерности(измерение расстояния между векторами высокой размерности не эффективно). Для устраения такой проблемы используются различные методы понижения размерности. Некоторые из этих методов такие как МГУА просто выбирают наилучшие признаки. Другие же методы такие как PCA и другие, изменяют признаки до неузнаваемости. И порой такие методы снижения размерности не нужно использовать, если в задаче есть огромные риски потерь и необходима четкая обоснованность тех или иных признаков в моделях.

2.3 Описание данных, используемых в работе

В этой работе используются данные, взятые из сборника научных работ Модели и архитектуры нейронных сетей в задаче «структура-свойство». - 2020, - 46 с, издательство ООО "МАКС Пресс"(Москва), с. 3-11

Набор соединений cox-2 inhibitor (нестероидные противовоспалительные препараты).

Из данных заранее были сформированы МД-матрицы. Порядка 20 штук. При составлении использовались разные алфавиты, с разными размерами цепочек и различными маркерами атомов и их связей.

2.4 Описание средств, используемых в работе

Представлены программы, реализующие некоторые из описанных методов. Программы написанны на языке программирования Python, в среде Anaconda. В них используются различные библиотеки Python. Такие как pandas, numpy, scipy, pytorch и другие.

Также для ускорения счёта программ, использовались облачные вычисления на платформе Google colab.

Для обеспечения эффективного ознакомления с работой, коды программ и данные представлены по ссылке https://github.com/SdobnovE/diplom.

2.5 Выводы главы 1

В главе была поставлена задача структура-свойство. Были описаны основные области изучения в хемоинформатике, а также были описаны способы обработки данных, которые, как мы предполагаем, должны улучшить результаты. Также основные способы решения задачи QSAR. Даны базовые представления о некоторых алгоритмах, описанных в последующих главах.

3 Глава 2

3.1 Основные методы понижения размерности данных

Предположим, что на основе молекулярного графа было выделено большое количество различных признаков. Вероятнее всего, большая часть этих признаков для предсказания того или иного свойства будет просто бесполезна, а иногда может даже и мешать. Для этого используются различные методы снижения размерности такие как: метод группового учёта аргументов, метод главных компонент и другие. Эти методы можно условно разделить на несколько категорий:

1) Методы отбора лучших признаков из существующих. Например МГУА. Также можно избавиться от признаков, для которых вектор признака не сильно коррелирует с целевым вектором

2) Метод замены признаков с помощью линейного преобразования и отбор лучших. Таким образом работает алгоритм главных компонент(PCA)

3) Метод замены признаков с помощью нелинейного преобразования. Например t-sne.

4) Метод выделения главных признаков на основе модели. К таким можно отнести модель линейной регрессии с $l1$-регуляризацией, и случайный лес. В линейной регрессии с такой регуляризацией признаки, которые не существенны, просто будут иметь нулевой или близкий к нулю вес. В случайном лесе будут выбраны наиболее распространённые признаки, которые участвовали в расщеплении вершин.

Далее подробно рассмотрим некоторые из них.

3.2 Метод группового учёта аргументов(МГУА)

Предположим, что мы уже выбрали неплохую модель для наших данных, теперь мы просто хотим уменьшить размерность вектора дескрипторов. Пусть выбранная модель — это обычная линейная регрессия, или гребневая регрессия(Ridge Regression, линейная регрессия с $l2$-регуляризацей). На начальном этапе нам необходимо выбрать размер буфера Q, число селекций S и минимальный порог корреляции MinCorr.

1-ая селекция:

Начинаем строить векторы $\hat{y}_k^1 = b_0 + b_1 x_i + b_2 x_j$, пробегая по всем столбцам матрицы. И кладём эти векторы $\hat{y}_k^1$ в первый буфер, упорядочивая их в порядке возрастания ошибок R^2. Где $R^2 = 1 - \frac{\|\varepsilon\|}{\|y - y_{mean}\|}$ При этом, при вставке вектора в буфер смотрим попарные корреляции с каждым вектором из буфера. Если хотя бы один из коэффициентов корреляции больше порогового коэффициента, то вставку в буфер мы не осуществляем. Также мы запоминаем индесы i и j, для которых были построены $\hat{y}$, чтобы в конце получить репрезентативные цепочки

дескрипторов.

L-ая селекция:

До этого шага мы построили $L-1$ буфер с векторами $\hat{y}$. Теперь начинаем строить векторы $\hat{y}_k^L = b_0 + b_1\hat{y}_i^{L-1} + b_2 x_j$. Также кладём эти векторы в L-ый буфер, упорядочивая векторы в порядке возрастания ошибок R^2.

В итоге получим множество буферов. Их число равно числу селекций. Т.к. каждый раз векторы следующего буфера мы строили на основе векторов предыдущего, то можем просто развернуть цепочку, выражая на каждом шаге $\hat{y}$ через $\hat{y}$ из предыдущего буфера и вектора из матрицы признаков:

$$\hat{y}_k^L = b_0 + b_1\hat{y}_i^{L-1} + b_2 x_j,$$
$$\hat{y}_i^{L-1} = a_0 + a_1\hat{y}_{i_1}^{L-2} + b_2 x_{j_1}$$

Тогда:

$$\hat{y}_k^L = b_0 + b_1(a_0 + a_1\hat{y}_{i_1}^{L-2} + b_2 x_{j_1}) + b_2 x_j$$

$$\hat{y}_k^L = b_0 + b_1 a_0 + b_1 a_1\hat{y}_{i_1}^{L-2} + b_1 b_2 x_{j_1} + b_2 x_j$$

Таким образом можем продолжать и дальше, пока $\hat{y}$ не станет просто x_t. Тогда получим некоторое наборы индексов векторов. Это векторы, которые дают наибольшую зависимость, для предсказания. Соответственно после всех процедур мы можем проверить точность только на отобранных признаках и на всех, и сравнить результаты.

Далее рассмотрим псевдокод МГУА

Algorithm 1 МГУА

for k = 0...S **do** ▷ Цикл по всем селекциям
  IterBuf = 0
  for i = 0...$BufSize_{k-1}$ **do** ▷ Цикл по всем векторам в предыдущем буфере
    for j = 0...M **do** ▷ Цикл по всем векторам в X
      X2 = (X_j, $Buf_{k-1,i}$) ▷ j-ый вектор из матрицы X,i-ый из предыдущего буфера
      model.fit(X2, y)
      if IterBuf < SizeBuf **then** ▷ Если не заполнили буфер ещё
        $\hat{y}$ = model.predict(X2)
        CurrCorr = max(CorrelationCoefficient($\hat{y}$, $Buf_{k,l}$)), l =0...IterBuf
        if IterBuf == 0 or CurrCorr < MinCorr **then** ▷ Корреляция меньше порога
          Buf.insert($\hat{y}$)
          IterBuf++
        end if
      else ▷ Если буфер уже заполнен
        $\hat{y}$ = model.predict(X2)
        ind = argmin($BufR^2$) ▷ Будем вставлять в место, где R^2 = min
        CurrCorr = max(CorrelationCoefficient($\hat{y}$, $Buf_{k,l}$)), l =0...IterBuf
        if $BufR^2[ind]$ < $R^2Score(\hat{y}, y)$ and CurrCorr < MinCorr **then**
          Buf.insert($\hat{y}$, position=ind) ▷ Вставляем на место ind
          $BufR^2[ind] = R^2Score(\hat{y}, y)$
        end if
      end if
    end for
  end for
  $BufSize_k = IterBuf$

3.3 Метод главных компонент(PCA)

Задача: Хотим построить новые признаки B на основе нашей МД-матрицы A, причём эти признаки должны получаться из основных с помощью линейного преобразования U.

Тогда формально постановку задачи можно записать как:

$$\|BU^T - A\| \to min$$

Получили оптимизационную задачу. Где в качестве параметров используется матрица новых признаков B и матрица перехода U. Причем, A размерности $l \times n$, B размерности $l \times m$, U размерности $n \times m$. Где m может быть много меньше n.

Основная теорема метода главных компонент:

Если $m \leq rkA$, то минимум задачи $\|BU^T - A\|$ достигается, когда столбцы матрицы U это собственные векторы матрицы F^TF, соответствующие m наибольшим собственным значениям. А матрица $B = A \times U$.

В результате матрица U ортонормирована. А матрица B ортогональна. И мы посроили новые признаки, через которые с помощью линейных комбинаций выражаются старые с небольшой ошибкой, а эту ошибку мы минимизировали. Этот метод основан на том, что мы максимизируем выборочную дисперсию данных вдоль главных компонент

3.4 TSNE

Стохастическое вложение соседей с t-распределением (англ. t-distributed Stochastic Neighbor Embedding, t-SNE) — это алгоритм машинного обучения для визуализации, разработанный Лоренсом ван дер Маатеном и Джеффри Хинтоном[4]. Он является техникой нелинейного снижения размерности, хорошо подходящей для вложения данных высокой размерности для визуализации в пространство низкой размерности (например двумерное). Метод подбирает точки в двумерном (или трехмерном) пространстве так, чтобы близкие точки в n-мерном пространстве оказались также близкими и в двумерном(трехмерном) пространстве.

Пусть имеем N объектов в n-мерном пространстве $x_1, x_2, \ldots, x_N$. Хотим построить новые объекты $y_1, y_2, \ldots, y_N$ в двумерном пространстве, причём близкие точки должны быть близко, а далёкие, соответственно, далеко друг от друга

Проделаем два шага:

1) Составим матрицу вероятностей похожести объектов i и j из $x_1, x_2, \ldots, x_N$,

следующим образом

$$p_{i|j} = \frac{exp(\frac{-\|x_i - x_j\|^2}{2\sigma_i^2})}{\sum_{k \neq i} exp(\frac{-\|x_i - x_k\|^2}{2\sigma_i^2})}$$

Ван дер Маатен и Хинтон объясняли: «Похожесть точки данных x_j точке x_i является условной вероятностью $p_{j|i}$, что для x_i будет выбрана x_j в качестве соседней точки, если соседи выбираются пропорционально их гауссовой плотности вероятности с центром в x_i»

Соответственно для $y_1, y_2, \ldots, y_N$ нужно проделать нечто похожее, но из-за того, что пространство низкой размерности надо использовать немного другие условные вероятности

$$q_{i|j} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i}(1 + \|y_i - y_k\|^2)^{-1}}$$

2) Теперь нужно сделать так, чтобы второе распределение вероятностей стало как можно похоже на первое (тогда близкие точки перейдут в близкие, а далёкие в далёкие). В качестве меры похожести двух распределений используется дивергенция Кульбака-Лейблера(несимметричная мера удалённости друг от друга двух вероятностных распределений).

$$\sum_{i=1}^{N} p_i log(\frac{p_i}{q_i})$$

Решая оптимизационную задачу на коэффициенты $y_1, y_2, \ldots, y_N$, получим, что мы отобразили пространство размерности N в пространство размерности 2. Причем с "сохранением"относительных расстояний.

3.5 Методы снижения размерности на основе используемых моделей

3.5.1 Линейная регрессия с $l1$-регуляризацией

Для понижения размерности мы можем воспользоваться простейшей линейной моделью. Для настройки параметров Линейной регрессии минимизируется следующий функционал:

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} (\langle \vec{x_i}, \vec{\omega} \rangle - y_i)^2 \to min$$

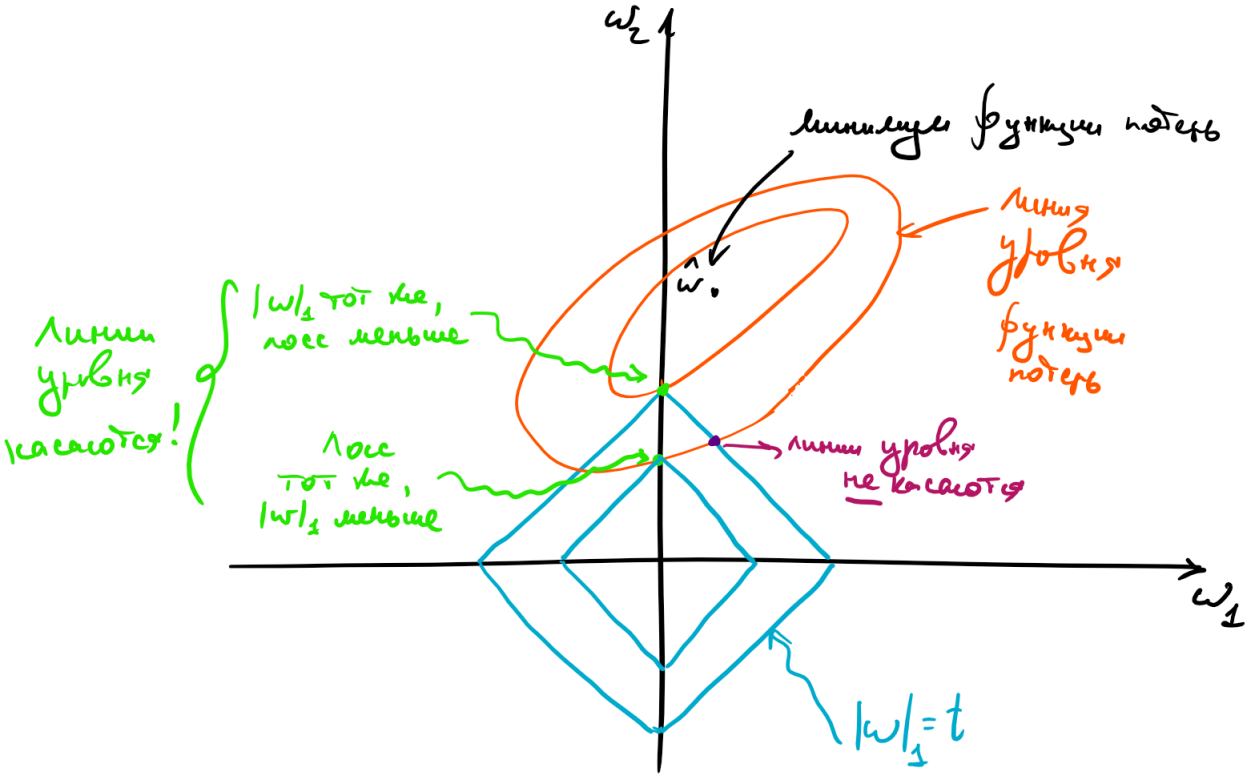
где x_i - строчка в МД-матрице X, а y_i - целевое значение.

В случае Lasso регрессии(регрессии с $l1$ регуляризацией), будем иметь немного другой вид оптимизируемого функционала:

$$L(w, \lambda) = (\frac{1}{N} \sum_{i=1}^{N} (\langle \vec{x_i}, \vec{\omega} \rangle - y_i)^2) + \lambda \|\vec{\omega}\| \to min$$

где λ подбирается на кросс валидации или на скользящем контроле.

Можно попробовать визуализировать и понять, почему менее значимые признаки, действительно будут близки к нулю.



Пусть в матрице всего два признака X_1 и X_2. Чёрной точкой обозначен оптимум нашего функционала. Синие кривые - линии уровня оптимума. тогда, подбираясь к определенной линии оптимума видно, что значение Lasso функционала будет меньше при $X_2 = 0$ в точке A. Данная картинка объясняет, почему оптимум Lasso функционала достигается в точках обнуления некоторых из весов. На основе этой модели, делается вывод, что признаки с около-нулевым весом можно не включать в модель.

3.5.2 Случайные леса

Случайный лес - модель машинного обучения, которая решает задачу классификации или регрессии. Строится на основе большого количества слабообученных деревьев решений. Дерево решений строится по следующим принципам:

1) Выбирается столбец в матрице выбирается порог. Обьекты со значением меньше порога идут в левую вершину. Иначе в правую. Этот столбец и порог выбираются так, чтобы максимизировать критерий информативности.

2) Первый шаг делается для каждого предка до тех пор, пока дерево не слишком глубокое, или все вершины не принадлежат одному классу(в случае классификации), или если дисперсия меньше порога(в случае регрессии) или из других соображений.

Деревья склонны к переобучению. Поэтому используется очень много слабообученных деревьев и делается средневзвешенный прогноз. В случайном лесе каждый раз столбец выбирается не из всей матрицы, а лишь из какого-то подмножества столбцов, случайно заданного для каждого дерева.

После обучения случайного леса, все признаки можно отсортировать по тому, сколько раз они участвовали в расщеплении вершины. Из них отбираются те признаки, которые встречались чаще других. Таким образом совершается отбор наиболее значимых признаков.

3.6 Выводы из главы 2

В данной главе были подробно разобраны основные алгоритмы снижения размерности данных. Приведена реализация метода группового учёта аргументов на псевдокоде.

4 Глава 3

5 Архитектура нейронной сети на основе МГУА

Как мы обсуждали ранее, МГУА выделяет множество наиболее качественных цепочек.

Каждая цепочка - это отдельные наборы признаков, причем все цепочки сильно отличаются друг от друга.

Т.к. цепочки не связаны(векторы предсказания скоррелированы меньше порога), можем попробовать построить ансамбль алгоритмов на полученных признаках.

Т.о. постараемся добиться хорошего качества, используя ансамбль не очень сильных моделей, которые практически независимы.

5.1 Почему идея МГУА работает

Рассмотрим задачу регрессии с квадратичной функцией потерь. Представим также для простоты, что целевая переменная y — одномерная и выражается через переменную x как:

$$y = f(x) + \varepsilon$$

где f — некоторая детерминированная функция, а ε — случайный шум со следующими свойствами:

$$E(\varepsilon) = 0, \mathrm{Var}(\varepsilon) = \sigma^2$$

В зависимости от природы данных, которые описывает эта зависимость, её представление в виде точной $f(x)$ и случайной ε может быть продиктовано тем, что:

1) данные на самом деле имеют случайный характер;

2) измерительный прибор не может зафиксировать целевую переменную абсолютно точно;

3) имеющихся признаков недостаточно, чтобы исчерпывающим образом описать объект, пользователя или событие.

Функция потерь на одном объекте x равна

$$\mathrm{MSE} = (y(x) - a(x))^2$$

Однако знание значения MSE только на одном объекте не может дать нам общего понимания того, насколько хорошо работает наш алгоритм. Какие факторы

мы бы хотели учесть при оценке качества алгоритма? Например, то, что выход алгоритма на объекте x зависит не только от самого этого объекта, но и от выборки X, на которой алгоритм обучался:

$$X = ((x_1, y_1), \ldots, (x_l, y_l))$$

$$a(x) = a(x, X)$$

Кроме того, значение y на объекте x зависит не только от x, но и от реализации шума в этой точке:

$$y(x) = y(x, \varepsilon)$$

Наконец, измерять качество мы бы хотели на тестовых объектах x — тех, которые не встречались в обучающей выборке, а тестовых объектов у нас в большинстве случаев более одного. При включении всех вышеперечисленных источников случайности в рассмотрение логичной оценкой качества алгоритма a кажется следующая величина:

$$Q(a) = E_x E_{X,\varepsilon}[y(x, \varepsilon) - a(x, X)]^2$$

Внутреннее матожидание позволяет оценить качество работы алгоритма в одной тестовой точке x в зависимости от всевозможных реализаций X и ε, а внешнее матожидание усредняет это качество по всем тестовым точкам.

Попробуем представить выражение для Q(a) в более удобном для анализа виде. Начнём с внутреннего матожидания:

$$E_{X,\varepsilon}[y(x, \varepsilon) - a(x, X)]^2 = E_{X,\varepsilon}[f(x) + \varepsilon - a(x, X)]^2 =$$

$$= E_{X,\varepsilon}[(f(x) - a(x, X))^2 + 2\varepsilon(f(x) - a(x, X)) + \varepsilon^2] =$$

$$= E_X[(f(x) - a(x, X))^2] + 2E_\varepsilon[\varepsilon] \cdot E_X(f(x) - a(x, X)) + E_\varepsilon[\varepsilon^2] =$$

$$= E_X(f(x) - a(x, X))^2 + \sigma^2$$

Из общего выражения для $Q(a)$ выделилась шумовая компонента σ^2. Продолжим преобразования:

$$E_X(f(x) - a(x, X))^2 = E_X[(f(x) - E_X[a(x, X)] + E_X[a(x, X)] - a(x, X))^2] =$$

$$= E_X[(f(x) - E_X[a(x, X)])^2] + E_X[(a(x, X) - E_X[a(x, X)])^2] +$$

$$+2E_X[(f(x) - E_X[a(x,X)]) \cdot (E_X[a(x,X)] - a(x,X))] =$$

$$= \text{bias}_X^2 a(x,X) + \text{Var}[a(x,X)]$$

Получили разложение для $Q(a)$ в виде смещения и дисперсии(разброса)
Если представить алгоритм $a(x)$ как:

$$a(x) = \frac{1}{k}(b_1(x) + \cdots + b_k(x))$$

При этом алгоритмы b_i были обучены на разных подмножествах признаков, а следовательно алгоритмы являются независимыми, то перепишем наше выражение для $Q(a)$

$$\text{bias}_X a(x,X) = f(x) - E_X[a(x,X)] = f(x) - E_X[\frac{1}{k}\sum_i b_i(x)]$$

$$= f(x) - \frac{1}{k}\sum_i E_X[b_i(x)] = f(x) - E_X[b(x)] = \text{bias}_X b(x)$$

Получили, что смещение композиции равно смещению одного алгоритма. Теперь посмотрим, что происходит с разбросом.

$$\text{Var}_X[a(x,X)] = E_X[a(x,X) - E_X[a(x,X)]]^2$$

$$= E_X[\frac{1}{k}\sum_i b_i(x) - E_X[\frac{1}{k}\sum_i b_i(x)]]^2 =$$

$$= \frac{1}{k^2} E_X[\sum_i (b_i(x) - E_X[b_i(x))]]^2 =$$

$$= \frac{1}{k^2}\sum_i \text{Var}_X b_i(x) + \frac{1}{k^2}\sum_{k_1 \neq k_2} \text{cov}(b_{k_1}(x), b_{k_2}(x))$$

Если предполагаем, что b_i некоррелированы, то:

$$\text{Var}_X[a(x,X)] = \frac{1}{k^2}\sum_i \text{Var}_X b_i(x) = \frac{1}{k^2}\sum_i \text{Var}_X b(x) = \frac{1}{k}\text{Var}_X b(x)$$

Получилось, что в этом случае дисперсия композиции в k раз меньше дисперсии отдельного алгоритма.

Теперь, используя множество цепочек и одинаковые регрессионные алгоритмы на них, получаем множество нескоррелированных алгоритмов, для которых совместный прогноз уменьшает дисперсию в k раз, где k - размер буфера

5.2 Нейронная сеть на основе МГУА

Зачем нужны сложности с МГУА, если мы можем просто построить нейронную сеть? Да, мы так и правда можем сделать, но т.к. выборка имеет небольшое число объектов, то полноценную глубокую нейронную сеть мы обучить не сможем.

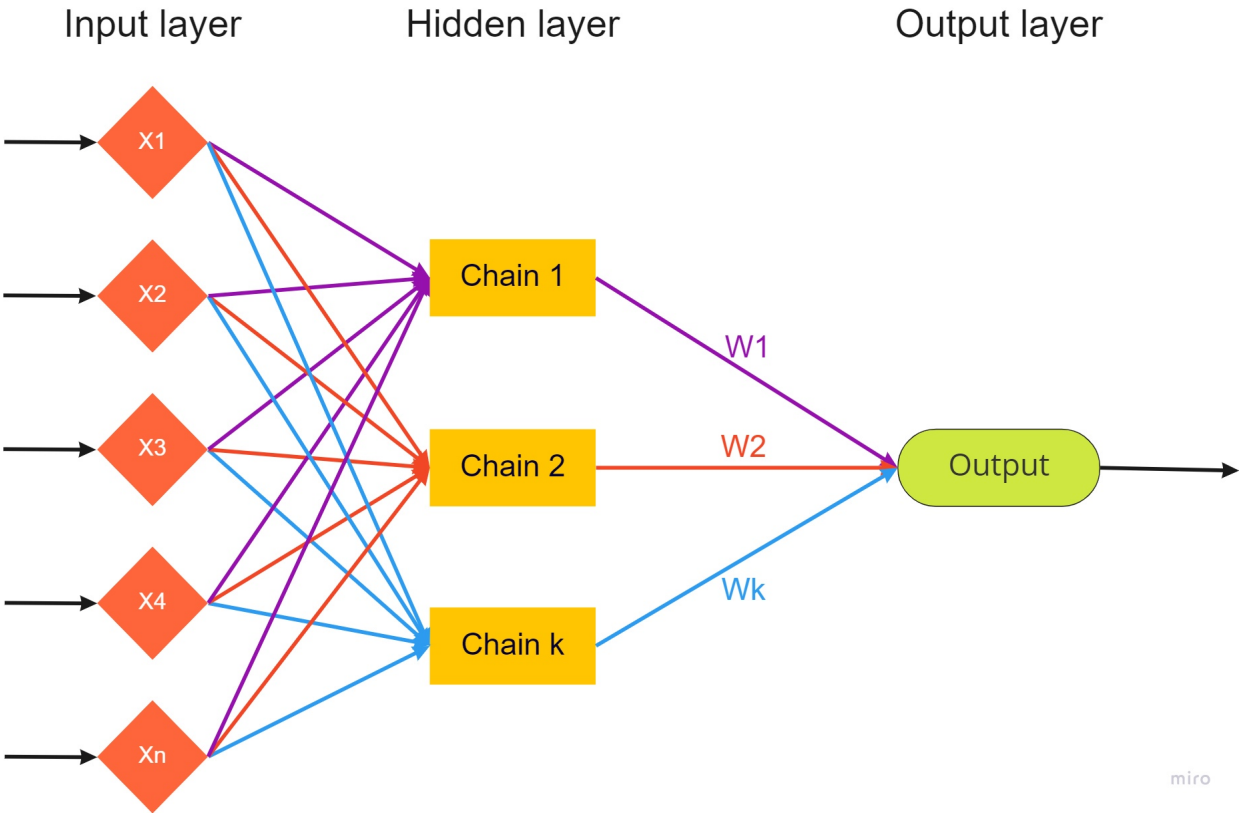
Воспользуемся регрессиями уже построенными на основе цепочек МГУА. Т.к. у нас заведомо есть веса регрессий на небольшом числе признаков, то первый слой сети уже готов.

Также в пункте выше мы уже обсудили ансамбль цепочек МГУА, который является просто средневзвешенным прогнозом.

Попробуем использовать вместо средневзвешенного прогноза, полносвязный линейный слой нейронной сети.

Число входов этого слоя будет равно числу цепочек МГУА, а выход это просто число, т.е. наш прогноз.

Ниже приведем схему:



Таким образом блоки скрытого слоя уже являются обученными и не требуют обучения в нашей сети, т.к. веса регрессии и векторы в цепочках уже определены.

В сети необходимо обучить только веса выходного слоя.

Далее подробнее обсудим как обучать сеть.

5.3 Подбор оптимальных параметров нейронной сети

Как известно, градиент функции в точке направлен в сторону её наискорейшего роста, а антиградиент (противоположный градиенту вектор) в сторону наискорейшего убывания. То есть, имея какое-то приближение параметра $\vec{\omega}$, мы можем его улучшить, посчитав градиент функции потерь в точке и немного сдвинув вектор весов в направлении антиградиента:

$$\omega_j \mapsto \omega_j - \alpha \frac{d}{d\omega_j} L(\omega, X, y)$$

где α – это параметр алгоритма ("темп обучения"), который контролирует величину шага в направлении антиградиента. Описанный алгоритм называется градиентным спуском.

Т.о. мы берём нашу нейронную сеть, определяем функционал потерь и начинаем производить оптимизацию градиентным спуском, каждый раз шагая в направление наискорейшего убывания нашей функции

Градиент нашего функционала потерь по j-й координате вектора весов равен:

$$\frac{dL}{d\omega_j} = \frac{d}{d\omega_j}(\frac{1}{n}\sum_{i=1}^{n}(y_i - \langle x_i, \omega \rangle)^2) =$$

$$= \frac{2}{n}\sum_{i=1}^{n}(y_i - \langle x_i, \omega \rangle)\frac{d}{d\omega_j}(\sum_{k=0}^{n} x_{ik}\omega_k - y_i) = \frac{2}{n}\sum_{i=1}^{n}(\langle x_i, \omega \rangle - y_i)x_{ij}$$

В матричном виде вычисление вектора градиента может быть записано так:

$$\frac{dL}{d\omega} = \frac{2}{n}X^T(X\omega - y)$$

Следовательно, имея данный вектор весов, мы знаем, как его сделать немного лучше. Но из какой точки стартовать процесс? Так как функционал выпуклый, это не так важно, и подойдет любое значение.

5.4 Алгоритм градиентного спуска

```
w = (0, ... , 0)                          ▷ Можно попробовать иную инициализацию весов
Repeat S times{                           ▷ Или abs(err) > tolerance
  f = X.dot(w)                            ▷ Считаем предсказание
  err = f - y                             ▷ Считаем ошибку
  grad = 2 * X.T.dot(err) / n             ▷ Считаем градиент
  w -= alpha * grad                       ▷ Обновляем веса
}
```

Вычислительная сложность градиентного спуска – $O(ndS)$, где, как и выше, n – длина выборки, d – размерность одного обьекта.

Сложность по памяти – $O(nd)$. В памяти мы держим и выборку, и градиент, но доминирует, разумеется, выборка.

5.5 Стохастический градиентный спуск

На каждом шаге градиентного спуска нам требуется выполнить потенциально дорогую операцию вычисления градиента по всей выборке (сложность $O(nd)$, плюс ещё память на хранение градиента). Алгоритм можно существенно ускорить, заменив градиент его оценкой на подвыборке (в английской литературе такую подвыборку обычно именуют batch, в русской разговорной терминологии тоже часто встречается слово "батч").

Как делить выборку на батчи? Ясно, что можно было бы случайным образом сэмплировать их из полного датасета, но даже если использовать быстрый алгоритм вроде резервуарного сэмплирования, сложность этой операции не самая оптимальная. Поэтому используют линейный проход по выборке (которую перед этим лучше всё-таки случайным образом перемешать), при котором на B очередных примерах [еще один новый параметр алгоритма = размер батча B] вычисляется градиент и производится обновление весов модели. При этом вместо количества шагов алгоритма обычно используют количество эпох E [да-да еще один новый параметр], где одна эпоха – это один полный проход по выборке. Заметим, что если выборка очень большая, а модель компактная, то может быть достаточно и одного неполного прохода.

5.6 Алгоритм стохастического градиентного спуска

Algorithm 2 Gradient descent

w = (0, ... , 0) ▷ Можно попробовать и что-то другое

Repeat E times:

  for (i = B, i <= n, i+=B)

    $X_{batch} = X[i - B : i]$

    $y_{batch} = y[i - B : i]$

    f = $X_{batch}.dot(w)$ ▷ Считаем предсказание

    err = f - y ▷ Считаем ошибку

    grad = 2 * $X_{batch}.T.dot(err)/n$ ▷ Считаем градиент

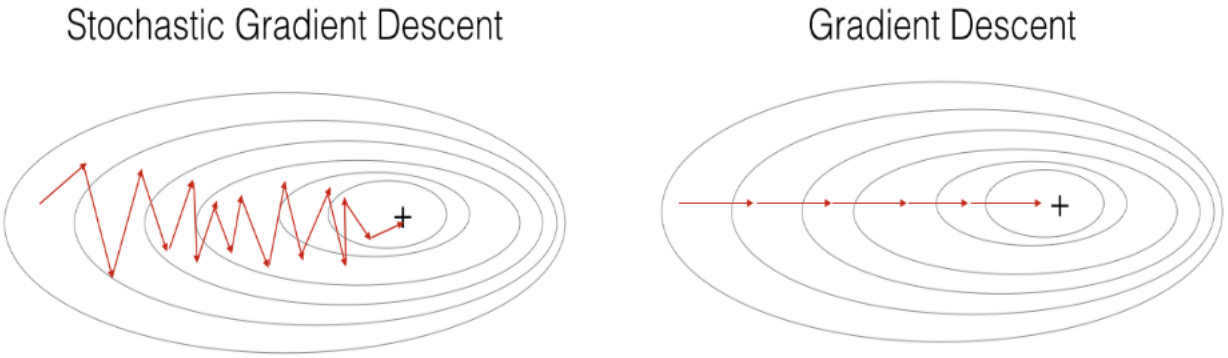
    w -= alpha * grad ▷ Обновляем веса

Сложность по времени – $O(ndE)$. На первый взгляд, она такая же, как и у обычного градиентного спуска, но заметим, что мы сделали в n/B раз больше

шагов, то есть веса модели претерпели намного больше обновлений.

Сложность по памяти можно довести до $O(Bd)$: ведь теперь всю выборку не надо держать в памяти, а достаточно загружать лишь текущий батч (а остальная выборка может лежать на диске, что удобно, так как в реальности задачи, в которых выборка целиком не влезает в оперативную память, встречаются сплошь и рядом). Заметим, что при этом лучше бы B взять побольше: ведь чтение с диска – намного более затратная по времени операция, чем чтение из оперативной памяти.

В целом, разницу между алгоритмами можно представлять как-то так:



Шаги стохастического градиентного спуска заметно более шумные, но их значительно быстрее считать. И в итоге они тоже сходятся к оптимальному значению [в случае выпуклого функционала качества] из-за того, что матожидание оценки градиента на батче равно самому градиенту. Для сложных моделей и лоссов стохастический градиентный спуск может сходиться плохо или застревать в локальных минимумах, поэтому придумано множество его улучшений.

5.7 Выводы из главы 3

В главе были показаны и доказаны идеи работы метода группового учета аргументов. Были расписаны доказательства разложения ошибки в виде смещения и дисперсии.

Построена нейронная сеть на основе МГУА.

Также было описано как подбирать веса для нашей нейронной сети с помощью градиентных методов оптимизации. Было объяснено, почему в задаче линейной регрессии градиентные методы эффективны и оптимум на саммом деле достигается.

Была доказана единственность точки минимума функционала наименьших квадратов. А также рассмотрен стохастический градиентный спуск

6 Глава 4

6.1 Google colab

Google Colab — это облачный сервис на основе Jupyter Notebook. Google Colab дает стандартный доступ к Jupyter Notebook прямо в браузере. Также, предоставляется доступ к достаточно мощным GPU(в нашем случае это NVIDIA Tesla t4 16gb).

Все данные и jupyter блокноты хранятся на google disk

С помощью Google Colab можно без проблем обучить модель за считанные секунды. Он поддерживает Python3 из коробки и множество библиотек машинного обучения.

Большой объем данных для обучения можно также хранить на удаленных дисках. В отличие от локальной машины, где объем дискового пространства может быть сильно ограничен.

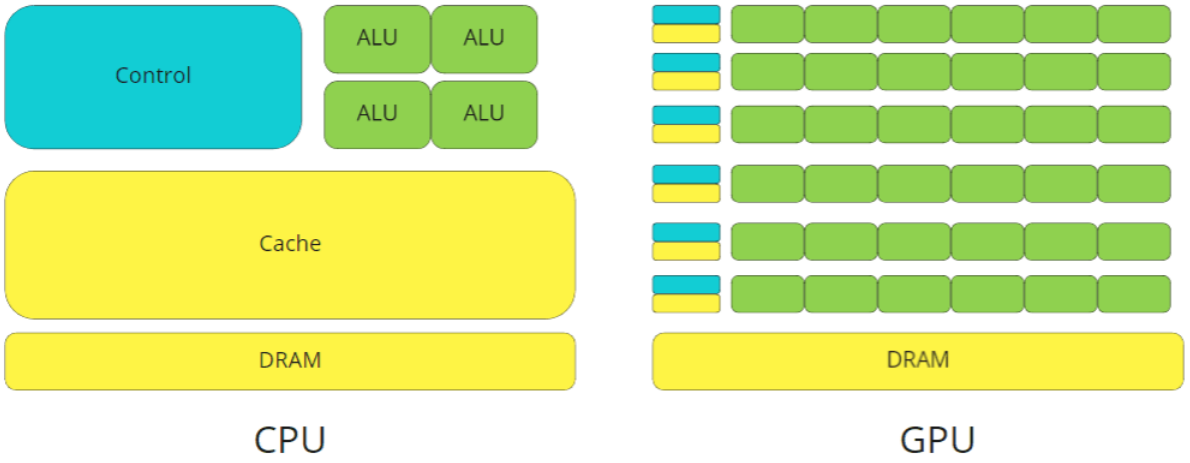
В частности, в данной работе **google colab** был использован как основная платформа для проведения численных экспериментов, ввиду того, что платформа предоставляет доступ к достаточно мощным **gpu** с поддержкой **cuda**

6.2 Cuda

CUDA (англ. Compute Unified Device Architecture) — это технология на базе программно-аппаратной архитектуры, которая позволяет повысить производительность параллельных вычислений.

Параллельные вычисления — это вычисления, при которых процесс разработки программного обеспечения делится на потоки. Потоки обрабатываются **параллельно** и взаимодействуют между собой в процессе обработки. Это возможно благодаря процессорам компании NVIDIA, на которых построена работа CUDA.

GPU (англ. Graphics Processing Unit) — это специальный графический процессор, который заточен на обработку 2D или 3D-графики. Он размещается на видеокарте, что позволяет автоматически освободить основной процессор от лишней нагрузки при обработке данных. GPU состоит из нескольких тысяч ядер, которые в совокупности потребляют небольшое количество энергии. CUDA ядра «выигрывают» у CPU по производительности на 1 ватт потребляемой мощности.



На базе этих основных процессоров были разработаны различные специализированные инструменты. Например, технология GPGPU.

GPGPU (англ. General-purpose computing on graphics processing units) — это технология, которая позволяет использовать графический процессор GPU в операциях, которые обычно выполняет центральный процессор CPU. Например, в математических вычислениях. С помощью GPGPU можно использовать видеокарту для выполнения неграфических вычислений. При этом графический процессор будет работать не вместо центрального, а в качестве вычислительного блока.

CUDA является улучшенной вариацией GPGPU. Она позволяет работать на специальном диалекте — это значит, что программисты могут использовать алгоритмы, предназначенные для графических процессоров при обработке неспецифических задач.

6.3 Pytorch

PyTorch — современная библиотека глубокого обучения, позволяющая производить операции над тензорами и вычислять на них градиенты.

Данная библиотека позволяет динамически строить граф вычислений, высчитывать градиенты и совершать шаги градиентного спуска, что очень полезно в задачах машинного обучения.

В свою очередь, когда граф вычислений статический, код, описывающий различные операции с тензорами, выполняется только один раз, в результате чего формируется статический граф вычислений, например в tensorflow. Чтобы получить результаты вычисления графа, необходимо подавать данные в граф и запустить вычисления для необходимых тензоров.

При использовании динамического графа вычисления происходят сразу при определении операций.

Таким образом, динамический граф вычислений является более удобным:

1. Вычисления выполняются сразу при определении операций;

2. Результаты вычислений доступны на любом шаге;

3. Более лёгкая и гибкая разработка, упрощённый дебагинг.

Ещё одной особенностью является простое использование cuda. Как только хочется воспроизводить вычисления на GPU, тензоры переносятся в память gpu и все операции выполняются на графических процессорах.

Тензор в pytorch - многомерная матрица, по каждому элементу которой можно осуществлять дифференцирование.

PyTorch использует «магнитофонную» систему автоматических градиентов – собирает информацию о том, какие операции и в каком порядке производились над тензорами, а затем воспроизводит их в обратном направлении, чтобы выполнить дифференциацию в обратном порядке (reverse-mode differentiation). Вот почему он такой супер-гибкий и допускает произвольные вычислительные графы.

6.4 Pytorch в применении к методу группового учета аргументов

Рассмотрим L-ую селекцию метода группового учета аргументов.

Пусть размер буфера - **buf_size**, **m** - число признаков в МД матрице, **n** - число объектов в выборке.

Для каждой пары нам нужно провести линейную регрессию двух векторов($\hat{\vec{y}} = b_0 + b_1\vec{x_1} + b_2\vec{x_2}$)

Итого получаем $\mathbf{buf\_size} \cdot \mathbf{m}$ регрессий

Сконструируем трехмерный тензор размерности $(\mathbf{buf\_size} \cdot \mathbf{m}, \mathbf{n}, 3)$

Где первая координата - номер регрессии. По двум оставшимся координатам получаем матрицу, в первом столбце стоит вектор из предыдущего буфера, во втором столбце стоит вектор из из матрицы признаков, третий вектор единичный, для смещения.

Для данного тензора используем формулу точного решения линейной регрессии с **l2**-регуляризацией

$$\omega = (X^T X + \lambda I)^{-1} X^T y$$

Т.о. за несколько матричных умножений мы получаем устойчивое решение для коэффициентов регрессии.

Из полученных коэффициентов мы легко получаем значения целевого вектора и начинаем отбор лучших в новый буфер.

После подобного использования pytorch для подбора цепочек МГУА, можем использовать стандартные возможности pytorch для обучения второго слоя нейронной сети.

6.5 Выводы из главы 4

В главе был проведен обзор используемых инструментов, таких как Google Colab, cuda и библиотека pytorch.

Были приведены положительные стороны данных инструментов, а также объяснен новый подход для реализации метода группового учета аргументов с помощью них

7 Глава 5

Текущая глава будет посвящена вычислительным экспериментам

В главе произведено сравнение реализаций метода группового учета аргументов на языке python.

Запуски производились на удаленном сервисе google colab.

7.1 Описание выборки

В качестве выборки в работе используется набор соединений cox-2 inhibitor (нестероидные противовоспалительные препараты). На основе этих соединений уже были сформерованы матрицы молекула-дескриптор, состоящие из различных цепочек и активных/неактивных маркеров. Данные можно найти по ссылке https://github.com/SdobnovE/diplom.

Все цепочки были объеденены в одну матрицу.

Итого, выборка получилась размером (393, 2999)

Все объекты выборки были случайно перемешены и разделены на обучающую и валидационную выборки в пропорциях 0.8 и 0.2 соответственно.

7.2 Сравнение реализаций МГУА на pytorch при запуске на cpu и gpu(cuda)

Ниже приведем таблицу запусков со временем счета:

	Размер буфера	Число селекций	CPU время	GPU время	Отношение
0	10	10	5.241535	1.175528	4.458877
1	10	20	11.016886	2.783242	3.958292
2	10	50	28.942173	12.170582	2.378044
3	10	100	58.491596	38.680934	1.512156
4	50	10	27.268342	14.900883	1.829982
5	50	20	57.639915	34.288368	1.681034
6	50	50	152.425282	139.299783	1.094225
7	50	100	313.112178	330.158582	0.948369
8	100	10	52.629370	30.877217	1.704473
9	100	20	112.384924	79.121775	1.420404
10	100	50	302.065299	342.397114	0.882207
11	100	100	626.872687	775.017379	0.808850
12	200	10	115.126790	140.535675	0.819200
13	200	20	240.955601	303.135240	0.794878
14	200	50	625.123623	797.050120	0.784297
15	200	100	1280.646762	1727.185946	0.741464

В таблице можно заметить странные результаты.

Иногда ускорение составляет больше 4.45, но иногда ускорение на gpu добиться не удалось.

Можно сделать вывод, что при большом размере буфера и большом числе селекций параллельность на cuda начинает забиваться обычным python последовательным кодом для выбора элементов в буфер.

Отметим, что при вставке элементов в буфер, когда он почти заполнен происходят очень редко и алгоритм начинает считать корелляции со всем буфером для всех векторов, это и приводит к снижению производительности.

7.3 Сравнение классических методов

Приведем таблицу сравнения базовых методов на различных признаках.

Для сравнения выборка была разделена на тренировочную и тестовую в процентном соотношении 80% и 20%.

Для алгоритмов были отобраны наилучшие параметры, на которых и были произведены финальные замеры метрик на отложенной выборке.

Full - предсказания на всей построенной матрице молекула-дескриптор.

PCA 70 - предсказания на матрице, построенной с помощью **PCA** на 70-ти компонентах.

TSNE 3 - предсказания на матрице, построенной с помощью **TSNE** на 3-х компонентах.

SVD 70 - предсказания на матрице, построенной с помощью **SVD** на 70-ти компонентах.

	Full	PCA 70	TSNE 3	SVD 70
Линейная регрессия	-3.818e+24	-0.587	0.074	-1.038
Случайный лес	0.106	0.151	0.032	0.146
Градиентный бустинг	0.170	0.220	0.161	0.162
Ridge	0.128	-0.586	0.074	-1.036

Из таблицы видно, что векторы сильно скореллированы, поэтому линейная регрессия на всей матрице показывает очень плохие результаты и переобучается под наши данные

Также градиентный бустинг показывает лучшие результаты, в сравнении с остальными алгоритмами.

Случайный лес также показывает неплохие результаты, но всё же не приближается к бустингу по стабильности и качеству.

7.4 Беггинг на основе МГУА

Проверим как работает МГУА и какое качество удается получить.

Ниже приведем таблицу:

Bagging демонстрирует качество усредненного прогноза на основе всех цепочек

Min демонстрирует минимальное качество прогноза среди всех цепочек

Max демонстрирует максимальное качество прогноза среди всех цепочек

	Размер буфера	Число селекций	Bagging	Min	Max
0	10	10	0.376385	0.267252	0.389333
1	10	20	0.364645	0.261011	0.377885
2	10	50	0.382551	0.370472	0.376345
3	10	100	0.385297	0.371158	0.389386
4	50	10	0.330596	0.189071	0.426987
5	50	20	0.357893	0.224518	0.423371
6	50	50	0.408164	0.250357	0.433115
7	50	100	0.398392	0.252109	0.436676
8	100	10	0.317223	0.039138	0.414069
9	100	20	0.353140	0.152923	0.434590
10	100	50	0.373363	0.058108	0.425583
11	100	100	0.368967	-0.130696	0.430934
12	200	10	0.286118	-0.064715	0.385815
13	200	20	0.315364	0.016404	0.424984
14	200	50	0.292283	-0.398493	0.429430
15	200	100	0.227217	-0.356872	0.440663

Из полученной таблицы можем заметить, что качество стабильно лучше любого алгоритма из предыдущей секции.

При этом качество стабильно лучше, а максимальное качество на отдельной цепочке ещё лучше, в сравнении с усредненным прогнозом.

7.5 Нейросеть на основе МГУА

Проверим работу нейронной сети построенной на основе МГУА:

Ниже приведем таблицу:

buf_size - размер буфера МГУА

chain_len - длина цепочки МГУА, также число селекций агоритма

No activation - на первом уровне не используется никакой функции активации

relu - на первом слое используется relu активация

sigmoid - на первом слое используется sigmoid активация
tanh - на первом слое используется tanh активация

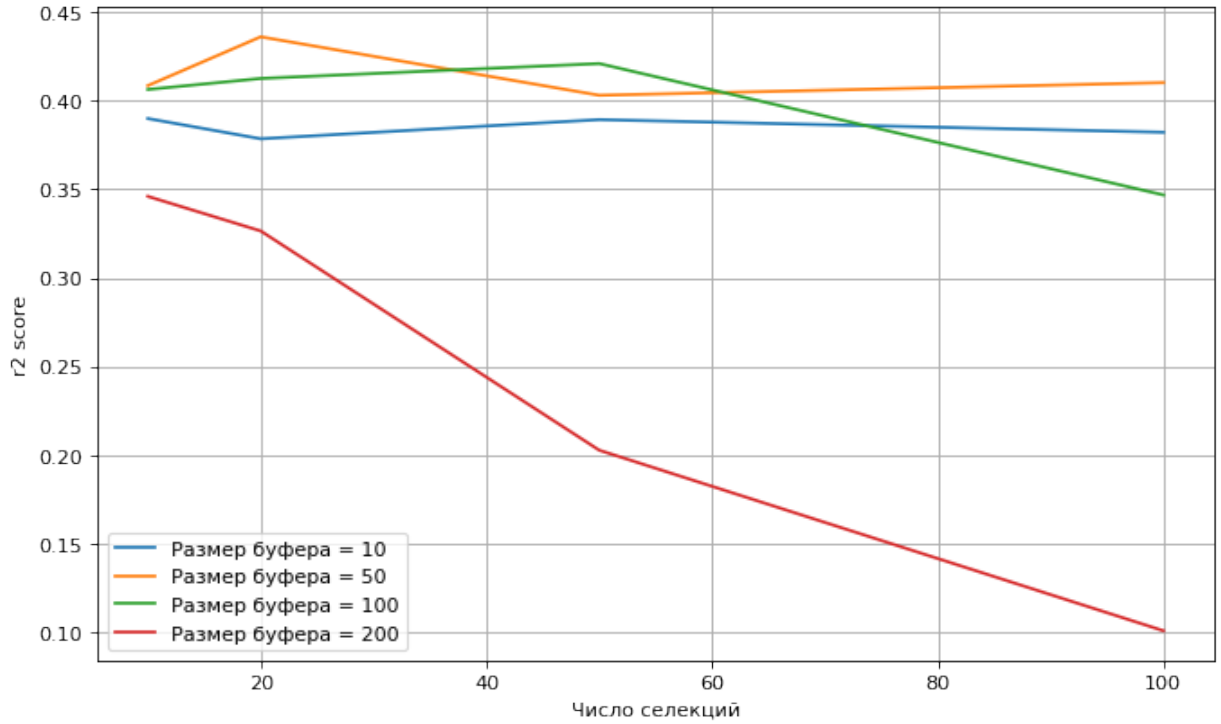
	buf_size	chain_len	No activation	relu	sigmoid	tanh
0	10	10	0.398962	0.371089	-0.000814	-1.322623e-03
1	10	20	0.373251	0.367681	-0.000156	-1.020183e-03
2	10	50	0.389006	0.388660	0.004308	7.119220e-04
3	10	100	0.378129	0.378119	0.005004	5.461376e-04
4	50	10	0.404939	0.405134	0.000266	3.851579e-08
5	50	20	0.438102	0.431509	0.000432	8.159716e-07
6	50	50	0.421315	0.380407	0.023768	4.006186e-02
7	50	100	0.409074	0.430254	0.037618	5.033838e-02
8	100	10	0.412537	0.400919	0.009010	2.577787e-02
9	100	20	0.420142	0.426915	0.027922	4.249161e-02
10	100	50	0.428085	0.411283	0.041614	5.372787e-02
11	100	100	0.359890	0.342787	0.054470	6.725224e-02
12	200	10	0.347622	0.364954	0.022994	5.623024e-02
13	200	20	0.368885	0.287894	0.053649	8.182365e-02
14	200	50	0.256745	0.137979	0.074037	1.037140e-01
15	200	100	0.225715	0.214920	0.078818	1.027157e-01

Из данной таблицы видно, что нейронная сеть на основе МГУА показала более хорошие результаты, чем беггинг над цепочками МГУА.

Анализ самой таблички рассмотрим далее.

7.6 Анализ таблицы точности для нейронной сети

Построим зависимости качества от числа селекций, а также качества от размера буфера, чтобы сделать вывод об оптимальном размере буфера и числе селекций для построенной нейронной сети

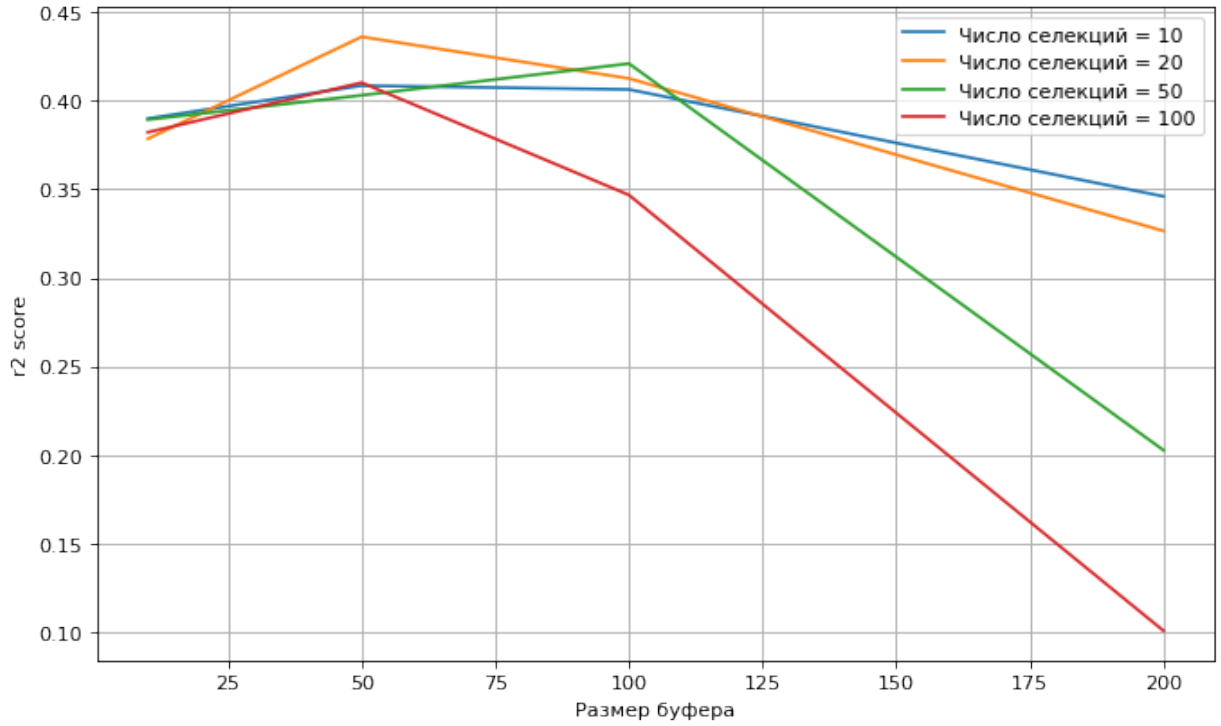


Из графика выше можно сделать вывод, что при большом размере буфера переобучение происходит намного быстрее.

Легко понять почему такое происходит.

Число нейронов первого слоя равно размеру буфера. Модели при длинных цепочках становятся более переобучаемыми и сама сеть с большим числом нейронов слоя также подвержена переобучению.

На графиках с меньшим размером буфера можем наблюдать эффект того, что по-началу качество растет при увеличении числа селекций МГУА, а потом начинает падать из-за того, что модель начинает переобучаться. При большом размере буфера пик смещается к меньшему числу селекций



На графике выше можно отметить, что при фиксированном числе селекций с с увеличением размера буфера качество также падает.

Это происходит из-за того, что модели также становятся сильно переобучаемыми и качество начинает портиться.

Отметим, что по графику можно определить пики качества, т.к. изначально увеличение размера буфера улучшает качество, по причине того, что модель с малым числом нейронов не может должным образом обучиться из-за малого числа параметров сети

7.7 Выводы из главы 5

В главе были рассмотрены результаты качества основных моделей машинного обучения, результаты взвешенного прогноза МГУА и результаты нейронной сети построенной на основе цепочек полученных Методом Группового Учета Аргументов.

Результаты показывают, что нейронная сеть на основе МГУА показала наилучшее качество в сравненни с градиентными бустингами и случайными лесами.

Было произведено сравнение производительности МГУА на CPU и GPU. Результаты показали, что при большом размере буфера параллельность на GPU начинает давать результаты немного хуже, чем на CPU ввиду того, что появляется много операций выполняемых на чистом python. Но на среднем размере буфера получилось достичь увеличения скорости более чем в 4 раза

	Размер буфера	Число селекций	No activation	relu	sigmoid	tanh
0	10	10	0.497805	0.492692	0.270795	0.292395
1	10	20	0.532584	0.544385	0.319384	0.296357
2	10	50	0.540568	0.539265	0.387749	0.290202
3	10	100	0.550456	0.553113	0.323423	0.278386
4	50	10	0.585764	0.582313	0.449373	0.345695
5	50	20	0.641754	0.641803	0.414151	0.393769
6	50	50	0.681675	0.675979	0.481731	0.421775
7	50	100	0.694425	0.705968	0.556345	0.414754
8	100	10	0.605713	0.603472	0.391367	0.398571
9	100	20	0.675749	0.670479	0.460058	0.416751
10	100	50	0.695577	0.693793	0.450016	0.419006
11	100	100	0.701524	0.700037	0.479508	0.408
12	200	10	0.635043	0.628722	0.491737	0.440087
13	200	20	0.665986	0.679799	0.497739	0.429419
14	200	50	0.706168	0.693506	0.540135	0.417545
15	200	100	0.704771	0.701315	0.519186	0.40246

7.8 Fish BCF Выборка

Для ислледования была сформирована выборка для прогнозирования фактора биоконцентрации

7.8.1 Сравнение реализаций МГУА на pytorch при запуске на cpu и gpu(cuda)

	Размер буфера	Число селекций	CPU время	GPU время	Отношение
0	10	10	8.890333	0.502121	17.705565
1	10	20	19.109052	1.027533	18.597028
2	10	50	47.679797	3.589779	13.282098
3	10	100	96.119612	7.102074	13.534020
4	50	10	44.709949	3.090833	14.465338
5	50	20	92.519639	9.074155	10.195951
6	50	50	239.806374	43.259489	5.543440
7	50	100	483.577194	100.006366	4.835464
8	100	10	90.525716	16.837088	5.376566
9	100	20	183.169925	34.893798	5.249355
10	100	50	475.774170	110.561080	4.303270
11	100	100	960.967428	238.709701	4.025674

7.8.2 Сравнение классических методов

	Full	PCA 70
Линейная регрессия	-1.1029+18	0.64
Случайный лес	0.68	0.616
Градиентный бустинг	0.711	0.654
Ridge	0.67	0.640

7.8.3 Нейронная сеть на основе МГУА

	Размер буфера	Число селекций	No activation
0	10	10	0.476841
1	10	20	0.481251
2	10	50	0.516124
3	10	100	0.513939
4	50	10	0.546071
5	50	20	0.634505
6	50	50	0.654914
7	50	100	0.651555
8	100	10	0.61461
9	100	20	0.671212
10	100	50	0.683306
11	100	100	0.676058
12	200	10	0.661144
13	200	20	0.695154
14	200	50	0.690325
15	200	100	0.667116

8 Заключение

В работе были описаны основные методы решения задач QSAR. В том числе способы векторизации данных и основные алгоритмы, которые зачастую показывают хорошие результаты. Также в попытках увеличить скорость обучения были использованы облачные технологии вычислений. Было проведено множество вычислительных экспериментов.

Из всех приведённых вычислительных экспериментов можно сделать несколько выводов:

Реализация Метода Группового Учета Аргументов на языке python отличатся простотой написания, но при этом очень медленная

Реализация с помощью тензоров pytorch показала себя намного лучше.

Алгоритм Метода Группового Учета Аргументов неплохо поддается распараллеливанию на cuda, получилось добиться хороших результатов.

Библиотека pytorch при запуске на cuda была удобной для реализации алгоритмов и показала хорошую скорость.

Полученные результаты качества показали, что МГУА выигрывает в точности у основных методов машинного обучения

9 Список литературы

1. Кумсков М.И. Выбор формальных элементов, покрывающий кластер молекулярных графов в метрике, заданной на их струк- турных символьных спектрах. // В сб. «Модели и архитекту- ры нейронных сетей в задаче «структура-свойство». М.: Изд- во «МАКСПРЕСС», 2020.

2. Кумсков М.И. «Методология Прогнозирования Свойств Хи- мических Соединений И Ее Программная Реализация» - Авто- реферат докторской диссертации, М.: Вычислительный Центр РАН. - http://istina.msu.ru/dissertations/3541505/

3. Кумсков М.И., Митюшев Д.Ф. Применение метода группового учета аргументов (МГУА) для построения коллективных оценок свойств органических соединений на основе индуктивного перечисления их структурных спектров / Проблемы управления и информатики, 1996, No4.

4. F.K. Brown. Chapter 35. Chemoinformatics: What is it and How does it Impact Drug Discovery (англ.) // Annual Reports in Med. Chem. : journal. — 1998. — Vol. 33. — P. 375. — doi:10.1016/S0065-7743(08)61100-8.

5. Brown, Frank. Editorial Opinion: Chemoinformatics – a ten year update (англ.) // Current Opinion in Drug Discovery Development : journal. — 2005. — Vol. 8, no. 3. — P. 296—302.

6. R. Todeschini, V. Consonni: Handbook of Molecular Descriptors. WILEY-WCH Publishers, Weinheim, 2000. ISBN 3-527-29913-0

7. van der Maaten L.J.P., Hinton G.E. Visualizing Data Using t-SNE // Journal of Machine Learning Research. — 2008. — Ноябрь (т. 9).

8. Скворцова М.И., Станкевич И.В., Палюлин В.А., Зефиров Н.С. Концепция молекулярного подобия и ее использование для прогнозирования свойств химических соединений. (в публикации)

9. Журавлев Ю. И. Об алгебраическом подходе к решению задач распознавания и классификации. - М.: Наука, 1978, вып. 33

10. Сенько О.В. Использование процедуры взвешенного голосования по системе базовых множеств в задачах прогнозирования. – Журнал вычислительной математики и математической физики, 1995, 35(10).

11. Рязанов В.В. О построении оптимальных алгоритмов распознавания и таксономии (классификации) при решении прикладных задач. – В кн.: Распознавание, классификация, прогноз: Математические методы и их применение – М.:Наука, 1998, вып. 1

12. Кумсков М.И., Смоленский Е.А., Пономарева Л.А., Митюшев Д.Ф., Зефиров Н.С. Системы структурных дескрипторов для решения задач «структура-свойство». - Доклады Академии Наук, 1994, 336

13. O.Ivanciuc, S.L.Taraviras, D.Cabrol-Bass - Journal of Chemical Information and Computer Sciences, 40

14. В. Магнусон, Д. Харрис, С. Бейсак В кн. Химические приложения топологии и теории графов. (Ред. Р. Кинг) - М.:Мир, 1987

15. Журавлев Ю.И., Гуревич И.Б. Распознава- ние образов и распознавание изображений. /

В сб.

Распознавание. Классификация. Прогноз. Мате- матические методы и их применение.

Вып.2, М.:

Наука, 1989, с.5-72.

16. Харари Ф. Теория графов. - Пер. с англ., - М.: Мир, 1973, 300с.

17. Кумсков М.И. «Методология Прогнозирова- ния Свойств Химических Соединений И Ее Про- граммная Реализация» - Автореферат докторской

диссертации, М., Вычислительный Центр РАН. - https://istina.msu.ru/dissertations/3541505/

18. Кохов В.А. Метод количественного опре- деления сходства графов на основе структур- ных спектров. / Изв. РАН, Сер. Техническая. ки- бернетика, 1994, No5, с.143-159.

19. Deep learning, stochastic gradient descent and diffusion maps Carmina Fjellström, Kaj Nyström

20. Ligandformer: A Graph Neural Network for Predicting Compound Property with Robust Interpretation Jinjiang Guo, Qi Liu, Han Guo, Xi Lu

21. On the Irregularity of Some Molecular Structures Hosam Abdo, Darko Dimitrov, Wei Gao

22. Intriguing Usage of Applicability Domain: Lessons from Cheminformatics Applied to Adversarial Learning Luke Chang, Katharina Dost, Kaiqi Zhao, Ambra Demontis, Fabio Roli, Gill Dobbie, Jörg Wicker

23. Aggarwal, C.C., Hinneburg, A., Keim, D.A.: On the surprising behavior of distance metrics in high dimensional space. In: International conference on database theory, pp. 420–434. Springer (2001)

24. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Srndi´c, N., Laskov, P., Giacinto, G., Roli, ˇ F.: Evasion attacks against machine learning at test time. In: Joint European conference on machine learning and knowledge discovery in databases, pp. 387–402. Springer (2013)

25. Biggio, B., Roli, F.: Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition 84, 317–331 (2018)

26. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In: International Conference on Learning Representations (2018)

27. Cao, X., Gong, N.Z.: Mitigating evasion attacks to deep neural networks via region-based classification. In: Proceedings of the 33rd Annual Computer Security

Applications Conference, pp. 278–287 (2017)

28. Carlini, N., Wagner, D.: Magnet and "efficient defenses against adversarial attacks" are not robust to adversarial examples. arXiv preprint arXiv:1711.08478 (2017)

29. Carlini, N., Wagner, D.: MagNet and "Efficient Defenses Against Adversarial Attacks" are Not Robust to Adversarial Examples. arXiv preprint arXiv:1711.08478 (2017)

30. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57 (2017)

31. Croce, F., Hein, M.: Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In: International Conference on Machine Learning, pp. 2206–2216. PMLR (2020)

32. Dalvi, N., Domingos, P., Sanghai, S., Verma, D.: Adversarial classification. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 99–108 (2004)

33. Dehak, N., Dehak, R., Glass, J.R., Reynolds, D.A., Kenny, P., et al.: Cosine similarity scoring without score normalization techniques. In: Odyssey, p. 15 (2010)

34. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in neural information processing systems, pp. 2962–2970 (2015)

35. Gadaleta, D., Mangiatordi, G.F., Catto, M., Carotti, A., Nicolotti, O.: Applicability domain for qsar models: where theory meets reality. International Journal of Quantitative Structure-Property Relationships (IJQSPR) 1(1), 45–63 (2016)

36. Gilmer, J., Adams, R.P., Goodfellow, I., Andersen, D., Dahl, G.E.: Motivating the rules of the game for adversarial example research. arXiv preprint arXiv:1807.06732 (2018)

37. S. Zheng, X. Yan, Y. Yang, J. Xu, Identifying Structure–Property Relationships through SMILES Syntax Analysis with Self-Attention Mechanism, J. Chem. Inf. Model. 59 (2019) 914–923. https://doi.org/10.1021/ACS.JCIM.8B00803.

38. R. Gómez-Bombarelli, J.N. Wei, D. Duvenaud, J.M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T.D. Hirzel, R.P. Adams, A. Aspuru-Guzik, Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules, ACS Cent. Sci. 4 (2018) 268–276. https://doi.org/10.1021/ACSCENTSCI.7B00572.

39. M.J. Kusner, B. Paige, J.M. Hernández-Lobato, Grammar Variational Autoencoder, 34th Int. Conf. Mach. Learn. ICML 2017. 4 (2017) 3072–3084. https://arxiv.org/abs/1703.01925v1 (accessed October 19, 2021).

40. H. Dai, Y. Tian, B. Dai, S. Skiena, L. Song, Syntax-Directed Variational Autoencoder for Structured Data, 6th Int. Conf. Learn. Represent. ICLR 2018 - Conf. Track Proc. (2018). https://arxiv.org/abs/1802.08786v1 (accessed October 19, 2021).

41. W. Jin, R. Barzilay, T. Jaakkola, Junction Tree Variational Autoencoder for Molecular Graph Generation, 35th Int. Conf. Mach. Learn. ICML 2018. 5 (2018) 3632–3648. https://arxiv.org/abs/1802.04364v4 (accessed October 19, 2021).

42. Z. Alperstein, A. Cherkasov, J.T. Rolfe, All SMILES Variational Autoencoder, (2019). https://arxiv.org/abs/1905.13343v2 (accessed October 19, 2021).

43. S. Mohammadi, B. O'Dowd, C. Paulitz-Erdmann, L. Goerlitz, Penalized Variational Autoencoder for Molecular Design, (2021). https://doi.org/10.26434/CHEMRXIV.7977131.V2.

44. M. Galushka, C. Swain, F. Browne, M.D. Mulvenna, R. Bond, D. Gray, Prediction of chemical compounds properties using a deep learning model, Neural Comput. Appl. 2021. (2021) 1–22. https://doi.org/10.1007/S00521-021-05961-4.

45. M. Lovrić, T. Đuričić, H.T.N. Tran, H. Hussain, E. Lacić, M.A. Rasmussen, R. Kern, Should We Embed in Chemistry? A Comparison of Unsupervised Transfer Learning with PCA, UMAP, and VAE on Molecular Fingerprints, Pharm. 2021, Vol. 14, Page 758. 14 (2021) 758. https://doi.org/10.3390/PH14080758