ХИМИЧЕСКИЙ
ФАКУЛЬТЕТ
МГУ ИМЕНИ
М.В. ЛОМОНОСОВА

teach-in
ЛЕКЦИИ УЧЕНЫХ МГУ

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ В ХИМИИ И МАТЕРИАЛОВЕДЕНИИ

МИТРОФАНОВ
АРТЕМ АЛЕКСАНДРОВИЧ и др.

ХИМФАК МГУ

КОНСПЕКТ ПОДГОТОВЛЕН СТУДЕНТАМИ, НЕ ПРОХОДИЛ ПРОФ. РЕДАКТУРУ И МОЖЕТ СОДЕРЖАТЬ ОШИБКИ. СЛЕДИТЕ ЗА ОБНОВЛЕНИЯМИ НА VK.COM/TEACHINMSU.

ЕСЛИ ВЫ ОБНАРУЖИЛИ ОШИБКИ ИЛИ ОПЕЧАТКИ, ТО СООБЩИТЕ ОБ ЭТОМ, НАПИСАВ СООБЩЕСТВУ VK.COM/TEACHINMSU.

БЛАГОДАРИМ ЗА ПОДГОТОВКУ КОНСПЕКТА
АСПИРАНТА ХИМИЧЕСКОГО ФАКУЛЬТЕТА МГУ
КИХАЙ ТАТЬЯНУ ФЕДОРОВНУ

Оглавление

Лекция 1. Лекция. Машинное обучение. Место ИИ в химии и материаловедении. Основные понятия и методы.

Наука о данных — это междисциплинарная область, которая использует научные методы, процессы, алгоритмы и системы для извлечения знаний и идей из зашумленных, структурированных и неструктурированных данных, а также для применения знаний и практических идей из данных в широком спектре прикладных областей.

Искусственный интеллект (ИИ) — это интеллект, демонстрируемый машинами, в отличие от естественного интеллекта, демонстрируемого животными, включая людей. Ведущие учебники ИИ определяют эту область как изучение *интеллектуальных агентов* – любая система, которая воспринимает свое окружение и предпринимает действия, максимизирующие его шанс на достижение своих целей".

Машинное обучение (ML) – это изучение компьютерных алгоритмов, которые могут автоматически улучшаться с помощью опыта и использования данных".

Общие принципы

Данные: сбор, очистка, разметка
Задачи: классификация, регрессия, кластеризация, генерация, оптимизация, анализ
Методы: «классическое» машинное обучение, нейронные сети, эволюционные алгоритмы, роевой интеллект
Human-readable vs. machine-readable
Функции потерь и метрики качества: классификационные, регрессионные, подобия
Тестовые наборы данных: как избежать переобучения?
Область применимости модели

Пример: QSPR (Количественные соотношения структуры и свойства):

В работы пытались создать модели машинного обучения, которые смогли бы предсказать константы комплексообразования различных органических соединений с металлами. Входные данные: структура органического соединения. На рисунке 1.1. представлена схема такого эксперимента:

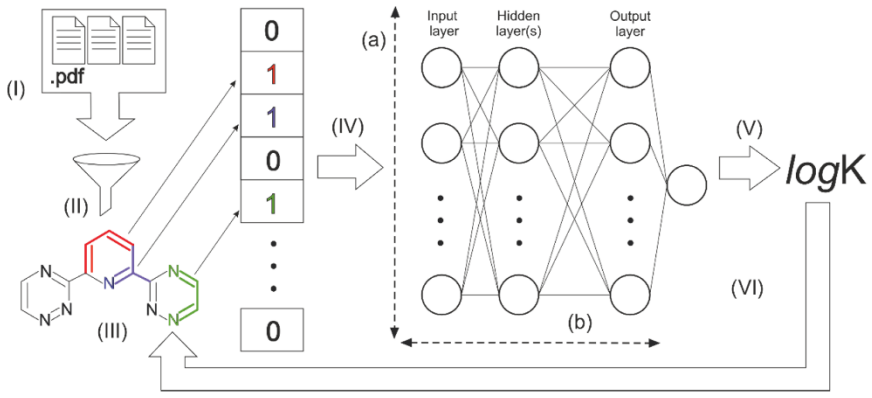


Рисунок 1.1. Общая схема процесса QSPR.

На первом этапе собирается база данных (иногда из литературы или уже из известных баз данных). Такая база должна быть очищена, стандартизирована по

условиям эксперимента (например, нет смысла сравнивать соединения, которые устойчивы при разной ионной силе). Далее данные о структуре (из двумерного изображения) переводят в вектора известной постоянной длины (вектор-признак). Данные вектора отправлялись на модель машинного обучения и выводился результат.

Данные. Основа для всего. Обычно говорится про большие данные, но их количество зависит от чистоты данных, метода, желаемых результатов. Обычно на диаграмме 1.2 данные находятся внизу данной пирамиды. Под данными подразумеваются факты, сигналы, измерения, обычно что-то не структурированное. Данные часто не натированы, разнородны, в неудобных форматах. Прежде чем использовать данное «болото» данных, они должны пройти процедуру упорядочивания данных, добавления к ним контекста, метаданных, добавления лейбла к таким данным, очистку, критический осмотр и т.д. Если кратко, то нужно обработать данные крайне полно, чтобы их можно было использовать.

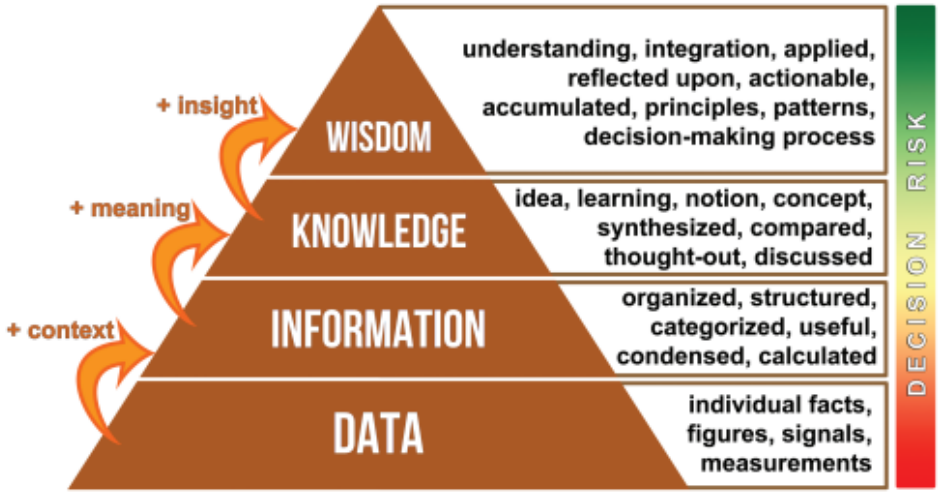


Рисунок 1.2. Пирамида данных

В данном курсе невозможно однозначно сказать, как превратить данные в информацию, так как данный вопрос довольно специфичен для каждой отдельной задачи. В данном курсе можно только показать какие данные можно считать информацией. Информация – это что-то структурированное, с известной погрешностью, однородное, доступное. На основании информации можно получить новое знание. На основании знания можно получить какую-то мудрость, глобальную закономерность, перейти от эмпирических данных к глобальным закономерностям. Алгоритмы искусственного интеллекта помогают перейти от информации к знаниям.

Чаще всего данных должно быть много для работы машинного обучения. И традиционные алгоритмы машинного обучения способны работать с каким угодно количеством данных, но не всегда увеличения количества данных приводит к улучшению производительности за пределами внутренних границ. Можно сказать, что в этом ограничении открылось преимущество нейронных сетей, глубокого обучения. Единственное, что здесь сложно – это определение количества данных. Для работы методов машинного обучения необходимо собрать хотя бы сотню, тысячи точек.

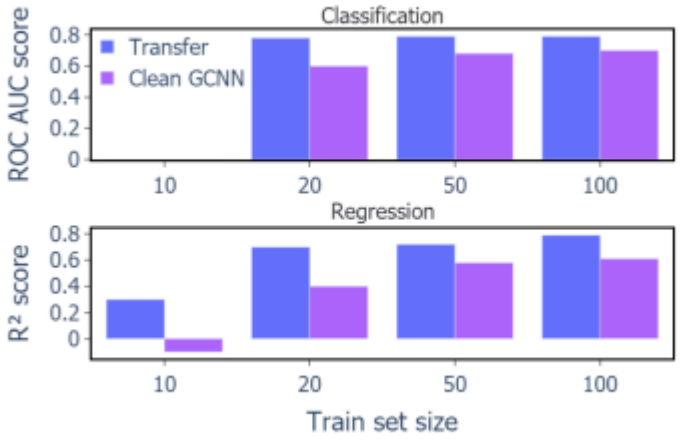


Рисунок 1.3. Пример расчета свойств органических молекул с помощью трансферного метода.

Но в принципе существуют техники (например, трансферное обучение), которые позволяют построить модели на маленьких данных. На рисунке 1.3. показан пример такой работы, где показали, что можно предсказать свойство органической молекулы при этом, модель тренировали на нескольких десятках молекул, хотя обычно речь идет о тысяче молекул.

Feature vector (вектор свойств, вектор признаков):

• Могут представлять собой данные (численные)

• Скорее всего, придется векторизовать данные других типов – текст, картинки, структурные формулы:

1) Записываем признаковое описание - много разных тестов

2) Записываем структуру - кодируем 2/3D разного размера в 1/2D постоянного размера

3) Предлагаем алгоритму сделать это за нас.

• Проверяем масштабирование (scaling)

• Формируют feature space

• Определяют Applicability domain (область применения)

Суммируя:

Основные определения: data science, artificial intelligence, machine leaning

Переход количества в качество

Нужно предварительно обрабатывать данные

Данные могут быть размеченные и неразмеченные

В data science существует три класса задач:

Обучение с учителем (работа с размеченными данными):

• Классификация

• Регрессия

Обучение без учителя (без размеченных данных)

• Кластеризация

Обучение с подкреплением (нет разметки данных, но можно выдать алгоритму информацию на граде, которую он получает в процессе обучения)*:*

• Генерация

Методы

В машинном обучении:

• Naïve Bayes (Байесовская статистика)

• Support vector machine (Метод опорных векторов)

• Nearest neighbors (Метод ближайших соседей)

• Decision tree/Random forest/XGBoost (Методы дерева решений)

• Linear regression (Линейная регрессия)

• Deep learning (Глубокое обучение – использование нейроных сетей, и когда пытаются абстрагировать идею молекулы внутри самой сети)

Генерация новых соединений:

• Evolutionary algorithms (Эволюционные алгоритмы)

• GAN (Генеративно состязательные сети с памятью)

• Swarm Intelligence (Роевой интеллект)

Other:

• Autoencoders (Автокодировщики)

• PCA/t-SEE (Построение двумерного массива данных из трехмерных массивов)

Нейронные сети

Нейронные сети позволяют сделать переход от количества к качеству на достаточно небольших данных. Кроме того, у нейронных сетей довольно простая архитектура, простая архитектура отдельных узлов. Название произошло от того, что она похожа на нейронную сеть человека (рис. 1.4.).

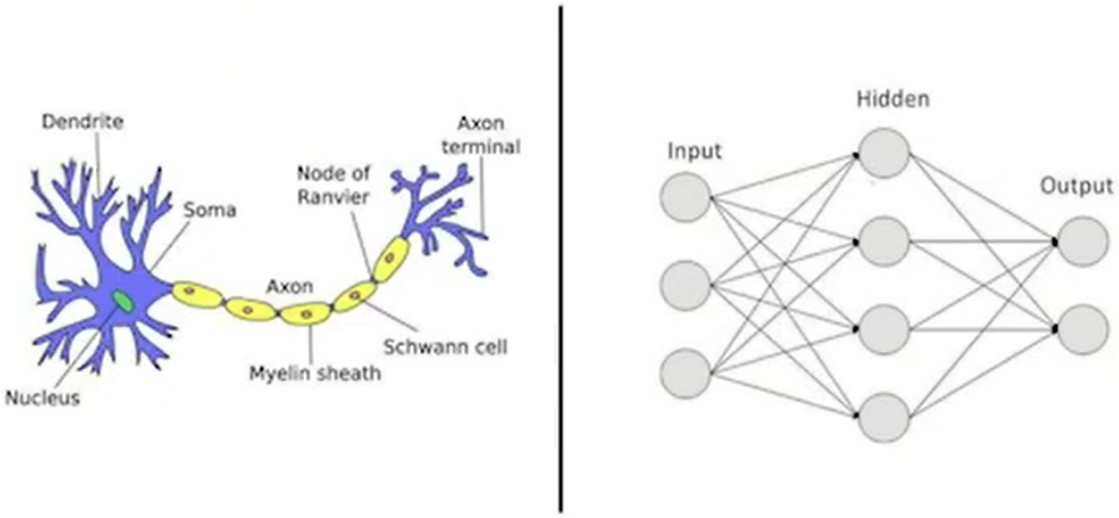


Рисунок 1.4. Схематичное изображение нейронной сети

Метрики. Функции потерь

Функция потерь - оптимизируется (минимизируется) в задаче:

• Определена для типа задачи (классификация/регрессия/генерация)
• Должна учитывать несбалансированность набора данных
• Часто должна быть дифференцируема
• Часто должна иметь физический/химический смысл
• Может быть составной
• Может быть связана с метрикой

Метрика - оценивается после обучения:

• Определена для типа задачи (классификация/регрессия/генерация)
• Должна учитывать несбалансированность набора данных
• Часто должны иметь физический/химический/логический смысл
• Может быть составной
• Лучше, когда их несколько

Суммируя:

• Нужна функция, которую мы хотим оптимизировать.
• Нужны метрики, которые мы хотим демонстрировать миру.

Что для этого нужно? По принципу: имея набор данных (Data Set) неплохо бы часть этого набора данных отложить «про запас». Это нужно для того, чтобы проверить как работает данная тренировочная модель.

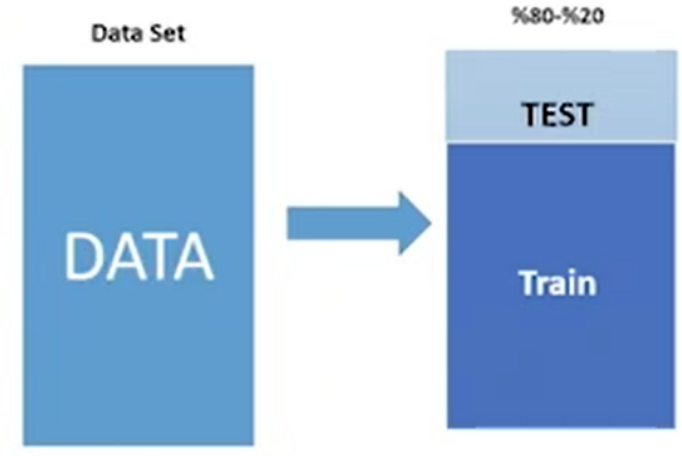


Рисунок 1.5. Схема использования данных в любой модели

Пример:

Кросс-валидация. Можно разбить тренировочный набор на пять курков, отделять 1/5, потом на остальной части тренировать модель. Так сделать с каждой из частей. В итоге получим 5 моделей, и новая модель будет представлять, например, среднее арифметическое от остальных моделей (Рисунок 1.6).

Область применимости

Научный подход от ненаучного отличает именно область применимости. Отметим, что модели обычно работают хорошо где-то вблизи тех данных, на которых они тренировались. Всегда нужно использовать те или иные критерии для оценки области применимости модели.

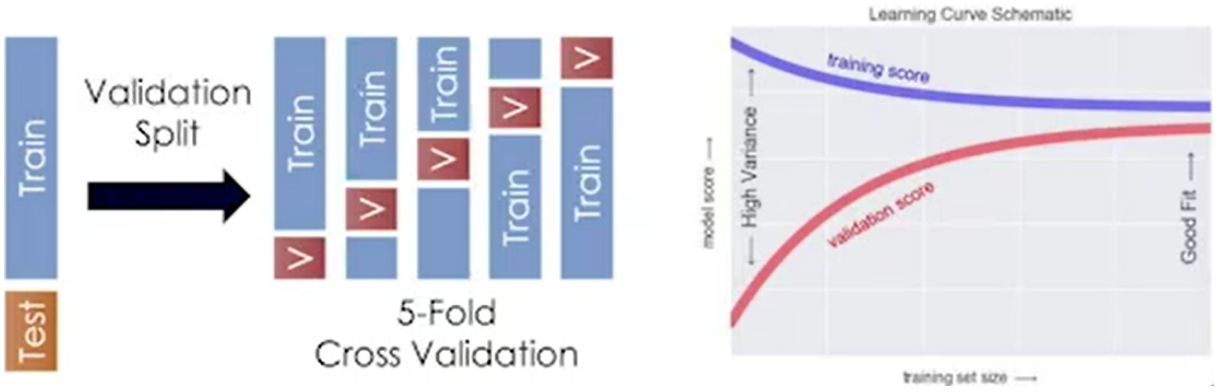


Рисунок 1.6. Концепция кросс-валидации.

Пример:

В данной работе взяли данные о веществах, которые относились к различным классификациям лекарственных средств. Авторы предложили модель машинного обучения и натренировали новое описание различных свойств молекул. С помощью этого они смогли предсказать биораспределение лекарств, которое было проверено на мышах (результаты хорошо совпали) (рис.1.7).

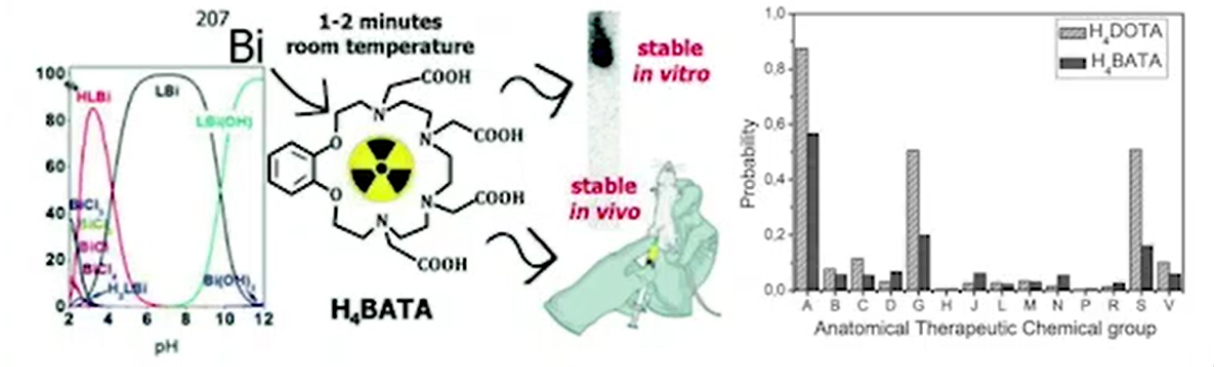


Рисунок 1.7. Пример тренировки новой модели для предсказания биораспределения веществ.

Занятие 2. Семинар. Введение. Python, jupyter notebook

Python – один из самых популярных языков программирования.

Индекс TIOBE - индекс, оценивающий популярность языков программирования, на основе подсчёта результатов поисковых запросов, содержащих название языка. Для формирования индекса используется поиск в нескольких наиболее посещаемых порталах: Google, Blogger, Wikipedia, YouTube, Baidu, Yahoo, Bing, Amazon. У Python самый высокий индекс TIOBE.

Области применения:

профессиональное программирование:

Web-разработка: backend (Фреймворки Django, Flask)

Разработка игр: Battlefield, Sims 4 (для моддинга), Civilization IV, World of Tanks

Десктоп программы: GDP, Bleader, инсталлеры

data science:

машинное обучение

анализ длинных

визуализация данных

автоматизация процессов:

програмирование Raspberry Pi

управление банкоматами

управление стансами, печами и другими оборудованием и приборами

первый язык программирования (обучение)

Откуда такая популярность?

1) На Python очень часто получается решить СЛОЖНУЮ проблему БЫСТРО (за меньшее число строк кода)

2) Огромное, нетоксичное комьюнити

3) Кроссплатформенность

4) Огромное количество библиотек

5) Простой в изучении

В этом семинаре будет использоваться имеено Python 3 (от Python 2 отличается синтаксисом)

Установка:

Несколько вариантов использования:

https://www.anaconda.com/oroducts/individual

В терминале/PowerShell пишем jupiter notebook, автоматически открывается вкладка в браузере

- Где и как еще можно писать на Python?

https://colab.research.google.com/?utm_source=scs-index - будем использовать на этом курсе;

https://notepad-plus-plus.org/downloads/

https://www.ietbrains.com/ru-ru/pycharm/

jupyterlab - улучшенная версия jupyter notebook, на стадии развития

Если смотреть на интерфейс, то можно заметить, что здесь все разбито на ячейки. Ячейка – форматер, в который можно писать либо код, либо текст:

Рисунок 2.1. Интерфейс в colab.research.google.com

Помимо кода можно: Markdown (регулировка заголовков)

заголовки поменьше

еще меньше

и еще

+Shift + enter

Получается:

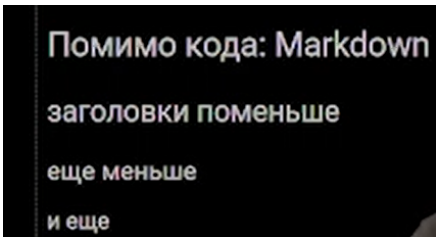


Рисунок 2.2. Написание разных заголовков

Можно сделать ссылку на сайт: [Python](https://python/org)

Основные команды:

Команды:

1. Shift+Enter - запуск ячейки
2. Enter - вход в режим редактирования (зеленая полоска)
3. Esc- выход из режима редактирования в коммандный режим (синяя полоска)

• Стрелки вверх и вниз- перемещения по ячейкам
• Space и Shift+Space - скролл вниз и вверх
• A- вставить ячейку над выделенной (Above)
• B - вставить ячейку под выделенной (Below)
• C,V,X-Копировать, вставить, вырезать ячейку
• DD - удалить ячейку
• Y- вид ячейки: программа
• M- вид ячейки: Markdown
• H - напоминалка с сокращениями
• P- набор комманд с коротким описанием

Элементы синтаксиса

1. Сложение: 1 + 2 + 3 + 4 + 5
   Вывод: 15
   Python просто читает команду, выполняет и переходит дальше (нет этапа компиляции)
2. Введение переменных:

x = 1 + 2 + 3 + 4 + 5
Вывод: x (сохранили числовое значение переменной)

3. Динамическое строгое типизирование переменных
   print (type(x)) (<class 'int'>)
   x = "Привет"
   print (type(x)) (<class 'str'>)
   x
   Вывод: Привет
4. Вывод информации
   Введем
   y = "Пока"
   x
   y
   Вывод: Пока
   Python выводит только последнее значение, поэтому нужно сделать так:
   Print (x, y)
   Вывод: Привет Пока

Типы данных

Числа:
целые: int
с плавающей точкой: float
комплексные: complex
Контейнеры:
строки: str
кортеж: tuple
список: list
словарь: dict
множество: set
Вызываемые объекты:
функции
функции-генераторы
методы объектов
классы
модули
Файлы (file)
Специальные типы (None, Notimplemented) и т.д.
Познакомимся с простыми (не составными) типами переменных Python 3
целое число, или înt
-10023104013240234543956293874568092475892
вещественные числа, называются еще float

1.6E123
Вывод: 1.6+123
комплексные числа:
640E3 + 4.10j
Вывод: (640000+4.1j)

Возможные математические действия

Присвоение по значению происходит при использовании неизменяемых типов (числа, строки)

```
x = 1
y = 2
print(f"x = {x}, y = {y}")
x = 1, y =2
x = 3
y = x
print(f"x = {x}, y = {y}")
x = 5
print(f"x = {x}, y = {y}")
```

присвоение по ссылке происходит при использовании изменяемых типов (лист - набор чисел)

```
x = [1, 2, 3, 4]
print( "x = {x}, y = {y}")
y = x
print(f"x = {x}, y = {y}")
x.append(5) # добавляем в x еще один элемент, равный пяти
print(f"x = (x}, y = {y}")
y.remove(4) а удаляем элемент равный 4
print(f"x = {x}, y = {y}")
```

Классические математические действия:

```
x = 42
y = 1007
print(f"x + y = {x + y}")
print(f"x - y = {x - y}")
print(f"x * y = {x " y}")
print(f"x / y = {x / y}")
print(f"x // y = {x // y}") # целочисленное деление
print (f"x % y = {x % y}") # остаток от деления
x + y = 1049
x - y = -965
x * y = 42294
x / y = 0.041700043694141016
```

x // y = 0

x % y = 42

Операции сравнения

a==b a равно b

a < b a больше b

a < b a меньше b

a <= b a меньше или равно b

a >= b a больше или равно b

a ?= b a не равно b

Занятие 3. Семинар. Обработка данных с помощью Python.

Первым китом, на котором стоит Python, является скорость. Сам по себя язык не очень быстрый. И когда дело качается большого анализа данных, то python – не самый лучший выбор. Но в данной ситуации есть выход – библиотека Numpy:
numpy — создание и обработка массивов
numpy.linalg — реализует операции линейной алгебры (простое умножение векторов и матриц есть в базовом варианте);
numpy.random — реализует функции для работы со случайными величинами;
numpy.fft — реализует прямое и обратное преобразование Фурье.

Нам интересна та часть, которая отвечает за сохранение табличных, векторных данных, тензорами:

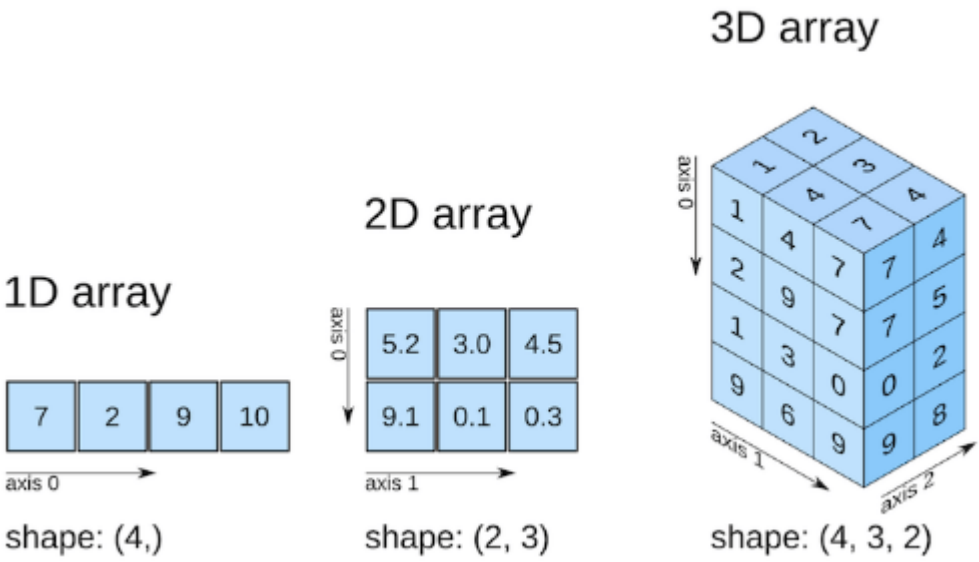


Рисунок 3.1. Примеры тензоров

- Скаляр— 0D тензор
- Вектор— 1D тензор с размерностью (features)
- Матрица — 2D тензор с размерностью (samples, features)
- Изображение — 3D тензор с размерностью (height, width, channels)
- Видео— 4D тензор с размерностью (samples, height, width, channels)

Переходя к практике:

```
# Создание массивов

import numpy as np

a = np.array([1, 2, 3])   # Создание путем преобразования из list

print(type(a))

print(a.shape)
```

```
a = np.zeros((2,2))   # Массив нулей
print(a)              # "[[ 0.  0.]
                      #  [ 0.  0.]]"
b = np.ones((1,2))    # Массив единиц
print(b)              # "[[ 1.  1.]]"
d = np.eye(2)         # Единичная матрица
print(d)              # Prints "[[ 1.  0.]
                      #  [ 0.  1.]]"
e = np.random.random((2,2))  # Массив случайных величин [0, 1)
print(e)                     # "[[ 0.91940167  0.08143941]
                             #  [ 0.68744134  0.87236687]]"
f = np.array([[1, 2, 3], [4, 5, 6]]) # Создание массива по подобию с дргуим массивом
f_like = np.ones_like(f)
```

Типы данных в Numpy

'int': [numpy.int8, numpy.int16, numpy.int32, numpy.int64]
'uint': [numpy.uint8, numpy.uint16, numpy.uint32, numpy.uint64]
'float': [numpy.float16, numpy.float32, numpy.float64, numpy.float128]
'complex': [numpy.complex64, numpy.complex128, numpy.complex256]
'others': [bool, object, bytes, str, numpy.void]

Доступ к элементам и слайсы

Когда научились работать с заполнением и созданием массивов, хорошо бы понять как манипулировать данными внутри массива:

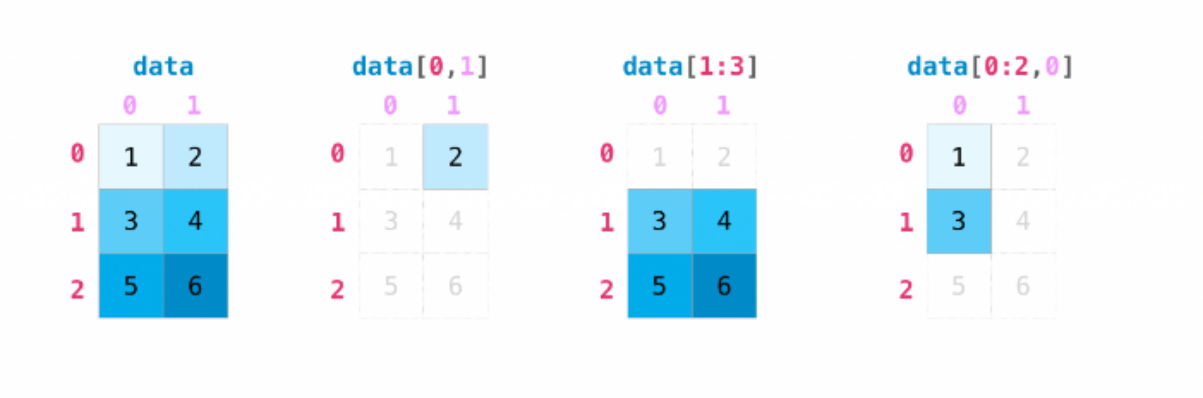


Рисунок 3.2. Пример использования элементов в матрицах

Давайте разберем какие манипуляции можно сделать:

```
        # Индексирование
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a[1])
print(a[2, 0])
        # Слайсы
# [x, y] Первое число(слайс) определяет выбранные строки,
#        Второе определяет стоблцы
b = a[:2, 1:3]
print(b)
```

```
        # Изменение слайса также изменит исходный массив, поскольку оба ссылаются
на одну область данных
print(a[0, 1])   # "2"
b[0, 0] = 77
print(a[0, 1])   # ???
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)  # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape)  # Prints "[[5 6 7 8]] (1, 4)"
        # Переменная размерность получаемого массива
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)  # "[ 2  6 10] (3,)"
print(col_r2, col_r2.shape)  # "[[ 2]
                             #   [ 6]
                             #   [10]] (3, 1)"
        # Булевый массив для доступа к элементам
arr = np.array([[1, 2, 3], [4, 5, 6]])
idx = np.array([[False, False, True], [ True, False, True]])
print(arr[idx])
arr = np.array([[1, 2, 3], [4, 5, 6]])
idx = np.array([[False, False, True], [ True, False, True]])
arr[idx] = 0
print(arr)
```

Арифметические операции над массивами

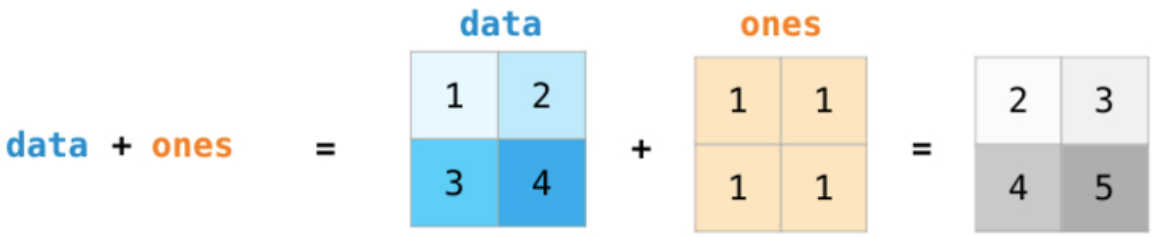


Рисунок 3.3. Пример сложения матриц

Посмотрим пример:

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Поэлементная сумма
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))
# Поэлементная разность
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
```

```
print(x - y)
print(np.subtract(x, y))
# Поэлементное произведение
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))
## Поэлементное деление
# [[ 0.2        0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))
# Поэлементное извлечение корня
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

Изменение размера массивов

Данную операцию иногда удобно делать для представления данных в разном виде. Перейдем сразу к практике:

```
# Определение размера массивов
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
# Размерность массива
print(a.ndim)
# Размер по осям
print(a.shape)
# Число элементов массива
print(a.size)
# Изменение размера по осям
A = np.array([1, 2, 3, 4, 5, 6])
A = A.reshape(2, 3)
print(A)
A = np.arange(24)
B = A.reshape(4, -1)
C = A.reshape(4, -1, 2)
print(B.shape, C.shape)
# Превратить массив в одномерный
A = np.array([[1, 2, 3], [4, 5, 6]])
B = A.reshape(-1)
# Транспонирование матрицы
C = np.arange(24).reshape(4, -1, 2)
print(C.shape, np.transpose(C).shape)
print()
print(C[0])
print()
print(C.T[:, :, 0])
```

Вспомним, как осуществляется матричное умножение:

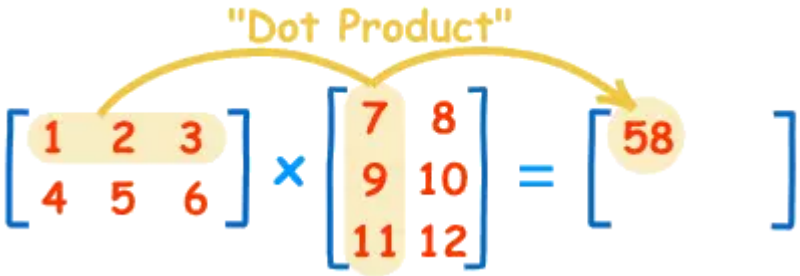


Рисунок 3.4. Осуществление матричного умножения.

Рассмотрим, как это сделать в программе:

```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([11, 12])
# Умножение двух векторов
print(v.dot(w))
print(np.dot(v, w))
# Умножение матрицы на вектор
print(x.dot(v))
print(np.dot(x, v))
# Перемножение двух матриц
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
e = np.random.randint(5, 10)
print(np.unique(e))
```

Зачем мы используем numpy, если все это можно делать без пакета. А дело в том, что библиотеки увеличивают скорость расчета примерно в 3 порядка. Чем больше будет размерность матрицы, тем сильнее разница будет ощущаться.

Pandas

Данный инструмент/библиотека предназначена для работы с таблиц. Давайте создадим таблицу:

```
import pandas as pd
# Создание датафрейма

df = pd.DataFrame([[1,'Bob', 'Builder'],
                   [2,'Sally', 'Baker'],
                   [3,'Scott', 'Candle Stick Maker']], columns=['id','name', 'occupation'])
df
# кусок csv
```

Далее пойдет работа с файлом, который предлагают разработчики курса. Но в принципе можно использовать любую таблицу с данными. Для этого надо подключить google.drive:

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

Далее прописать точный путь данных:

```
# Работа с csv файлами
df = pd.read_csv("/content/drive/MyDrive/ML in chemistry/3_seminar/nyc-rolling-sales.csv")
df[:100].to_csv("/content/drive/MyDrive/ML in chemistry/3_seminar/nyc-rolling-sales_sample.csv")
# Общая информация о таблице
df.info()
```

Какие операции можно сделать с данной библиотекой:

```
# Выведет первые 3 строчки
df.head(3)
# Выведет последние 5 строчек
df.tail(5)
# Преобразование данных из таблицы
print(df.columns.tolist())
print(df['YEAR BUILT'].tolist()[:10])
# Запросы (query) в pandas
print(df.shape)
df.dropna().shape
df.query('`RESIDENTIAL UNITS` == `COMMERCIAL UNITS`')
# Убрать ненужные колонки
df.drop(['ZIP CODE', 'SALE DATE'], axis=1)
```

Для совершения арифметических действии в таблицах нужно перевести нужные значения в нужный формат:

```
# df[df['SALE PRICE'] < "6625000"]
df['SALE PRICE'] = pd.to_numeric(df['SALE PRICE'], errors='coerce')
df[df['SALE PRICE'] < 6625000]
# Вывести все уникальные значения в столбце
vals = list(enumerate(np.unique(df["NEIGHBORHOOD"])))
change_dict = {val[1]: val[0] for val in vals}
change_dict
# Приведение данных столбца к определенному типу
df["SALE PRICE"] = pd.to_numeric(df["SALE PRICE"], errors='coerce')
# с заменой ошибок на None
# Так тоже можно, но с нюансами
df["NEIGHBORHOOD"]=df["NEIGHBORHOOD"].map(lambda x: change_dict[x])
# Запросы (query) в pandas
print(df.shape)
df.dropna().shape
#анализ столбцов между собой
df.query('`RESIDENTIAL UNITS` == `COMMERCIAL UNITS`')
# иногда нужно ограничить данные
df.query("`RESIDENTIAL UNITS` != 0 & `COMMERCIAL UNITS` != 0")
val_df = df.query("`TOTAL UNITS` != 0")
val_df.dropna(subset=['SALE PRICE'])
val_df = val_df.query("`SALE PRICE` != 0")
```

```
val_df = val_df[val_df['SALE PRICE'].notna()]
val_df = val_df[val_df['SALE PRICE'] > 1000]
val_df
```

Визуализация

Самым используемым методом визуализации является **matplotlip.** По сути библиотека для построения графиков**.** Поработаем с ним:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.rcParams["figure.figsize"] = (8, 8)
# data science не только модели
```

Посмотреть как зависит один параметр от другого:

```
plt.scatter(val_df["TOTAL UNITS"], val_df["SALE PRICE"])
plt.xlabel("total units")
plt.ylabel("sale price")
# val_df["SALE PRICE"].describe()
```

И если значения слишком большие, то можно перейти в логарифмическую шкалу:

```
val_df = val_df.query("`TOTAL UNITS` > 10")
plt.scatter(val_df["TOTAL UNITS"], val_df["SALE PRICE"])
plt.xlabel("total units")
plt.ylabel("sale price")
plt.yscale("log")
plt.xscale("log")
plt.axis([10, 10**4, 1, 10**10])
```

Переведем из графика в гистограмму:

```
nhoods = {key: df["TOTAL UNITS"][df['BOROUGH'] == key].mean() for key in df["BORO
UGH"].unique()}
nhoods
fig  = plt.figure()
fig = plt.bar(nhoods.keys(), nhoods.values())
```

Занятие 4. Лекция 2. Химические данные. Способы хранения химической информации.

Что из себя представляют данные в химии?

Основной объект - химическая структура (редко: спектры, электронная структура), различные размеры, не машиночитаемый "как есть". Нам нужно преобразовать его. Кроме того, обычно имеет нестандартные форматы.

Что обычно мы имеем:

1) Лабораторный журнал
2) Данные с приборов (с разными форматами)
3) То, что мы публикуем (успешные эксперименты)
4) То, что не публикуем (Fails)

Data Lake (озеро данных). Озеро данных – это как бы работали электронные журналы, если бы заполнялись единообразно, были бы в электронном виде и все имели бы доступ к ним:

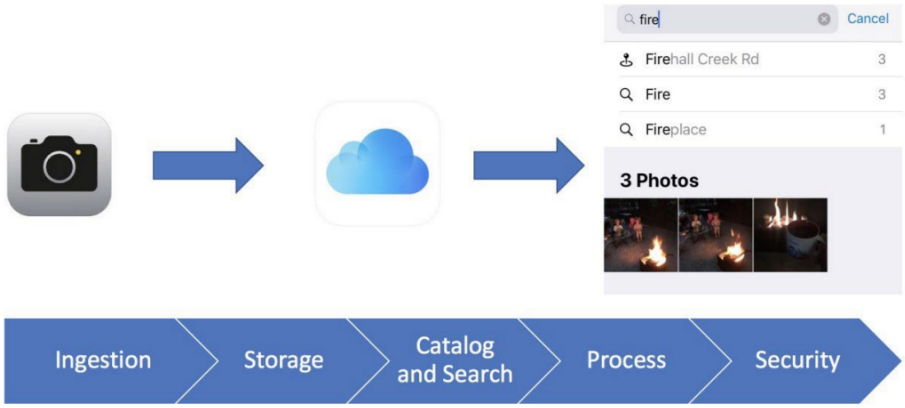


Рисунок 4.1. Как работает data lake на примере работы телефона

В первую очередь озеро данных – это большое количество данных, для которого известна процедура добавления данных, каталогизация и поиска по хранилищу. Если встроенные средства для обработки, и чаще всего есть средства для защиты. Но в реальности такие базы крайне редки.

Та структура данных, которую мы получили в поиске данных должна лежать в основе структуры данных (см. лекцию 1).

Curated data:

Очистка (нет дубликатов, неполных данных, банальных ошибок)

Оценка качества

Аннотация и т.д.

Кроме того, необходимо делать labelled data: для понимания того, с чем мы работаем. Например, простые лейблы химических данных все видели на банках с реактивами (не есть, не курить возле, смертельно и т.д.). Данный процесс может использоваться для машинного обучения.

Суммируя:

FAIR data или это можно расшифровать так:

F – Findable (обнаружимый) – данные и материалы, обогащены с метаданными, которым присвоен уникальный идентификатор.

A – Accessible (доступный) – данные и метаданные хранятся в надежном хранилище с открытым и бесплатный протоколом. Доступный для машин и людей.

I – Interoperable (совместимый) – используя словари и онтологии общественного достояния, на метаданные можно ссылаться и связывать их.

M – Reusable (многоразовый) – Дополнительная документация и протоколы, описывающие получение данных, лицензированные с подробным указание происхождения.

Необходимо отметить *разницу* между машиночитаемым кодом и человекочитаемыми данными:

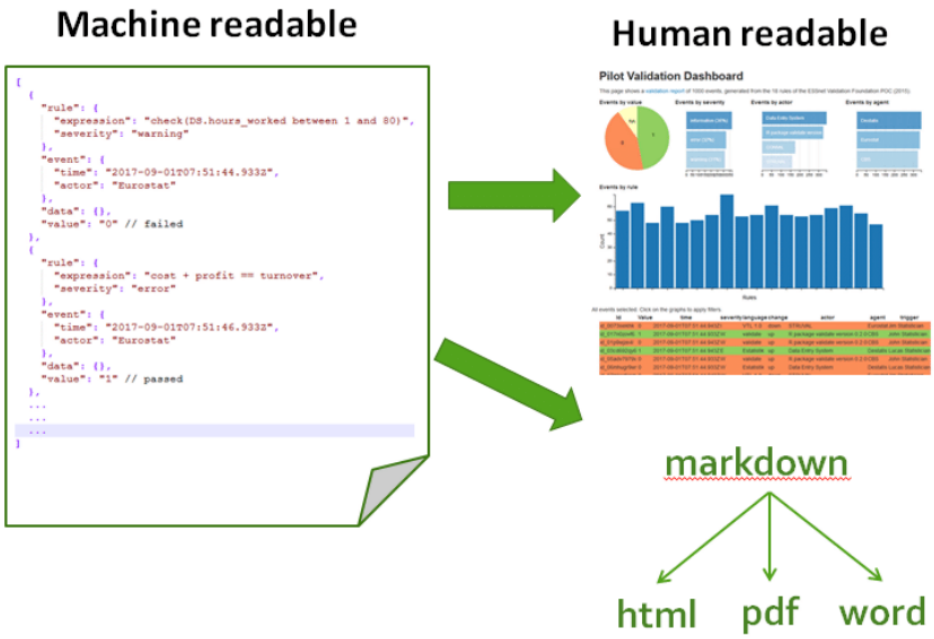
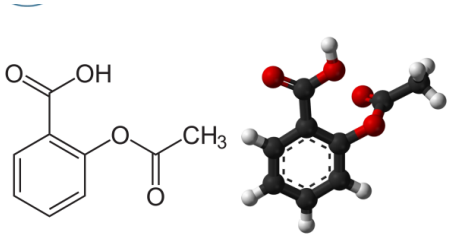


Рисунок 4.2. Разница между машиночитаемыми и человекочитаемыми данными.

Человеку удобно воспринимать графики, таблицы и т.д., но данные в компьютере хранятся именно в html формате. Поэтому перевод из html формата и наоборот, крайне важная функция. Перед нами стоит задача как перевести химические данные в массив данных.

Форматы

Начнем с примера. Возьмем ацетилсалициловую кислоту и представим в виде машиночитаемых данных. У аспирина множество названии помимо номенклатуры ИЮПАКА, однако это может внести некую путаницу. Мы можем представить формулу вещества в двумерном изображении (структура). Кроме того, можно представить в виде молекулярного графа (матрица смежности):



2-ацетилоксибензойная кислота,
ацетилсалициловая кислота, C9H8O4

	C	C	O	O	C	C	C	C	C	C	C	O	O
C	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C	1.0	0.0	2.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
O	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
O	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	0.0	1.0	0.0	1.5	0.0	0.0	0.0	1.5	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	1.5	0.0	1.5	0.0	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	0.0	1.5	0.0	1.5	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	0.0	0.0	1.5	0.0	1.5	0.0	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.5	0.0	1.5	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	1.5	0.0	0.0	0.0	1.5	0.0	1.0	0.0	0.0
C	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	2.0	1.0
O	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
O	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0

Рисунок 4.3. Представление химической формулы в массиве

Такое представление (молекулярный граф) довольно легко генерируем из двумерной структуры и удобен в использовании.

Самые простые форматы:

формат.xyz - запись координат каждого атома (простой вариант записи):

```
14
Comments line
C          -1.2540710000          0.3888570000         -0.1001430000
O          -1.7850710000          1.7148570000         -0.0311420000
C           0.2339290000          0.4098570000          0.2938570000
O           0.9539290000          1.2578570000         -0.6041420000
C           0.8059290000         -1.0181430000          0.2188580000
O           2.1879280000         -0.9981430000          0.5848580000
H          -1.3540710000          0.0168570000         -1.0991420000
H          -1.7930710000         -0.2461430000          0.5718580000
H          -2.7130710000          1.7018570000         -0.2771420000
H           0.3339290000          0.7818570000          1.2918580000
H           0.8649290000          0.9248570000         -1.5001420000
H           0.2679290000         -1.6541430000          0.8908580000
H           0.7069290000         -1.3911430000         -0.7791430000
H           2.5439290000         -1.8891430000          0.5388580000
```

Рисунок 4.4. Представление глицерина в формате. Xyz

Внутри себя программа обычно преобразует декартово представление в Z-матрикс (Internal). Это уже позволяет отобразить химический смысл, однако это сложно построить (но много способов задать):

N	Symbol	R	Angle	Dihedral	NR	NA	ND
1	C						
2	O	RO2			1		
3	C	RC3	AC3		2	1	
4	O	RO4	AO4	DO4	3	2	1
5	C	RC5	AC5	DC5	4	3	2
6	O	RO6	AO6	DO6	5	4	3
7	H	RH7	AH7	DH7	6	5	4
8	H	RH8	AH8	DH8	7	6	5
9	H	RH9	AH9	DH9	8	7	6
10	H	RH10	AH10	DH10	9	8	7
11	H	RH11	AH11	DH11	10	9	8
12	H	RH12	AH12	DH12	11	10	9
13	H	RH13	AH13	DH13	12	11	10
14	H	RH14	AH14	DH14	13	12	11

Рисунок 4.5. Z-matrix.

Формат.mol

Чаще всего используемый формат в машинном обучении. Неплохой баланс между человекочитаемые и машиночитаемые форматами. Чаще всего в .mol отсутствуют водороды. Кроме координат в файл входит название, количество атомов, связей, координаты элементов, дополнительные параметры. Выглядит так:

```
Benzene, ID: C001
  SJC    20160623 1   1.00000     0.00000
Example Benzene mol file.
  6  6  0     0  0              1 V2000
   -1.3961    0.0013   -0.0504 C   0  0  0  0  0  0           0  0  0
   -0.7402   -0.3516    1.1313 C   0  0  0  0  0  0           0  0  0
    0.6556    -0.344    1.1844 C   0  0  0  0  0  0           0  0  0
    1.3956    0.0123    0.0546 C   0  0  0  0  0  0           0  0  0
    0.7398    0.3611   -1.1284 C   0  0  0  0  0  0           0  0  0
    -0.656    0.3577   -1.1803 C   0  0  0  0  0  0           0  0  0
  2  1  2  0     0  0
  1  3  1  0     0  0
  4  2  1  0     0  0
  3  6  2  0     0  0
  5  4  2  0     0  0
  6  5  1  0     0  0
M  END
```

Рисунок 4.6. Формат. mol для бензола

Формат .sdf

Крайне похожий на .mol. Отличается наличием тегов и лейблов в файле. \

Формат SMILES

Игнорируем водороды (если это не критично), выбираем самую длинную цепочку, разрываем циклические фрагменты, записываем (-, =, #, : или не записываем) связи, записываем боковые цепочки в скобках.

Но: С какого атома начать? Ароматические системы?

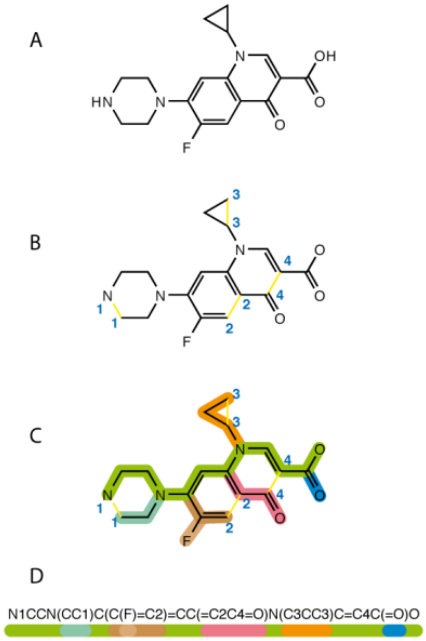


Рисунок 4.7. Перевод из двумерной структуры в SMILES

Данные представления молекул (в строке) были в различных вариантах:

- InChi
- SMARTS
- SYBYL
- SELFIES ….

Форматы для данных о твердых телах:

Формат .cif

Формат записи кристаллической структуры, который разбит на блоки, в которых идет перебор каких-то параметров с ключами, параметрах ячейки, R-фактор и т.д.:

```
data_1000041
loop_
_publ_author_name
'Abrahams, S C'
'Bernstein, J L'
_publ_section_title                 Accuracy of an automatic diffractometer. ...
_journal_coden_ASTM                 ACCRA9
_journal_name_full                  'Acta Crystallographica (1,1948-23,1967)'
_journal_page_first                 926
_journal_page_last                  932
_journal_paper_doi                  10.1107/S0365110X65002244
_journal_volume                     18
_journal_year                       1965
_chemical_formula_structural        'Na Cl'
_chemical_formula_sum               'Cl Na'
_chemical_name_systematic           'Sodium chloride'
_space_group_IT_number              225
_symmetry_cell_setting              cubic
_symmetry_Int_Tables_number         225
_symmetry_space_group_name_Hall     '-F 4 2 3'
_symmetry_space_group_name_H-M      'F m -3 m'
_cell_angle_alpha                   90
_cell_angle_beta                    90
_cell_angle_gamma                   90
_cell_formula_units_Z               4
_cell_length_a                      5.62
_cell_length_b                      5.62
_cell_length_c                      5.62
_cell_volume                        177.5
_refine_ls_R_factor_all             0.022
_cod_database_code                  1000041
loop_
_symmetry_equiv_pos_as_xyz
x,y,z
y,z,x
...
1/2+z,y,1/2-x
1/2+z,1/2+y,-x
loop_
_atom_site_label
_atom_site_type_symbol
_atom_site_symmetry_multiplicity
_atom_site_Wyckoff_symbol
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
_atom_site_occupancy
_atom_site_attached_hydrogens
_atom_site_calc_flag
Na1 Na1+ 4 a 0. 0. 0. 1. 0 d
Cl1 Cl1- 4 b 0.5 0.5 0.5 1. 0 d
loop_
_atom_type_symbol
_atom_type_oxidation_number
Na1+ 1.000
Cl1- -1.000
```

Рисунок 4.8. Пример формата .cif

Данные для высокомолекулярных соединений:

Формат .pdb (для белков и нуклеиновых кислот):

```
HEADER    EXTRACELLULAR MATRIX                    22-JAN-98   1A3I
TITLE     X-RAY CRYSTALLOGRAPHIC DETERMINATION OF A COLLAGEN-LIKE
TITLE    2 PEPTIDE WITH THE REPEATING SEQUENCE (PRO-PRO-GLY)
...
EXPDTA    X-RAY DIFFRACTION
AUTHOR    R.Z.KRAMER,L.VITAGLIANO,J.BELLA,R.BERISIO,L.MAZZARELLA,
AUTHOR   2 B.BRODSKY,A.ZAGARI,H.M.BERMAN
...
REMARK 350 BIOMOLECULE: 1
REMARK 350 APPLY THE FOLLOWING TO CHAINS: A, B, C
REMARK 350   BIOMT1   1  1.000000  0.000000  0.000000        0.00000
REMARK 350   BIOMT2   1  0.000000  1.000000  0.000000        0.00000
...
SEQRES   1 A    9  PRO PRO GLY PRO PRO GLY PRO PRO GLY
SEQRES   1 B    6  PRO PRO GLY PRO PRO GLY
SEQRES   1 C    6  PRO PRO GLY PRO PRO GLY
...
ATOM      1  N   PRO A   1       8.316  21.206  21.530  1.00 17.44           N
ATOM      2  CA  PRO A   1       7.608  20.729  20.336  1.00 17.44           C
ATOM      3  C   PRO A   1       8.487  20.707  19.092  1.00 17.44           C
ATOM      4  O   PRO A   1       9.466  21.457  19.005  1.00 17.44           O
ATOM      5  CB  PRO A   1       6.460  21.723  20.211  1.00 22.26           C
...
HETATM  130  C   ACY   401       3.682  22.541  11.236  1.00 21.19           C
HETATM  131  O   ACY   401       2.807  23.097  10.553  1.00 21.19           O
HETATM  132  OXT ACY   401       4.306  23.101  12.291  1.00 21.19           O
```

Рисунок 4.9. Пример формата .pdb

Базы данных

1. **PubChem –** база данных органических соединений, с огромным объемом. К сожалению, в ней существуют повторные записи, недостающие данные и т.д. На текущий момент в ней 110 млн. соединений.
2. **ChEMBL –** база данных, которая ориентируется на биологически активные соединения. Кроме того включает большое количество свойств помимо целевых активностей.

3. **ZINC53 –** интересна тем, что основной мотив для включения в базу данных – коммерческая доступность.
4. **PDB** – база данных для структур белков. Существуют уже разрешенные комплексы белок/лиганд.
5. **ICSD –** база данные неорганических соединений, структуры
6. **CSD –** база данных просто кристаллических структур
7. **Crystallography Open Database –** структуры, можно полностью сказать.

“Хорошая” база данных:

1) Большая (иногда не всегда хорошо)
2) Обработано
3) Обычные форматы, экспортируемые
4) Готовность к вычислению
5) С маркировками
6) Бесплатная

Занятие 5. Семинар. Молекулярные дескрипторы

Остановимся на теме представления малых органических молекул в комп. читаемом виде. Стандартно используется библиотека rdkit. Установим ее:

pip install rdkit-pypi

Обсудим для чего нужны молекулярные дескрипторы. Прежде всего, чтобы компьютер понимал о каких структурах идет речь:

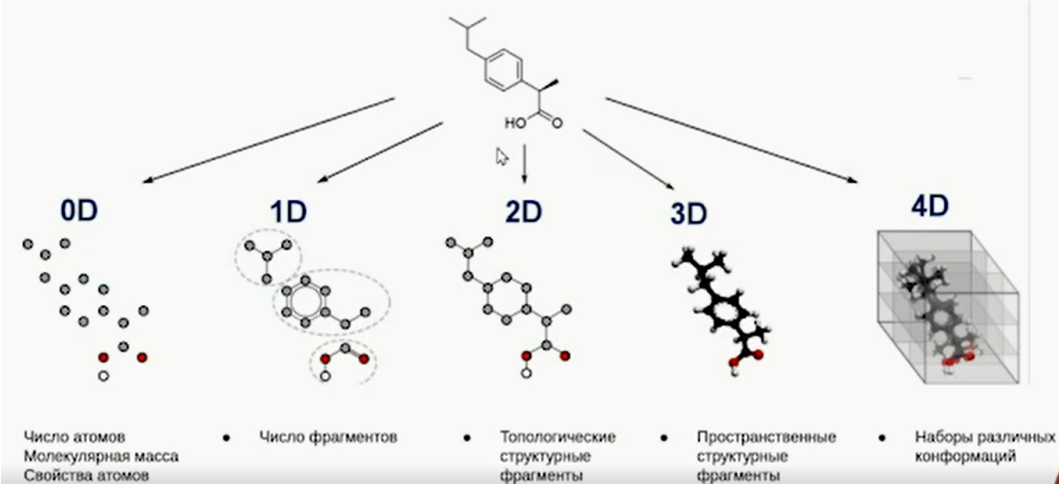


Рисунок 5.1. Пример перевода органической формулы в дескриптор

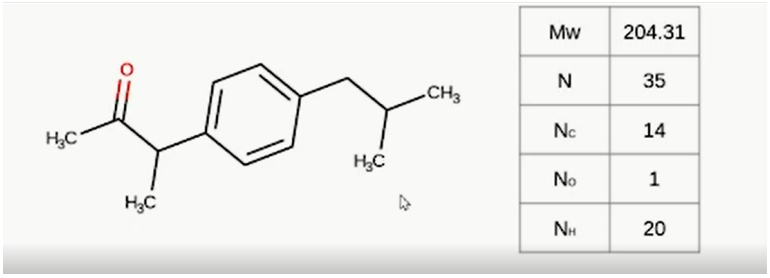
Mol file sample
RDKit 20
5 4 0 0 0 0 0 0 0 0 0999 V2000
0.9937 2.3657 0.0000 C 0 0 0 0 0 0 0 0 0 0
1.7437 1.0060 0.0000 O 0 0 0 0 0 0 0 0 0 0
0.9937 -0.2922 0.0000 C 0 0 0 0 0 0 0 0 0 0
1.7437 -1.5913 0.0000 N 0 0 0 0 0 0 0 0 0 0
-0.5062 -0.2922 0.0000 O 0 0 0 0 0 0 0 0 0 0
1 2 1 0
2 3 1 0
3 4 1 0
3 5 2 0
M END

Используем пакет SMILES
SMILES sample
COC(N)=O
Ароматика в Smiles
CC(C)CC1=CC=C(C=C1)C(C)C(O)=O

Экспериментальные дескрипторы

LogP, Молекулярная рефракция, Дипольный момент, Поляризуемость

OD -десприпторы



Mw	204.31
N	35
Nc	14
No	1
NH	20

Рисунок 5.2. Пример одномерного дескриптора для органической молекулы

```
def mol_vector(molecule: str):
COC(N)=O
pass
```

1D-Десприпторы

Данная форма представления химических соединений куда более интересная и широко используемая. В данном случае в качестве битов, составляющих вектор, каждое число представляет не только атом, но и подструктуру (из банка структурных фрагментов). Наиболее используемая библиотека подструктур: Maccs keys.

Разберем пример:

```
import rdkit
import numpy as np
from rdkit.Chem import MACCSkeys, AllChem, DataStructs
from rdkit.Avalon.pyAvalonTools import GetAvalonFP
arr = np.zeros((l,), dtype=int)
mols = rdkit.Chem.SDMolSupplier (“/content/drive/MyDrive/ML in chemistry/6_seminar/mol.mol’’)
mol = mols [0]
# descr - GctAvalonFP(mol)
# descr - AllChem.GetHashedAtomPairFingerprint(mol)
descr - AllChem.GetHAshedAtomFingerprint(mol)
print(rdkit.Chem.MolToSmiles(mol))
print(descr)
DataStruct.ConvertToNumpyArray(descr, arr)
print(arr)
```

Получаем представление в формате Smiles

2D-Дескрипторы

Extended-Connectivity Fingerprints (ECFPs)

Рисунок 5.3. Пример 2D-дескриптора

2D-дескрипторы одни из самых широко используемых, в плане представления, дескрипторов. Давайте разберем примитивный морgановский фингерпринт:

```
from rdkit import Chem
def get_daylight_rule(atom):
return (len(atoe GetNeighbors()),
        atom.Get Total Valence() - atom.GetTotalNumHs(),
        atom.GetAtomicNum(),
        int(atom.Get Mass()),
        atom.GetFormalCharge(),
        atom.GetTotalNumHs(),
        int(atom.IsInRing())
# Отличие ECFP от обычного daylight списка
```

Рассмотрим также функцию хеширования (перевод объекта в чисто (int)):

```
def make_identifier(props):
return hash(props)
```

Введем словарь:

```
bond_rename = {Chem. rdchem.BondType. SINGLE: 1,
    Chem.rdchem.BondType.DOUBLE: 2,
    Chem. rdchem.BondType. TRIPLE: 3,
    Chem. rdchem. BondType. AROMATIC: 4}
```

Инициализируем пустой массив:

```
identifiers = np.empty(0)
fp_length = 1024
```

Рассмотрим все на примере:

```
mol = Chem.MolFromSmiles("COC(N)=O")
atoms = mol. GetAtoms()
for atom in atoms:
    idfr = make_identifier(get_daylight_rule(atom))
    identifiers = np.append(identifiers, idfr)
    atom.SetProp("identifier", str(idfr))
type(atoms[O])
rdkit.Chem.rdchem.Atom
help(rdkit.Chem.rdchem.Atom.GetTotalNumHs)
help(rdkit.Chem.rdchem.Atom.GetProp)
```

Данным кодом получили информацию о конкретном атоме, но нам еще интересна информация об окружении атомов:

```
for atom in atoms:
  id_tuple ="[1, int(atom.GetProp("identifier"))]
  atom_neighbors = atom.GetNeighbors()
  bonds_to_neighbors = atom.GetBonds()
for a, b in zip(atom_neighbors, bonds_to_neighbors):
id_tuple.append(bond_rename[b.GetBondType()])
```

```
        id_tuple.append(a.GetProp("identifier"))
        id_tuple = tuple(id_tuple)
        idfr = make_identifier(id_tuple)
        identifiers = np. append(identifiers, idfr)
        atom.SetProp("identifier", str(idfr))
        identifiers = np. unique(identifiers)
```

Как получить дескриптор (вектор) длинной 1024:

```
fp = np.zeros(fp_length)
remainders = identifiers % fp_length
        for val in remainders:
        fp[int(val)] = 1
        print(fp)
```

Тут возникает проблема: что есть будет одинаковый остаток для двух разных identifiers (проблема Bit collision). Для этого как раз увеличивают вектор до 1024, чтобы уменьшить вероятность Bit collision.

Как посчитать fp в rdkit (используют метод Моргана):

```
ecfp_fp = AllChem.GetMorganFingerprint(mol, 6, useFeatures=False)
        from rdkit.Chem import Draw
        bi = {}
        fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, bitInfo=bi)
```

Functional-Class Fingerprints (Ffp) имеют свои правила:

1. hydrogen bond acceptor
2. hydrogen bond donor
3. negatively ionizable
4. positively ionizable
5. aromatic
6. halogen

Кроме того, под данные правила могут попадать атомы разной природы. И это было сделано сознательно. Тем самым максимально обобщили информацию об атомах. Расчет у них аналогичен.

3D/4D-дескрипторы. Данные дескрипторы используют информацию о 3D структуре молекулы, а 4D о конформациях. Их существует великое множество. А считаются они аналогично 2 D-дескрипторам.

Область применимости

Допустим мы натренировали какую-либо модель, используя определенный набор молекул для тренировки (допустим 1000 молекул размером до 9 атомов). Что будет если такой системе дать молекулу, которая в этом наборе не была представлена. Это и есть вопрос области применимости. До какого момента наша область применимости будет работать хорошо? Для подсчета ОП существует множество методов. Рассмотрим *метод ближайших соседей*. Каждый объект представлен вектором определенной длины. И мы можем посчитать расстояние от одного вектора до другого. И сделать это для каждого из

векторов. Затем найти среднее расстояние до k ближайших векторов. Теперь мы хотим понять можно ли использовать модель для других векторов на этом наборе данных. Проведем аналогичную процедуру. Если полученное расстояние будет меньше, чем среднее – то можно использовать модель. Если нет – нельзя.

Еще есть способы определения того, что вектор подходит для модели: Kernel Density и расстояние Махаланобиса. Данные методы дают хорошую оценку.

Занятие 6. Семинар. Классические алгоритмы машинного обучения.

Типы машинного обучения. Существуют 3 больших типа: Supervised (на основе размеченных данных), Unsupervised (на основе неразмеченных данных), Reinforcement (обучение модели из некоторых попыток предсказания значения). В данном семинаре речь пойдет только о первой группе.

Классификация и регрессия. Модель Supervised также можно разделить на две группы: задачу классификации и регрессии. Во время построения модели для классификации, наша цель определить является ли конкретный набор признаков, относящийся к одному из параметров. А в регрессионной задаче цель найти конкретное число, которое соответствует таргету, который мы ищем:

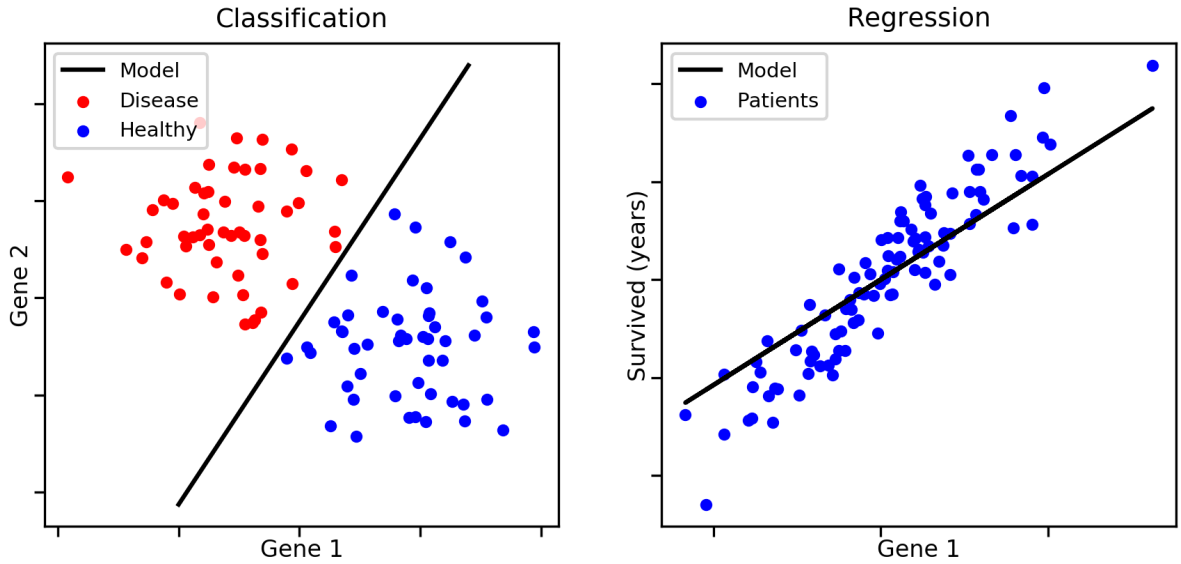


Рисунок 6.1. Классификация и регрессия в модели Supervised.

Оба вида моделей имеют набор метрик (оценка функции, которая получилась):

Регрессионные метрики

MAE – Mean Absolute Error

$$MAE = \frac{\sum_i^n |y_i - f_i|}{n} \tag{6.1}$$

MSE – Mean Squared Error

$$MSE = \frac{\sum_i^n |y_i - f_i|^2}{n} \tag{6.2}$$

R^2 – Coefficient of Determination

$$R^2 = 1 - \frac{\sum_i^n |y_i - f_i|^2}{\sum_i^n |y_i - f_i|} \tag{6.3}$$

Классификационные метрики

True Positive (TP)
True Negative (TN)
False Positive (FP)
False Negative (FN)

Accuracy:

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \quad (6.4)$$

Precision aka Positive Value (PPV) aka Specitivity:

$$Precision = \frac{TP}{TP+FP} \quad (6.5)$$

Recall aka Sensitivity:

$$Recall = \frac{TP}{TP+FN} \quad (6.6)$$

$F_1 - score$

$$F_1 = \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} \quad (6.7)$$

MCC – Matthews Correlation Coefficient

$$MCC = \frac{(TP*TN-FP*FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (6.8)$$

CM -Confusion Matrix

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix} \quad (6.9)$$

Алгоритмы решающий деревьев

Перейдем непосредственно к алгоритмам машинного обучения. Первым будет разобран алгоритм, решающий деревьев. Пример можно видеть на картинке:

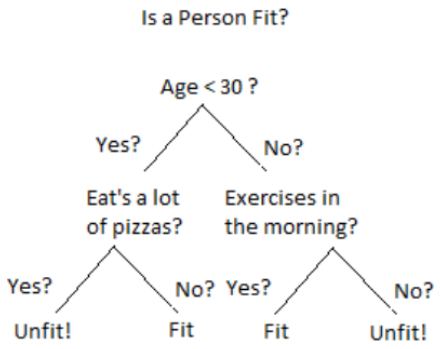


Рисунок 6.2. Пример алгоритма решающих деревьев.

Суть модели проста: построить дерево решений (смотри выше). Какие определенные функции по которому можно распределить на две составляющие ветвь:

Энтропия $Entropy = -\sum_i^n p_i * logpi$

Индекс Джини $Gini = 1 - \sum_i^n p_i^2$

Попробуем построить дерево решений по данным о погоде:

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
import numpy as np
weather = pd.read_csv("/content/drive/hyDrive/ML in chemistry/7_seminar/weather_ds.csv")
weather
```

```
# Имплементируем функцию по расчету индекса Джини
def gini_index(x_arr, target_arr):
# Найдем все уникальные значения в x_arr
states = np. unique(x_arr)
# зададим начальные значения для массива состояний и индекса джини
state_indexes = []
gini = 0
# для каждого возможного значения проверяемого массива
for state in states:
# Найдем все значения целевого массива, которые соответствуют проверяемому
state_target = target_arr[x_arr == state]
# Определим количество нулей и единиц для этого случая
ones = state_target[state_target == "yes"]
zeros = state_target[state_target == "no"]
state_indexes.append((len(state_target), 1 - (len(ones) / len(state_target)) ** 2 - (len(zeros) /
len(state_target)) ** 2))
# найдем сумму индекса джини для всех значений проверяемого массива
for tuple in state_indexes:
gini += tuple[0] *tuple[1]/len(target_arr)
return gini
# чем больше индекс, тем больше разнообразие получаемой выборки
print(f ' Outlook gini is {gini index(weather["outlook"], weather["play"])}")
print(f ' temperature gini is {gini index(weather["temperature"], weather["play"])}")
print(f ' Humidity gini is {gini_index(weather["humidity"], weather["play"])}")
print(f ' windy gini is (gini_index(weather["windy"], weather["play"])}")
```

Результат:

Outlook gini is 0.34285714285714286
Temperature gini is 0.44647619047619047
Humidity gini is 0.3673469387755103
Windy gini is 0.42857142857142855

Линейная и логистическая регрессия

Попробуем написать функции с нуля. Но они будут отличатся от классических регрессии, которые используют обычно. Классическая линейная регрессия строится по алгоритму метода наименьших квадратов. Но в данном семинаре будет использован метод градиентного спуска, так как можно будет легко перейти к нейронным сетям.

```
#Подключим датасеты
from sklearn.datasets import load_iris, load_diabetes
# Подключим необходимые метрики
from sklearn.metrics import fi_score, r2_score, accuracy_score, confusion_matrix
import numpy as np
from math import sqrt
```

```
# Определим функции активации
def sigmoid(z):
return 1 / (1 + np.exp(-z))
def identity(z):
return z
# Определим функции ошибок
def log_loss(y, y_hat):
        loss = -np.mean(y"(np.log(y_hat)) - (1-y)"np.log(1-y_hat))
        return loss
def nase(y, f):
    return sqrt((1 / len(y)) * p.sum((y - f) ** 2))
```

Получаем:

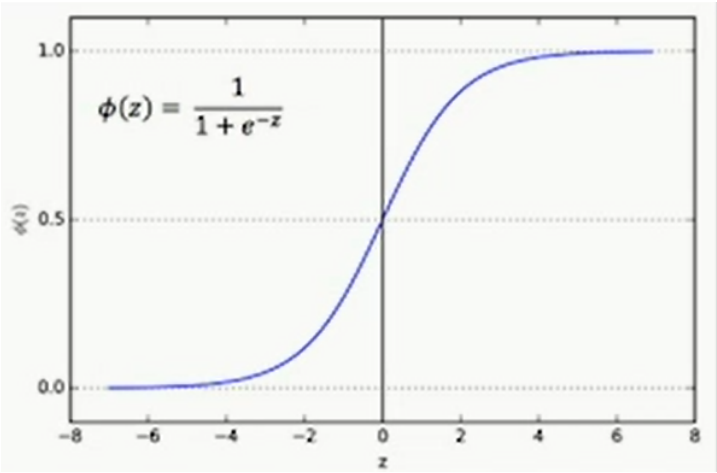


Рисунок 6.3. Функция для линейной регрессии

```
def gradients(x, 55 y hat):
# x --> --> входные данные.
# y--> целевой вектор.
# y_hat --> предсказанные значения целевого вектора.
m = X.shape[0]
# Подсчет градиентов весов и байеса
dw = (1/m)*np.dot(X.T, (y_hat - y))
db = (1/m)*np.sum((y_hat - y))
return dw, db
def normalize(X):
m, n= X.shape
for i in range(n):
X = (x - X.mean(axis=0))/x.std(axis=0)
return X
def train(X, y, bs, epochs, lr, loss=low_loss, act_f=sigmoid):
# bs --> размер батча.
# epochs --> количество итераций.
```

```
# lr --> скорость обучения.
m, n = X.shape
# Задаем начальные значения для весов и байеса
w = np.zeros((n,1))
b = 0
# Приведем данные к нужной форме
y = y.reshape(m,1)
# Нормализируем фичи
×= normalize(X)
# Лист для записи значений функции ошибки
losses = []
for epoch in range(epochs):
for i in range((m-1)//bs + 1):
# Получим данные одного батча
start i = i*bs
end_i = start + bs
xb = x[start_i:end_i]
yb = y[start_i:end_i]
# Расчет значений целевого вектора
y_hat = act_f(np.dot(x b, w) + b)
# Получим градиенты
dw, db = gradients(xb, yb, y_hat)
w-= Ir*da
b -= Ir*db
# Подсчет значения функции ошибки
1 - loss(y, act_f(np.dot(x, w) + b))
losses.append(1)
return w, b, losses
def predict(x, w, b, mode "class"):
× = normalize(X)
# В зависимости от типа задачи приведем выведем значения в нужной форм
if mode == "class":
preds = sigmoid(np.dot(X, w) + b)
pred_class = [1 if i > 0.5 else 0 for i in press]
return np.array(pred_class)
if mode = "reg":
return np.dot(x, w) + b
```

Применим функцию к набору данных:

```
dataset = load_iris(as_frame=True, return_X_y=True)
× - dataset[o][dataset[1] <= 1).to_numpy()
y = dataset[1][dataset[1] <= 1].to_mumpy()
w, b, l = train(x, y, bs=100, epochs=1000, lr=0.01)
```

```
prediction = predict(x, w, b)
print(fi_score(s, prediction))
print(accuracy_score(y, prediction))
print(confusion_matrix(y, prediction))
```

На выходе получаем хорошо модель на представленном наборе данных:

```
1.0
1.0
[[50 0]
[0 50]]
```

Однако, если взять более сложный набор данных, например, диабетиков:

```
dataset = load_diabetes(as_frame=True, return_X_y=True)
X = набор данных[0].to_numpy()
y = набор данных[1].to_numpy()
w, b, l = поезд(x, y, bs=100, эпохи=10000, lr=0.01, потеря =nase, act_f=идентичность)
print(r2_score(y, predict(x, w, b, mode="reg")))
# Выход:
 0,49575640502850626
```

Мы получили крайне маленькое значение R^2, что говорит о том, что модель не обучилась. В чем причина? Возможно, функция написана с ошибкой. Для сравнения можно взять такую же модель из skllearn:

```
from sklearn. linear_model import LinearRegression
model = Linear Regression()
model.fit(x, y)
print(r2_score(y, model. predict(x)))
# Выход
 0.5177494254132934
```

Отличие от предыдущего значения небольшие, следовательно проблема в наборе данных. И линейная регрессия на таких наборах хорошо работать не будет.

Метод ближайших соседей

Метод довольно простой и понятный (именно алгоритм), при этом довольно мощный. В чем его суть:

1. Выбор количества соседей К
2. Расчёт попарного расстояния между всеми точками
3. Сортировка полученных значений
4. Выбор К соседей из отсортированного списка

$$d = \sqrt{\sum_{i}^{N} (p_i - q_i)^2} \qquad (6.10)$$

Приступим к написанию кода:

```
from sklearn.neighbors import kweighborsclassifier, kweighborstegressor
```

knn_model = Kneighborsclassifier(n_neighbors=10)

Байесовский формализм

Напомню Теорема Байеса: $P(A|B) = \frac{P(B|A)\,(A)}{P(B)}$

Код:

```
from sklearn.naive_bayes import GaussianNB
gnb = Gaussian()
```

Данный алгоритм имеет очевидный плюс: крайне быстрый. Сам по себе расчет условный вероятностей также быстрый. Плюс алгоритм хорошо работает при малом количестве признаков.

Ансамблевые алгоритмы и случайный лес

Алгоритм разрешающих деревьев нельзя использовать в сложных задачах. В этом нам на помощь приходят ансамблевые алгоритмы. Суть ансамблевых алгоритмов в том, что мы строим множество моделей дерева решений. И в качестве результата берем среднее значение того, что получилось.

Код:

```
from sklearn.ensemble import Random For Loading…ier, RandomForestRegressor
from sklearn.ensemble import BaggingRegressor, Bagging Classifier
rf_model = RandomForestclassifier(n_estimators=100,
criterion="gini",
max depth None)
```

Занятие 7. Лекция 3. Нейронные сети. Принципы работы.

Использование готовых программных пакетов, предназначенных для обучения нейронных сетей. Для полноценной работы необходимо знать, как работают базовые нейронные сети:

Рисунок 7. 1. Схема структуры нейронных сетей.

Существует несколько типов машинного обучения:

1) С учителем (классификация и регрессия)
2) Без учителя
3) Обучение с подкреплением

Сформулируем в общем виде задачу, которую мы будем решать с помощью искусственного интеллекта:

Задача регрессии: $f: R^n \to R$ (сопоставление некому многомерному вектору вещественных чисел скалярного значения). На практике мы имеем некую выборку объектов, в каждом из которых ставится соответствие его признаку в описание. На основании этого признака, мы хотим иметь возможность предсказывать значения интересующего нас свойства.

Решение. Линейная регрессия: $\hat{y} = \sum_{i=1}^{N} w_i x_i + b$, через функцию активации переходим:

$$\hat{y} = h\left(\sum_{i=1}^{N} w_i x_i + b\right) \xrightarrow{x_0 = 1} h\left(\sum_{i=0}^{N} w_i x_i\right) \tag{7.1}$$

Искусственный нейрон

Для сети нейронов нужно сделать обобщение. На вход мы предполагаем признаковое описание объекта. В случае нейронов, мы на вход можем предполагать не только признаковое описание, но и сигналы с других нейронов. И это свойство можно использовать для создания нейронных сетей.

Полносвязная нейронная сети прямого распространения представляет собой слои из нейронов:

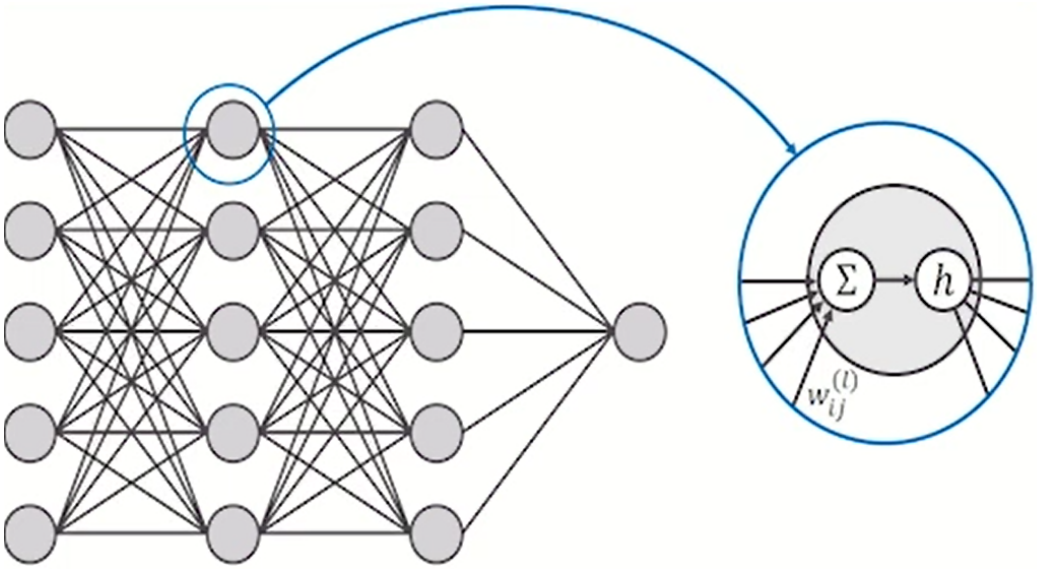


Рисунок 7.2. Полносвязная нейронная сети прямого распространения.

В один слой нейроны входят тогда, когда принимают сигналы от одного и того же множества нейроны и в свою очередь передают сигнала на другое, но одно и тоже множество нейронов. Этого вполне достаточно для описания данной сети. Имея в распоряжении аналитическую форму соответствующую одному искусственному нейрону, а также понимаю каким образом нейроны связаны между собой, мы может записать соответствующее аналитическое выражение для всей сети. В модели есть соответствующие весовые коэффициенты w_i , которые нам необходимо подобрать для полезной работы. Таким образом мы подошли к задаче обучения нейронной сети

Обучение нейронной сети

Обучение нейронной сети сводится к подбору весовых коэффициентов:

$$w^* = argmin\, L(f(x, w), y) \tag{7.2}$$

И данная задача в свою очередь сводится к минимизации некой функции потерь

$$\nabla_w L = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \dots \\ \frac{\partial L}{\partial w_n} \end{pmatrix} \tag{7.3}$$

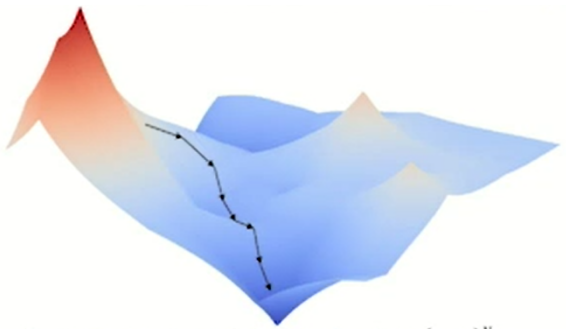


Рисунок 7.3. Поиск минимума функции потерь.

Алгоритм обратного распространения ошибки

Данный алгоритм позволяет рассчитать частные производные функции потерь по весам нейронной сети. На первом этапе (прямое распространение сигнала) начиная с первого слоя сети мы производим расчет взвешенных сумм (входные сигналы нейронов, весовые коэффициенты) и далее на суммы действуем некой функцией активации:

$$a_i = \sum_i^N w_i z_i\,; \quad z_i = h(a_i)$$

Данная операция выполняется для всех узлов нейронной сети. На втором этапе происходит расчет частных производных функции потерь:

$$\frac{\partial L_n}{\partial w_{ji}} = \frac{\partial L_n}{\partial a_j}\frac{\partial a_j}{dw_i} \quad \rightarrow \frac{\partial L_n}{\partial w_{ji}} = \delta_j z_i \tag{7.4}$$

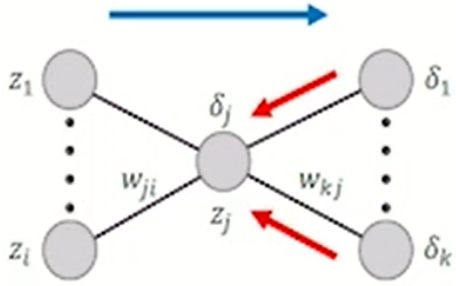


Рисунок 7.4. Алгоритм обратного распространения ошибок в схеме

Но в таком «сыром» виде использовать алгоритм крайне неудобно. Поэтому используют некие приемы.

Модификации метода градиентного спуска

Метод моментов. Предполагает учет инерциальности движения по поверхности (направление движения определяется не только значение градиента, но и тем в каком направлении двигался шарик за мгновение до).

Adagrad. Учет разной скорости сходимости частных производных их функции потерь.

Функции потерь

Средняя абсолютная ошибка (MAE)

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{7.5}$$

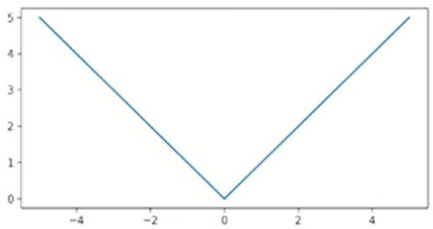


Рисунок 7.5. Средняя абсолютная ошибка

Среднеквадратичная ошибка (MSE)

$$MSE = \frac{1}{N\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \quad (7.6)$$

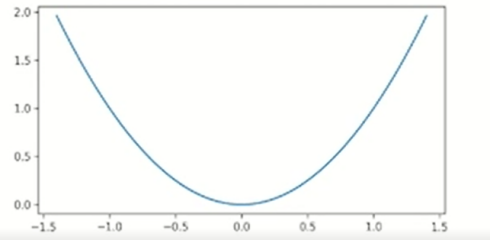


Рисунок 7.6. Среднеквадратичная ошибка

Дивергенция (расстояние) Кульбака-Лайблера

$KL(P||Q) = \sum p(y) \log \frac{p(y)}{q(y)} = \sum p(y) \log p(y) - \sum p(y) \log q(y) = H(p) + H(p,q)$:

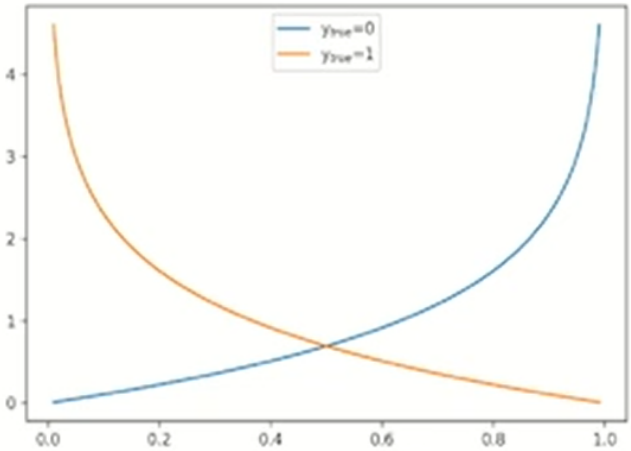


Рисунок 7.7. Дивергенция KL

Функция активации

Для того, чтобы функция могла быть использована в качестве функции активации к ней выдвигается ряд требований:

Возрастающая функция

На всем промежутке дифференцируема

Нелинейная функция

Примеры:

$$f(x) = \frac{1}{1+e^{-x}}$$

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f(x) = \max(o, x)$$

Ранняя остановка (контролируем качество работы модели на отложенной выборке):

Рисунок 7.8. Схема выборки при ранней остановке.

Обучение стоит остановить, если качество работы не улучшается на отложенном валидационном множестве на n-ное количество шагов.

Dropout. На каждом этапе работы градиентного спуска, мы отключаем n-ное количество нейронов.

Batch normalization. В ходе обучения нейронной сети происходит изменение весовых коэффициентов отдельных нейронов. В результате происходит изменение выходных сигналов. Для решения данной проблемы, мы нормализируем данные на конкретном множестве (рассчитать средние значения и на них нормализовать).

Обработка «табличных» данных

Полносвязные нейронные сети полностью игнорирует наличие внутренних структурных данных. В случаи химических данных структура имеет важное значение (особенно в структурных методах). Как это учесть? Привести 2 D массив к 1 D массиву. Но используют более продвинутую локальную сеть. Используют операцию свертки.

Операция свертки. Надо выделить локальную область и действует окном свертки:

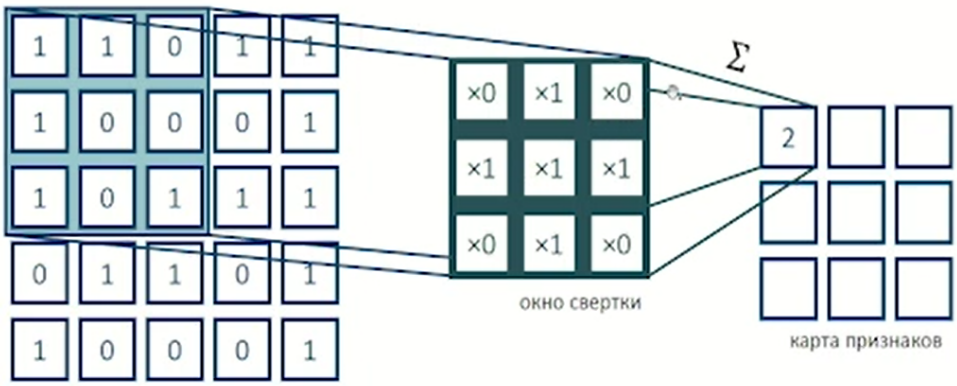


Рисунок 7.9. Операция свертки

Автокодировщики

В качестве выходного сигнала используется сигнал на вход. Используется модель «бутылочного горлышка»:

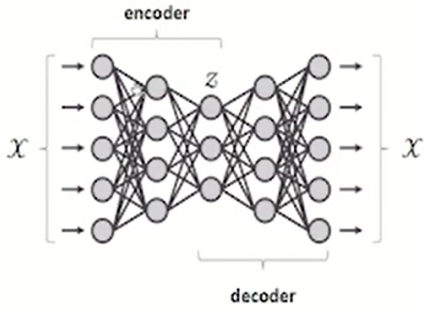


Рисунок 7.10. Схема автокодировщика.

Данная архитектура может быть использована для генерации новых объектов, обучению представлениям, снижение размерности, сжатию данных, шумоподавлению.

Занятие 8. Семинар. Нейронные сети.

На этом семинаре «заглянем» под «капот» нейронной сети и научимся пользоваться tenserflow для создания нейронных сетей.

Перед работой не забудьте загрузить данные из google.drive.

Нейронная сеть – это ряд слоев, состоящих из нодов, которые связанны между собой. В первую очередь подключим несколько функции:

```
from sklearn.datasets import make_moons
# make_moons – метод для теста данных
from sklearn.metrics import f1_score
import numpy as np
X, y = make_moons(200, noise=0.20)
num_examples = len(y) # training set size
nn_input_dim = X.shape[1] # input layer dimensionality
nn_output_dim = 2 # output layer dimensionality
# Gradient descent parameters
epsilon = 0.01 # learning rate for gradient descent
reg_lambda = 0.01 # regularization strength
# Перевод результата в вероятности вхождения в двух классах
def sigmoid(z):
return 1 / (1 + np.exp(-z))
# функция подсчета ошибки
def calculate_loss(model):
W1, b1, W2, b2 = model['Wi'], model['bi'], model['W2'], model['b2']
# получить выходные значения сети
z1 = X.dot(W1) +| b1
a1 = np.tanh(z1)
z2 = a1.dot(W2) + b2
probs = sigmoid(z2)
# Расчет значения ошибки
correct_log probs = -np.log(probs[range(num_examples), y])
data_loss = np.sum(correct_log probs)
# Добавление параметра регуляризации
data_loss += reg_lambda / 2 * (np.sum(np.square(W1)) + np.sum(np.square(W2)))
print(data_loss)
return 1. / num_examples * data_loss
```

Вспомогательная функция предикта:

```
def predict(model, x):
W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']
# Прямое распространение
z1 = x.dot(W1) + b1
a1 = np. tanh(z1)
```

```
z2 = a1.dot(W2) + b2
exp_scores = np.exp(z2)
probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
return np.argmax(probs, axis=1)
```

Функция расчета парметров нейронной сети

```
    - nn_hdim: Число нодов скрытых слоев
    - num_passes: Количество эпох
    - print_loss: Флаг для вывода значения ошибки
def build_model(nn_dim, num_passes=20000, print_loss=False):
# Инициализация весов сети:
np. random. seed(0)
    W1 = np.random.randn(n n_input_dim, nn_time) / np.sqrt(n n_input_dim)
b1 = np.zeros((1, nn_hdmi))
W2 = np.random.randn(n n_dim, nn_output_dim) / np.sqrt(n n_hdim)
b2 = np.zeros((1, nn_output_dim))
# Словарь параметров модели
model = {}
# Градиентный спуск
for i in range(0, num_passes):
# Прямое распространение
z1 = X.dot(W1) + b1
a1 = np.tanh(z1)
z2 = al.dot(W2) + b2
exp_scores = np.exp(z2)
probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
# Обратное распространение
delta3 = probs
delta3[range(num_examples), y] -= 1
dW2 = (a1.T).dot(delta3)
db2 = np.sum(delta 3, axis=0, keepdims=True)
delta2 = delta3.dot(W2.T) * (1 - np.power(a 1, 2))
dW1 = np.dot(X.T, delta2)
db1 = np.sum(delta 2, axis=0)
# Учет регуляризации
dW2 += reg_lambda * W2
dW1 += reg_lambda * W1
# Обновление значений весов
W1 += -epsilon + dW1
b1 += -epsilon * db1
W2 += -epsilon * dW2
b2 += -epsilon * db2
```

```
# Запись значений в словарь
model = {'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2}
if print_loss and i % 1000 == 0:
print("Loss after iteration %i: %f" % (i, calculate_loss(модель)))
return model
```

При запуске программы мы видим, как ошибка уменьшается при каждом прогоне. В итоге ошибка уменьшилась в 5 раз.

Tensorflow

Tensorflow – самый известный framework от Google. В данном семинаре будем использовать для решения задачи растворимости. Для начала загрузим необходимые библиотеки:

```
import tensorflow as tf
from rdkit import Chem
from rdkit.Chem import AllChem, Datastructs
print("TensorFlow version:", tf.__version__)
```

Загрузим файл с растворимостью из google.drive:

```
mols = Chem.SDMolSupplier("/content/drive/MyDrive/ML in chemistry/10_seminar/logS_dataset.sdf")
× = []
y = []
def to_cat(val):
if val == "true":
return 1
else:
return 0
def get_descr_value(molecule):
out_arr = np.zeros((1,), dtype=int)
descriptor = AllChem.GetMorganFingerprintAsBitVect(molecule, 6, 1024)
if isinstance(descriptor, DataStructs.cDataStructs.ExplicitBitVect):
Data Structs.Convert To Numpy Array(descriptor, out_arr)
return out_arr
return descriptor
for mol in mols:
y.append(to_cat(mol.GetProp("Soluble")))
x.append(get_descr_value(mol))
× = np.asarray(x)
y = p. asarray(y)
×_train, x_test, y_train, y_test = ×[:-100, :], x[-100:, :], y[:-100], y[-100:]
# Формирование модели
model = tf.keras.models.Sequential([
tf. keras.layers.Dense(1024, activation='relu'),
```

```
        tf.keras.layers.Dropout(0.2),
        tf. keras.layers.Dense(512),
        tf. keras.layers.Dropout(0.2), I
        tf. keras.layers.Dense(1)])
# Функция ошибки
 loss_fn = tf.keras.losses.Binary Cross entropy(from_logits=True)
# Компилирование модели
 model.compile(optimizer='sgd',
               loss=loss_fn,
               metrics=['accuracv'])
model.fit(x_train, y_train, epochs=5)
```

при запуске функции видно, что ошибка упала с 0.5 до 0.34, а accurancy возросло до 0.89. Если поставить epochs=10, то результаты будут еще лучше.

Альтернативы Tenserflow

- Pytorch
- Caffe
- Theano
- MXNet

Занятие 9. Лекция. Методы представления структур твердого тела.

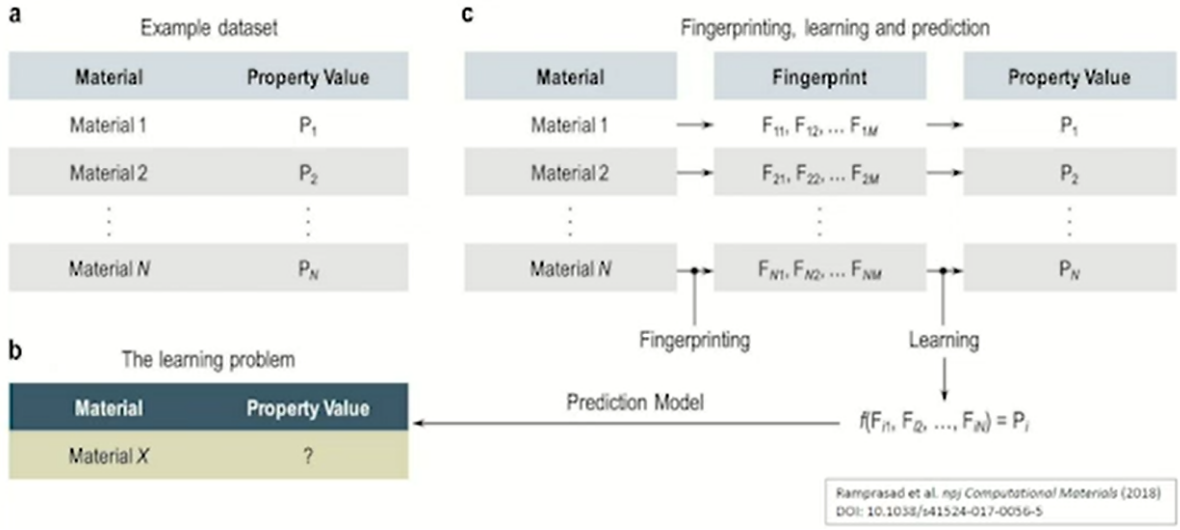


Рисунок 9.1. Схема построения предсказательной модели

На рисунке 9.1. представлена схема, где изображены этапы построения предсказательной модели физико-химических свойств. Данная схема отражает реальную последовательность действий, а не ту, которую следует ожидать. Схема начинается с конкретного набора данных, хотя обычно начинают с общей постановки задачи (в данном случае это второй шаг). Далее идет третий этап: перевод материалов в некоторое представление, пригодное для входного сигнала, а далее обучение предсказательной модели.

Методы представления, доступные на данный момент крайне многочисленны (рис 9.2.). При попытке классифицировать их исходя из универсального критерия вас ждет неудача. Поэтому в данном курсе ограничатся небольшой выборкой методов представления, но она является довольной представительной. Критерий – учет или игнорирование информации о локальном игнорировании отдельных атомов, содержащиеся в кристаллической структуре. На схеме представлено слева представления с учетом положения атомов, а слева с его игнорированием.

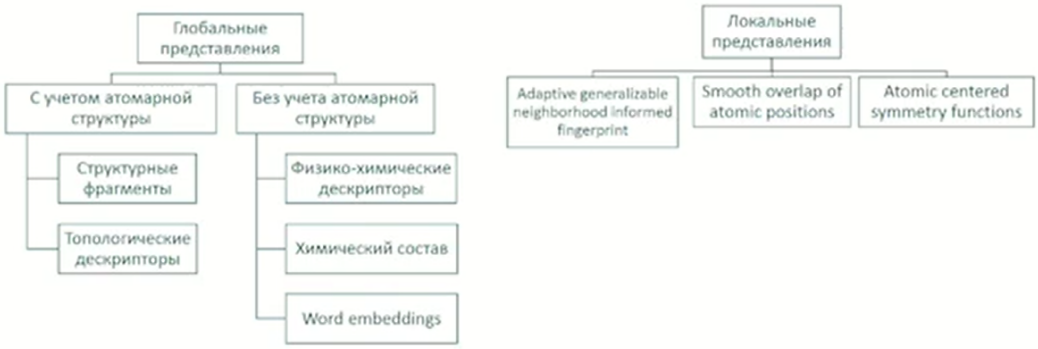


Рисунок 9.2. Классификация методов предскавления

Нейросетевые потенциалы взаимодействия

Начнем с первой группы методов. Какие свойства предсказываются в локальных представлениях: потенциальная энергия кристаллической структуры. Например, в нашем примере на рис.9.3. используются нейронные сети с предсказательными моделями. И каждая из моделей может предсказать энергетический вклад отдельного атома. Таким образом, мы работаем в рамках аддитивной схемы, где энергия всей структуры представляется как сумма отдельных энергетических вкладов:

$$E = \sum_{i=1}^{Nelem} \sum_{\alpha=1}^{Natom} E_j^i \qquad (9.1)$$

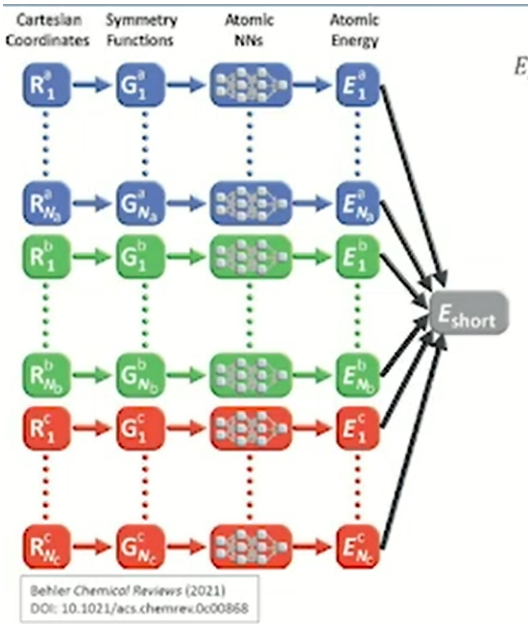


Рисунок 9.3. Нейронные сети с предсказательными моделями.

Данное приближение советует нейросетевым приближениям 2 поколения (учитывают виды симметрии). Кроме этого, существует 3 (учитывает дальнодействующие взаимодействия) и 4 (учитывают и локальный перенос заряда) поколения. Входными данными для данных сетей являются атомноцентрированные функции симметрии:

$$G_{i,\mu}^{rad} = \sum_{i \neq j}^{Natom \in Rs} \exp\left[-\eta\left(R_{ij} - R_s\right)^2\right] f_c\left(R_{ij}\right) \qquad (9.2)$$

По сути, это функция содержит радиальную и угловую часть. Используя обе части, можно учесть расстояния между атомами и взаимное расположение. Если же рассматривать угловую часть функции, это будет выглядеть так:

$$G_{i,\mu}^{ang} = 2^{1-\zeta} \sum_{i,j,k} \left(1 + \lambda cos\theta_{i,j,k}\right)^{\zeta} \exp\left[-\eta\left(R_{ij}^2 + R_{i,k}^2 + R_{j,k}^2\right)\right] f_c\left(R_{ij}\right) f_c(R_{ik}) f_c\left(R_{jk}\right) \qquad (9.3)$$

Данное сочетание нейронный сетей и атомноцентрированных функции в качестве функций представления оказалось настолько успешным, что нашло применение для аппроксимации потенциальной энергии для самых разнообразных химических систем. Так, например, с помощью данного подхода было установлено упорядочение молекул воды, находящихся в межслоевом пространстве (рис.9.4 а) гексогонального нитрида бора. Или восставленное с достаточно высокой точностью значение параметров решетки для гибридной системы металл органического каркаса (рис. 9.4 (б)). Третий пример: биметаллическая система, в которой вычислили какой их элементов предпочтительнее будет находится на поверхности.

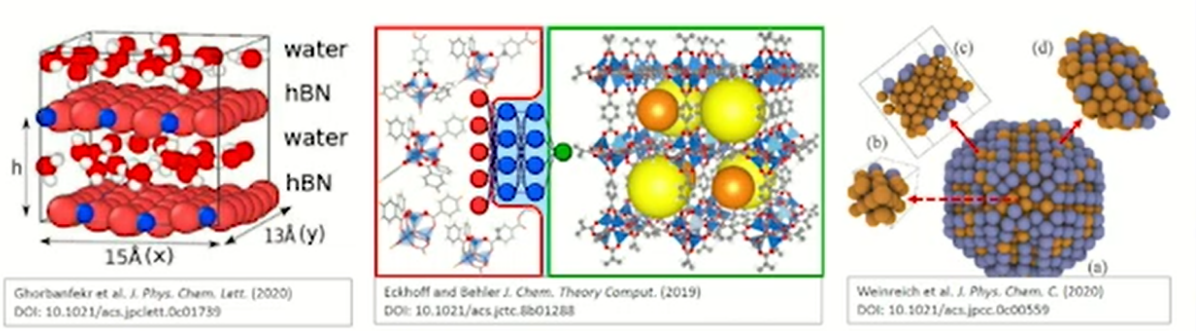


Рисунок 9.4. Упорядочение воды в межслоевом пространсве

Адаптивная генерализированная система fingerprint с учетом соседей (AGNI)

С одной стороны данных подход похож на тот, что описан выше. Включает в себя гауссово расширение электронной плотности, но критическое отличие состоит в том, что мы учитываем проекции, которые соединяет центральный атом с его соседями на выделенные направления:

$$V_i^u(\eta) = \sum_{i \neq j} \frac{r_{i,j}^u}{r_{i,j}} \exp\left[-\left(\frac{r_{ij}}{\eta}\right)^2\right] f_d(r_{ij}) \tag{9.4}$$

$$f_d(r_{ij}) = \frac{1}{2\left[\cos\left(\frac{\pi r_{ij}}{R_c}\right) + 1\right]} \tag{9.5}$$

Из этого следует, что мы можем использовать данный метод для предсказания сил, действующих на атом. Подобная методология успешна была использована в моделировании модели металлического алюминия. Ошибки в данном методе были минимизированы не только благодаря методу, но и количеству структур, на которых обучалась модель.

Сглаженные перекрытия атомных позиций (SOAP)

В рамках данной модели мы используем иной набор базисных функций. А именно набор радиальнобазисных функций, а также сферические гармоники. А также различные коэффициенты, которые позволяют отличить различные компоненты друг от друга.

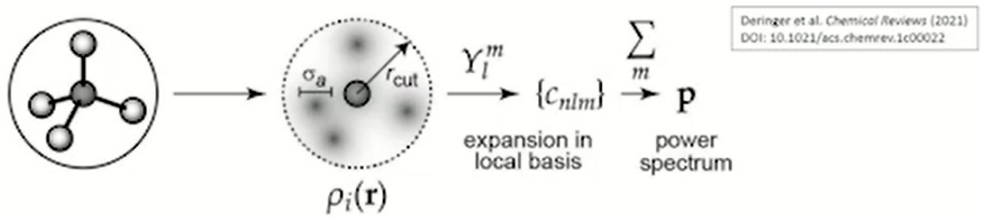


Рисунок 9.5. Набор сферических гармоник

$$\rho^{i,a}(r) = \sum_j \delta_{aaj} \exp\left(-\frac{|r - r_{ij}|^2}{2\sigma_a^2}\right) f_{cut}(r_{ij}) \tag{9.6}$$

Данный способ представлений используется для решения задачи для атома водорода. Кроме того, с помощью SOAP можно проводить количественное сравнение двух локальных окружений:

$$K(p, p') = \left(\frac{p * p'}{||p||||p'||}\right)^{\zeta} \tag{9.7}$$

Пример использования:

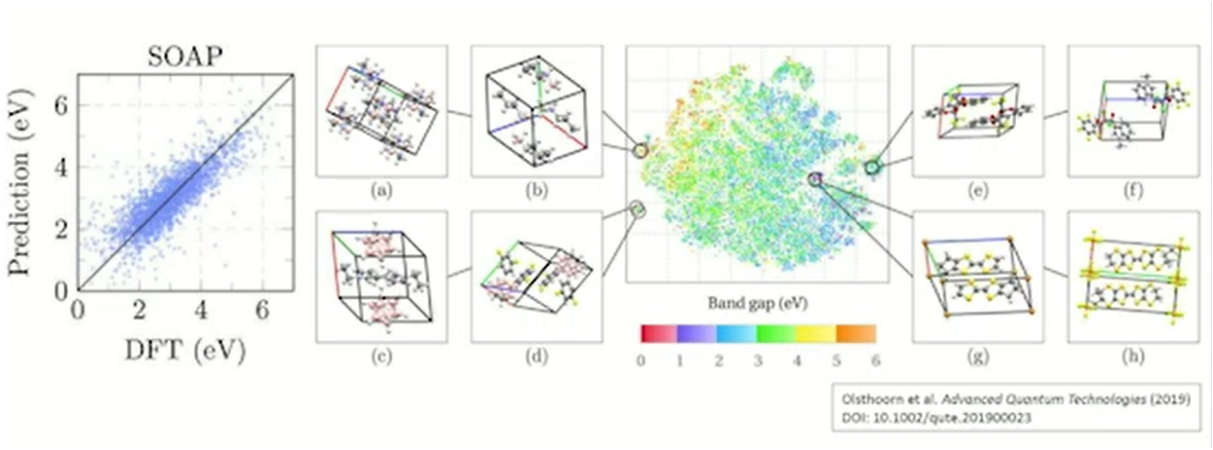


Рисунок 9.6. Предсказание ширины запрещенной зоны

Предсказана ширина запрещенной зоны молекулярного кристалла. Эта работа примечательна тем, что SOAP дескрипторы используются непосредственно, как представление молекулярных кристаллов и для визуализации соответствующего химического пространства.

Теперь поговорим о тех глобальных преставлениях, которые игнорируют информацию о локальном окружении атомов.

Физически интерпретируемые комбинации дескрипторов.

Здесь идет речь о тех дескрипторах, которые описывают физико-химические свойства, описывающие входные данные для предсказания другого физико-химического свойства. Естественно, мы используем некий набор аналитических функций формируем разнообразные комбинаций физико-химических свойств и далее данный набор используем для векторного представления для специфического метода машинного

обучения. В данном случает речь идет о модификации линейной регрессии (лассо регрессии). В рамках которой стараемся минимизировать отклонение предсказанных величин от их истинных значений, но еще и накладываем дополнительный штраф на абсолютные значения весовых коэффициентов:

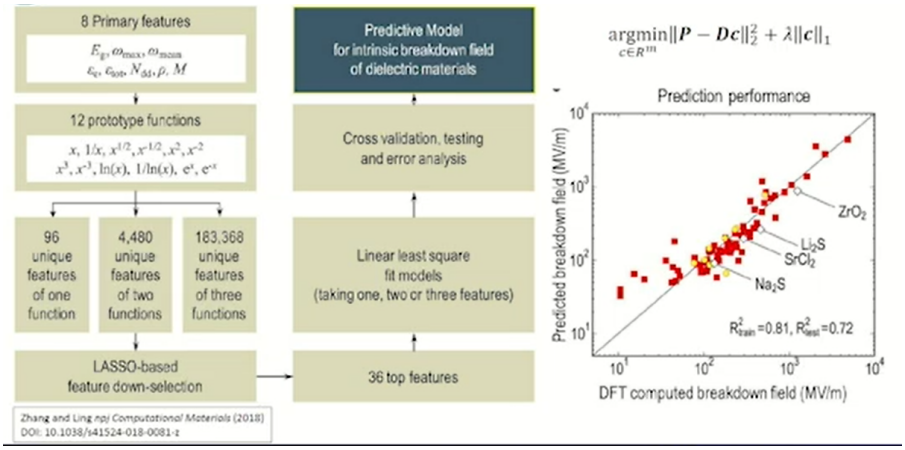


Рисунок 9.7. Предсказательная модель, основанная на линейной регрессии.

Если говорить простыми словами, вместо черного ящика (сложной аналитической функции) мы остаемся с явной аналитической функцией, которая будет включать в себя исходные физико-химические свойства и некие численные коэффициенты.

Представление на основе химического состава

Данный подход нашел широкое применение из-за того, что мы практически всегда знаем химический состав соединения. Далее стоит вопрос о том, как данную информацию можно подать в предсказательную модель. В данном случае речь идет о векторе известной длины, который содержит стехиометрический коэффицент хического элемента с атомным номером i. Пример представлен на рисунке:

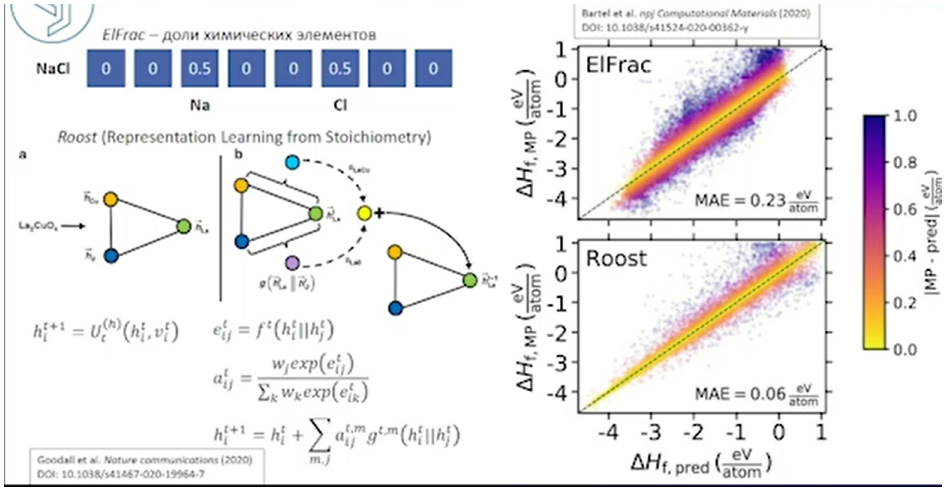


Рисунок 9.8. Предсказательная модель для химических данных

В правой части рисунка представлен пример предсказания энтальпии образования химических соединений. Значение абсолютной ошибки составляет 0.23 еВ/атом. При

использовании нейронных сетей можно добиться точности 0.06 eB/атом, исходя из точно таких же исходных данных. В чем заключается работа данных нейронных сетей? Каждый химический состав мы представляем в виде полносвязного графа. Каждая вершина соответствует химического элементу входящего в состав. Эти вершины описываются с помощью n-вектора, который содержит информацию и о химическом составе. Мы обновляем значение векторных представлений элементов, в ходе некоторого этаративного процесса, который учитывает значение векторных представлений элементов.

Векторное представление слов как представление химических элементов

Данное представление аналогично обрабатывает данные так, чтобы свести ее к векторному представлению отдельных химических элементов. Но главное отличие состоит в обработке естественного языка. Представим, что в нашем распоряжении имеется информация в виде химических текстов, исходя из этих данных формируется набор слов, наиболее часто встречающихся в химических текстах. Каждому слову представляем некий вектор, фиксированной длинны, полностью заполненный 0, за исключением i позиций. Такие векторы являются входным сигналом для модели. Выходным сигналом будет вероятность того, что слово в позиции j встречается в исходном наборе текстов в схожем контексте слову i. Конечная цель является векторное представление слов, т.е. выходной сигнал промежуточного расчета. В конечном счете мы можем каждому слову присвоить n-мерный вектор вещественных чисел. Таким образом мы имеем векторные представления для химических элементов и используя его, можем получить вектор, соответствующий химическому соединению. Данное представление позволяет предсказывать энергию образования.

Структурные фрагменты кристаллического графа

Современные структуры предполагают представление в виде графа, где в качестве вершин, представлены атомы, а ребрами являются химические связи. Пользуясь некой процедурой, которая учитывает межатомные расстояния, мы в конечном счеты получаем кристаллических граф. Но как данный объект свести к некому представлению, которые имело бы фиксированную размерность. Этого добиться можно за счет разбиения данного графа на подграфы. Для выбора подграфа использовалась информация о химических элементах, связях и информация о наличии или отсутствии линейных фрагментов и локальных окружений. Такой метод предсказаний может дать информацию о: ширине запрещенной зоны, коэффициент теплового расширения, теплоемкость.

Топологические дескрипторы

Данный метод представления учитывает атомарную структуру. В нашем распоряжении имеется некий набор точек, находящийся в трехмерном пространстве. Мы хотим неким образом описать их взаимное расположение. Для этого необходимо провести некую подготовительную работу, а именно разбить пространство на ячейки:

$$Vor(\rho) = \{x \in R^3 \big| \big||x - p|\big| \leq \big||x - q|\big| \forall q \in P\} \qquad (9.10)$$

Далее мы заменяем точки из подготовительного набора на сферы с центром в этих точках. Потом формируем альфа формы, которые представляют собой пресечения этих форм:

$$alpha_r(P) = \{\sigma \in P \cap B_r(p) \cap Vor(p) \neq \emptyset\} \quad (9.11)$$

Достаточно понимать, что мы таким образом отслуживаем те значения радиусов сфер, при которых появляются некие топологические особенности (пустоты, связанные компоненты – циклы). Данную информацию можно представить в виде графика, где на оси абсцисс откладываем радиус, при котором топологическая особенность появляется, а по оси ординат радиус при котором исчезает топологическая особенность.

Под набором точек мы подразумеваем положение атомов в кристаллической структуре. И данных подход позволяет добиться высокой точности предсказания энтальпии образования. В другом подходе можно рассмотреть не отдельные атомы, а точек семплированных на поверхности внутренних пор. Такой способ представления актуален для нанопористых материалов и описание адсорбционных свойств.

Дело не ограничивается теми представлениями, которые мы успели обсудить. Даже если смотреть только на один класс нанопористых материалов, то его можно описать десятками разных способов.

2D дифрактограммы как структурное представление

Не все методы вписываются в общепринятые классификации методов и особняком стоит метод 2D дифрактограмм. В качестве предсказанной величины выступает сингония кристалла. Суть в том, что мы не пытаемся представить структуру, не пользуясь ни одним из выше обсуждаемых методов. Мы работаем с теми данными, который имеется в распоряжении исходно. Однако обработка дифрактограмм, спектральных данных и т.д. не является «мейнстримом». Для данных представлений используются нейронные сети на графах и сверточные нейронные сети.

Занятие 10. Семинар. Методы оптимизации гиперпараметров.

Оптимизация гиперпараметров. В любой модели машинного обучения есть 2 вещи: параметры и гиперпараметры. Параметры – это то, что модель подстраивает в процессе обучения. Гиперпараметры – это то, что задаем мы (модель машинного обучения). Для улучшения подбора гиперпараметров были созданы некоторое алгоритмы. Рассмотрим на примере.

Подключим все необходимие функции:

```
import tensorflow as tf
from rdkit import Chem
import numpy as np
from rdkit.Chem import AllChem, DataStructs
from sklearn.model_selection import train_test_split
print("TensorFlow version:", tf.__version__)
```

Подключим гугл драйв:

```
mols = Chem.SDMolSupplier("/content/drive/MyDrive/ML in chemistry/15_seminar/herg_short.sdf")
x = []
y = []
def to_cat(val):
if val == "true":
return 1
else:
return 0
def get_descr_value(molecule):
out_arr = np.zeros((1,), dtype=int)
descriptor = AllChem.GetMorganFingerprintAsBitVect(molecule, 6, 2048)
if isinstance(descriptor, DataStructs.cDataStructs.ExplicitBitVect):
Data Structs.Convert To Numpy Array(descriptor, out_arr)
return out_array
 return descriptor
for mol in mols:
  try:
y.append(int(mol.GetProp("herg_activity")))
x.append(get_descr_value(mol))
  except KeyError:
× = np. asarray(×)
Y = np. asarray(y)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
```

Следующий метод – make_model, который позволяет работать с нейронной сетью – хороший, но долгий. Давайте его попробуем:

```
    def make_model(n_dense=3, dropout=0.4, activation="relu", hidden_size=1024,
lr=0.001, optimizer=tf.optimizers.Adam):
loss_fn = tf.keras.losses.Binary Cross entropy()
model = tf.keras.models. Sequential()
model.add(tf.keras.layers.Dense(2048, activation=activation))
for _ in range(n_dense - 1):
        model.add(tf.keras.layers.Dense(hidden_size,activation=activat)
        model.add(tf.keras.layers.Dropout(dropout))
        # выходные слои
model.add(tf.keras.layers.Dense(hidden_size, activation=activation))
model.add(tf.keras.layers.Dense(1))
model.add(tf.keras.layers.Activation(activation='sigmoid'))
optimizer = optimizer(learning_rate=lr)
model.compiler(optimizer=optimizer, loss=loss_fn)
return model
# запускаем тренировку модели
model = make_model()
callback=tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=5)
model.fit(x_train,y_train,epochs=l0,validation_data=(x_test,y_test), callbacks=[callback])
```

Но данная модель дает не очень хороший результат, поэтому попытаемся ее оптимизировать. Например, с помощью Random Forest.

```
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, log_loss
    rf = RandomForestClassifier()
    print("Parameters currently in use:\n')
    print(rf.get_params())
```

Выберем параметры, по которым будем оптимизировать:

- n_estimators = Число деревьев
- max_features = Максимальное число разветвлений из одной ноды
- max_depth = Максимальная глубина дерева
- bootstrap = Метод выбора данных (с возвратом или без)

```
    # Давайте зададим грид
    param_grid = {'n_estimators': [int(x) for x in np.linspace(start = 100, stop = 500, num =
10)],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [int(x) for x in np.linspace(2, 20, num = 5)],
    'bootstrap': (True, False],
    'criterion': ["gini", "entropy"]
    # Функция, оценивающая качество модели
    def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    print('Model Performance')
```

```
# print("Average loss: {:0.4f}"format(log loss(test labels, predictions)))
print('f1={:0.2f)%.',format(f1_score(test_labels,predictions)))
```

Алгоритмы оптимизации.

Перейдем к алгоритмам оптимизации. Самые популярные – это Grid Search и Random Search. На картинке 10.1. представлена разница между ними:

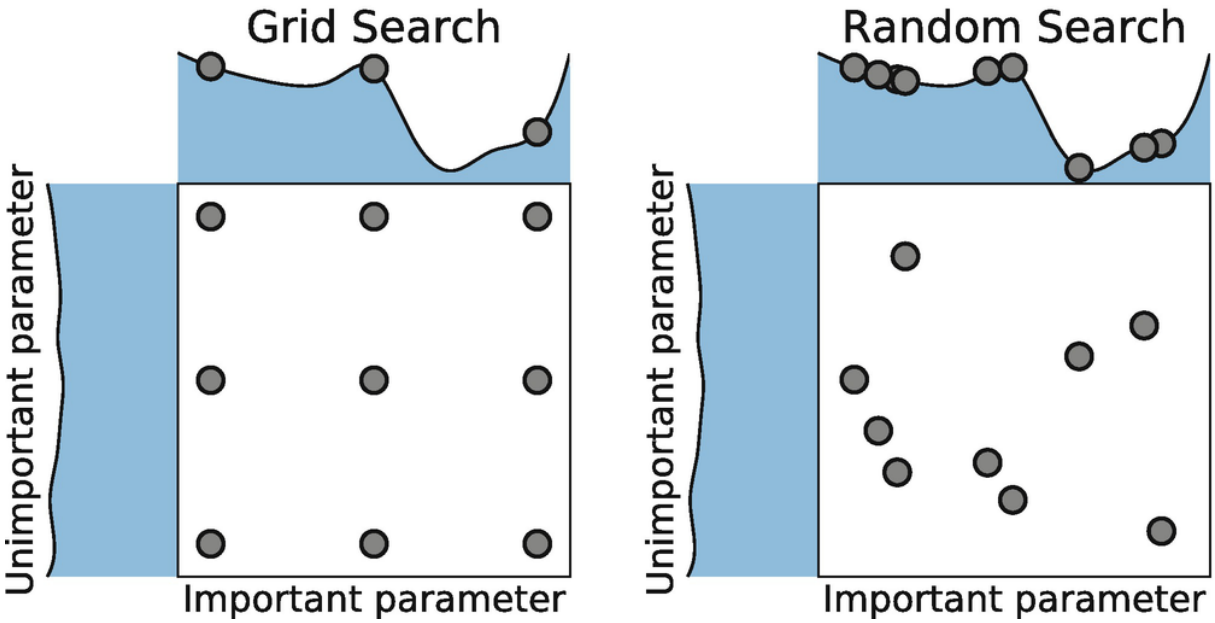


Рисунок 10.1. Разница между двумя алгоритмами оптимизации.

Начнем с grid Search:

```
from sklearn. model_selection import GridSearchCV
grid_search = Grid Search CY(param_grid=param_grid, estimator=rf, cv=3, verbose=2, n_jobs=-1)
# Fit the grid search model
grid search fitix train, y train)
base_model = RandomForestClassifier(n_estimators = 10, random_state = 42)
base_model.fit(x_train, y_train)
evaluate(base_model, x_test, y_test)
best_grid = grid_search.best_estimator_
evaluate(best_grid, x_test, y_test)
```

Перейдем к random search:

Метод чуть лучше, чем Grid search. Выбирая значения случайно, мы сильно уменьшаем область поиска, поэтому можно случайно упустить оптимальное значение. Запуск его аналогичен:

```
from sklearn.model_selection import Randomized Search C V
random_search = RandomizedSearchCV(estimator = rf, param_distributions = param_grid, n_iter = 10, cv = 3, verbose=2, r
#Fit the random search model
random_search.fit(x_train, y_train)
base_model = RandomForestClassifier(n_estimators = 10, random_state = 42)
```

```
base_model.fit(x_train, y_train)
evaluate(base_model, x_test, y_test)
best_random = random_search.best_estimator_
print(best_random.get_params())
evaluate(best_random, x_test, y_test)
```

Tree Parzen Estimator (TRE) search

Перейдем к тому, что сейчас реально используется в работах. **Tree Parzen Estimator (TRE) search.** Преимущество данного алгоритма состоит в том, мы учитываем историю тренировок. Давайте попробуем воспользоваться данным методом:

```
    import pandas as pd
    from sklearn import metrics
    from sklearn.model_selection import cross_val_score
    from sklearn.preprocessing import StandardScaler
    from hyperopt import type, hp, fmin, STATUS_OK, Trials, STATUS_FAIL
    from hyperopt.pyll.base import scope
    import warnings
    warnings.filterwarnings("ignore")
    space = {"n_estimators": scope.int(hp.quniform("n_estimators", 100, 600, 100)),
    "max_depth": hp.quniform("max_depth", 2, 15,1),
    "criterion": hp.choice("criterion", ['gini', 'entropy']),
    'max_features': hp.choice("max_features", ['auto', 'sqrt']),
    'bootstrap': hp.choice("bootstrap", [True, False])
scorer=metrics.make_scorer(metrics.log_loss,greater_is_better=False)
    def hyperparameter_tuning(params):
    rf = RandomForestClassifier(**params, n_jobs=-1)
    rf.fit(x_train, y_train)
    y_pred = rf.predict(x_test)
    acc = metrics.log_loss(y_test, y_pred)
    if acc is None:
    return {"status": STATUS_FAIL}
else:
    return {"loss": ace, "status": STATUS_OK}
    trials = Trials()
    best = fmin(
    fn=hyperparameter_tuning,
    space=space,
    algo=tpe.suggest,
    max_evals=10,
    trials=trials
    print("Best: {}".format(best))
```

Проверим насколько хороши получилось:

```
base_model = RandomForestClassifier(n_estimators = 10, random_state = 42)
base_model.fit(x_train, y_train)
evaluate(base_model, x_test, y_test)
```

При проверке получилось, что значение повысилось на 0.24, в сравнении с базовом алгоритмом. Хотя здесь еще есть простор для улучшений.

Занятие 11. Лекция. Методы глобальной оптимизации

Если до этого мы решали прямую задачу предсказания свойств по структуре, то сейчас можно сгенерировать новые молекулы, кристаллы и т.д. под заданные свойства. Прежде чем перейти к этому, необходимо разобраться с отдельным классом алгоритмов, которые называются глобальной оптимизацией. Разделения на различные классов по номенклатуре алгоритмов весьма условное.

Задачи моделирования

$$F(x) => Y$$

Знаем набор X и Y
Численно моделируем F
Цель - найти Y для новых известных X

$$F(X, c_1, c_2, c_3, \dots) => Y$$

Знаем набор X и Y
Численно моделируем F при разных наборах (гипер)параметров
Цель - найти оптимальный набор параметров

$$X, \exists F(X) = Y$$

Знаем X
Можем найти Y
Цель - найти (экстремумы) F используя минимальный набор X

$$F^{-1}(Y) => X$$

Переход к обратной задаче. При известных Y, найти X?

В чем сложности? Почему задачи, которые пытаются найти минимумы считаются небанальной? Почему нужно придумывать новые методы этой оценки. Посмотрим на пример:

Рисунок 11.1. Поиск максимум и минимумов

На данной функции видим два минимума, локальный и глобальный. По значению самой функции эти значения сильно различаются. Если смотреть на модели машинного обучения, то в одном случае мы можем получить так хорошую (если попасть в глобальным минимум), так и плохую модель (локальный минимум).

Локальная оптимизация

Если предположить, что на поверхность, заданную какими-либо параметрами, бросили случайную точку. Можно ли поискать локальный минимум у функции, которая окружает эту точку. В этом случае можно использовать метод градиентного спуска:

$$x_{k+1} = x_k - \alpha f'(x_k) \quad (11.1)$$

где k – номер шага интеграции, $\alpha -$ размер шага. Данный метод сводится к достаточно простой мысли, мы в точке, которую случайно бросили на неизвестную параметрическую функцию, оцениваем значение функции, а также градиент этой функции. Однако этот метод поможет найти только ближайший локальный минимум, а настоящий глобальный минимум может находится вообще в другом месте. То есть нужно перейти от локальной оптимизации к глобальной.

Глобальная оптимизация

Пример, можно составить *сетку* тех параметров, которые мы будем оптимизировать. Далее по этой сетке начинаем проверять набор данных:

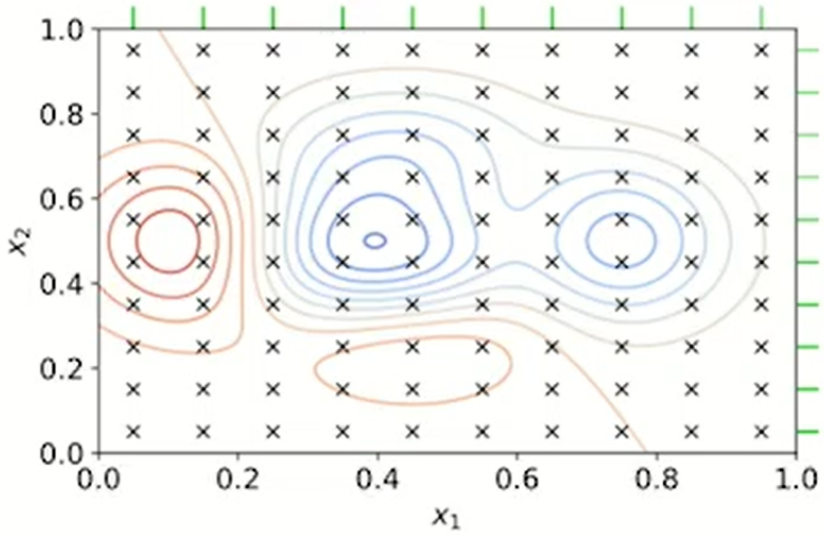


Рисунок 11.2. Проверка набора данных на сетке.

Данная сетка состоит из 10*10 точек (100), видно, что некоторые точки довольно близки к глобальному минимуму, но в них не попали. Поэтому нужно взять сетку поплотнее. В абстрактной задаче это не сложно. Но если это эксперимент, в котором надо получить данные, ограничен.

В качестве альтернативы можно использовать случайные значения (те же 100 точек) и надеяться, что мы попадем в локальные/глобальные минимумы/максимумы. С одной стороны, мы можем не попасть, но с другой стороны, посмотрев на плотность распределения по каждой отдельной переменной (зеленые штрихи), мы увидим, что рассмотреть гораздо более полное распределение.

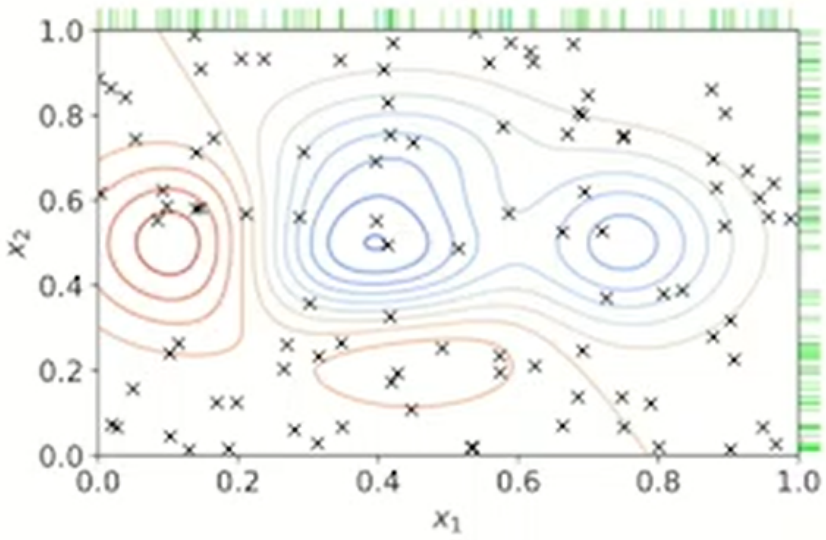


Рисунок 11.3. Проверка случайного набора данных на сетке

Но чем плохи данные подходы:

- можно промахнуться
- проклятье размерности (если параметров больше, чем 2)
- нет учета истории поиска

Байесовский формализм

Классическая (частотная) статистика: $P(D|\theta)$

Частота появления информации D при проведении эксперимента θ.

Модель (θ) фиксирована, информация (D) меняется.

Байесовская статистика

Степень уверенности в истинности суждения θ, при наличии информации D.

Информация (D) фиксирована. Модель (θ) меняется

Теорема Байеса

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)} \quad (11.1)$$

Показывает, как обновить существующую гипотезу основываясь на новых доказательствах.

Представим, что у нас есть функция. На рисунке 11.3 представлена прерывистой линией, саму функцию мы не знаем, но должны оптимизировать. При этом мы можем поставить несколько случайных точек, т.е. определим несколько точных значений этой функции (черные точки). Мы знаем значение этой функции в трех точках. Далее эту информацию используем для оценки всей функции и попытаемся найти минимум и максимум. Воспользуемся двумя вспомогательными классами функции. Проведем некую гладкую кривую через три точки. Повторим этот процесс несколько раз, для того чтобы распределение плотности данных гладких функций было гауссовским. То есть в каждом вертикальном срезе у нас сгенерировано много функций так, чтобы распределение было гауссовским. На основании данной модели нам надо сделать предположение куда ставить точку дальше, чтобы найти минимум или максимум. Мы строим вспомогательную функцию, которая уравновешивается двумя параметрами.

Либо там, где мы предполагаем максимум, или там, где мы про функцию знаем меньше всего. Важен баланс обоих предположений, чтобы найти именно глобальный максимум, а не локальный.

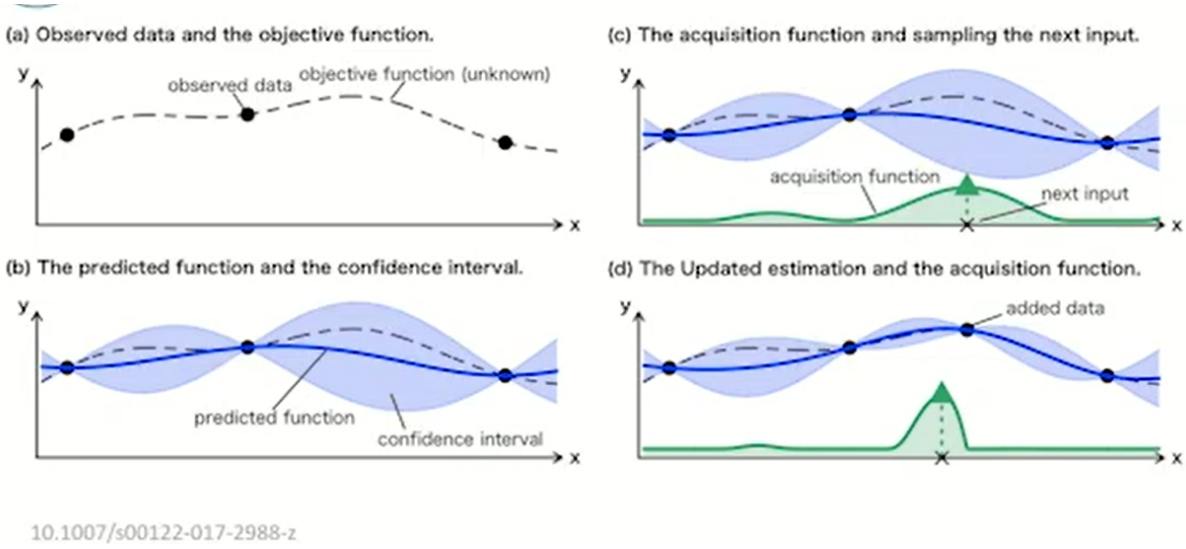


Рисунок 11.4. Применение байесовского формализма

Нейронные сети, которые помогают найти глобальные максимумы/минимумы:
GAN
Seq2Seq
Autoencoder
(Смотри лекцию по нейронным сетям)

Природные алгоритмы (Nature-inspired algorithms)

Evolutionary (имитируют Дарвиновскую эволюцию):
• Genetic
Swarm intelligence zoo:
• Artificial bee colony
• Gray wolf optimization
• Ant colony optimization
• Firefly optimization

Эволюционные алгоритмы

Моделируем естественный отбор (генетический отбор) или ищем самую приспособленную особь. Иными словами, оптимизировать некую функцию для того, чтобы искать глобальную минимум/максимум. Зададим основные определения. Один параметр – ген особи (выделен в красную рамку на рис. 11. 5) Совокупность данных генов – это особь. Кроме того, мы будем моделировать совокупность генов не для одной особи, а для целой популяции (фиолетовая рамка). Цель – найти самую приспособленную особь. При этом первое поколение будет сгенерировано абсолютно случайно. Далее нужно оценить значение целевой функции (для каждой особи мы посчитаем таргетное значение). Потом зададим правило отбора (те особи, которые имеют худший набор – исключаются). Оставшиеся особы необходимо размножить до определенного размера популяции (проведем кроссовер). Кроссовер – обмен генами между особями (может быть частичный). Добавим случайные мутации (какие-то значения в векторе после кроссовера случайно поменяем).

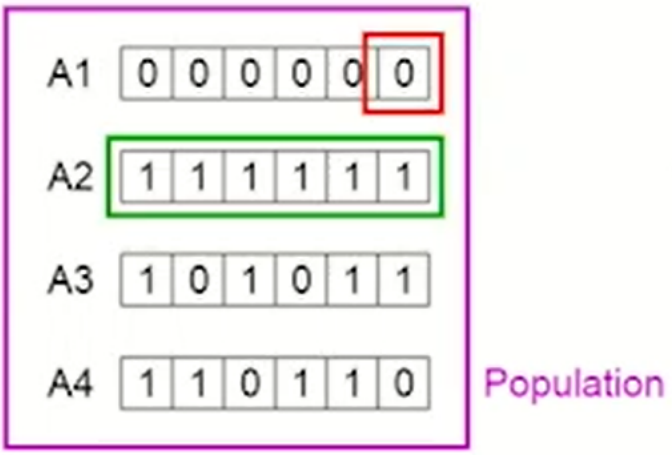


Рисунок 11.5. Представление эволюционного алгоритма

После того, как мы проделали все действия можно вернутся к оценке значения целевой функции. С каждым последующим поколением мы будем подбираться к глобальному минимуму.

Однако, существуют сложности: время, оценка целевой функции, масштабируемость, остановки, локальный минимум.

Swarm intelligence

Коллективное поведение децентрализованной самоорганизующейся системы:

Nature-inspired

Поиск минимума или пути

Например, можно поведение насекомых, небесных тел и т.д. имитировать для поиска минимумов/максимумов.

Рассмотрим несколько вариантов роевого интеллекта:

Particle swarm optimization - социальные взаимодействия:

• Мы все куда-то бежим (генерируем решения и направления их изменения):

• Вспоминаем старые добрые времена и стремимся вернуться в место, где было хорошо (добавляем к решению память о решениях на предыдущих итерациях)

• Не хотим останавливаться (поиск новых решений)

• Общаемся с другими и смотрим, куда бегут они - может там еще лучше? (обмен решениями)

• Импульсивны (добавляем случайные изменения к вектору скорости)

Муравьи

Ant Colony Optimization - поиск пути (или минимума):

• Муравей находит хоть какой-то путь от точки до точки и помечает его феромонами

• Муравьи ищут новые пути, помечая их феромонами

• Муравьи стремятся к феромонам

• Феромоны испаряются со временем, но усиливаются другими проходящими муравьями

Пчелы

Artificial Bee Colony (ABC) - пчелы ищут нектар:

• Собиратели - носят нектар от источников пищи и смотрят по сторонам (обозначение положения известных минимумов и локальный поиск в их окрестностях)

• Наблюдатели - смотрят за собирателями и направляют их в новые места (вероятностная оценка формы целевой функции)

• Разведчики - ищут новые места, если собиратели ленятся (если минимум не оптимизируется за несколько последовательных ходов, случайным образом ищут новый)

Гравитационный алгоритм

Gravitational search algorithm - рой планет:

• У каждого небесного тела есть масса, скорость и ускорение (аналогично рою частиц, но совокупность всех сил вместо обмена информацией)

• Тела притягиваются согласно Ньютоновской динамике

• Обновляем гравитационную постоянную

Поиск гармонии

Harmony search:

• Каждый музыкант играет какую-то ноту (переменная)

• Если хор гармоничен - музыкантов премируют (запоминаем хорошее решение)

• Если нет - кого-то выгоняют (выкидываем особь из популяции)

• Просим всех попробовать еще (добавляем мутаций)

Теорема об отсутствии бесплатных завтраков

• Не бывает бесплатных завтраков (There's no such thing as a free lunch)

• Два алгоритма оптимизации одинаково (не)эффективны на выборке из всех возможных задач

Где использовать методы, рассмотренные в лекций:

• Оптимизация гиперпараметров

• Параметризация моделей:

Термодинамические модели (10.1080/00986445.2017.1390455)

Потенциалы межатомного взаимодействия (10.1021/acs.jpcc.8b09917)

DFT (10.1007/s00706-018-2335-3)

• Оптимизация геометрии

• De novo design

• Планирование экспериментов

Занятие 12. Семинар. Предсказание модели в материаловедении

Работа с кристаллографическими данными

Начнем с импортирования данных:

```
import warnings
warnings. filterwarnings("ignore") # игнорируем предупреждения
import zipfile
with zipfile.ZipFile("/content/drive/Myorive/Colab Notebooks/12/datasets.zip", "r") as zip_ref:
    zip ref.extractall("/content/drive/HyDrive/Colab Notebooks/12/")
```

Crystallographic Information File (CIE) является наиболее популярным способом хранения кристаллографической информации.Рассмотрим содержимое файла данного формата, загрузив его с помощью стандартных средств Python.

```
with open("/content/drive/My Drive/Colab Notebooks/12/datasets/sio2_mp-546794_computed.cif", 'r') as file: cif_data=file. read()
print(cif_data)
# generated using pymatgen
data_si02
symmetry_space_group name H-M "P 1"
_cell_length_a 5.13584237
_cell_length_b 5.13584237
_cell length c 5.13584237
_cell_angle_alpha 121.02203994
_cell angle beta 121.02203994
_cell_angle gamma 88.23870671
_symmetry Int Tables number
_chemical formula_structural Sio2 Sio2
_chemical_formula_sum
_cell volume 94.26185406
cell formula_units_Z  2
1oop_
_symmetry_equiv_pos_as_xyz
1    ‘x, y, z’
loop_
_atom site_type_symbol
_atom_site_label
_atom_site_symmetry multiplicity
_atom site_fract_x
_atom site fract y
_atom_site_fract_z
_atom site occupancy
```

Для автоматического извлечения информации, содержащейся в CIF файле, удобно воспользоваться библиотекой pymatgen, предоставляющей широкий набор

инструментов в области вычислительного материаловедения (computational materials science). Основной объект (точнее, экземпляр класса), с которым нам предстоит работать - pymatgen.core. structure.Structure. Данный класс содержит встроенный метод, позволяющий импортировать содержимое GIF файла, хранящегося на диске:

```
from pymatgen.core. structure import Structure
structure = Structure.from_file("/content/drive/my brave/crab Notebooks/12/datasets/sio2_mp-546794_computed.cif")
structure
```

В случае корректной обработки файла у нас появляется возможность обращаться к отдельным составляющим кристаллографической информации с помощью встроенных атрибутов класса pymatgen.core. structure.Structure:

```
structure.lattice # информация о кристаллической решётке - длины рёбер и углы в кристаллической ячейке, её объем и т.д.
structure.atomic_numbers # порядковые номера химических элементов, соответствующие атомам в ячейке
structure.frac_coords # в дробные координаты атомов в системе кристаллической ячейки
structure. cart_coords # декартовы координаты атомов
```

Возможности pymatgen простираются далеко за пределы обработки CIF файлов. Для демонстрации ниже импортировано несколько классов (методов) данной библиотеки, чьё предназначение очевидным образом следует из названия.

```
# для работы с непериодическими структурами
from pymatgen.core.structure import Molecule
# для работы с внутренними границами раздела - границами зерен
from pymatgen.analysis.gb.green import Grain Boundary
# для работы с фазовыми диаграммами
from pymatgen.analysis. phase diagram import Phase Diagram
# расчет размерности структуры методом, предложенным в работе:
from pymatgen. analysis.dimensionality import get dimensionality larsen
```

Предсказание ширины запрещённой зоны

Первая модель будет использована для предсказания запрещенной зоны двойных перовскитов ($AABBO_6$). Вся необходимая информация находится в файле формата .csv, чьё содержимое мы загружаем с помощью (уже хорошо знакомой) библиотеки pandas.
В нашем распоряжении имеются только химические формулы рассматриваемых соединений. В данном случае pymatgen.core. structure.Structure не является оптимальным объектом для хранения/обработки информации – воспользуемся pymatgen.core.composition.Composition.

```
from pymatgen.core.composition import composition
# переводим содержимое столбца df["formula"] в список экземпляров класса Composition
compositions = list(map(Composition, df['formula']))
# взглянем на первый элемент в списке
```

compositions[0]
Comp: LaI NbI AlI AgI 06

Как и в случае с pymatgen. core. structure. Structure, в классе pymatgen. core.composition. Composition имплементировано множество полезных атрибутов и методов для работы с химическим составом.

```
compositions[0].chemical_system # химическая система
```

Element fractions

Для создания признакового описания воспользуемся одним из самых простых подходов - представим каждое соединение в виде вектора, содержащего атомные дс и входящих в него химических элементов.

Метод **get_atomic_fraction()** класса pymatgen.core.composition.Composition позволяет реализовать функцию для расчёта такого вектора в нескольких строках кода, однако более "продвинутые" алгоритмы векторизации твердого тела (в особенности те, что учитывают кристаллическую структуру) могут потребовать значительно больших усилий. На помощь нам придёт ещё одна замечательная библиотека python - **matminer** в которой имплементировано большинство популярных методов векторизации твердого тела, включая вышеупомянутый (под именем Element Fraction).

```
import numpy as np # понадобится в дальнейшем для работы с численными массивами
from mainer featurizers.composition.element import ElemntFractich # для расчёта вектора атомных долей
featurizer ElementFraction() # инициализируем экземпляр класса Element Fraction
X = - ip.array(featurized.features_and(compositions)) # рассчитывает векторы атомных долей для списка химических составов
X.shape
```

Теперь взглянем на распределение целевой переменной - ширины запрещенной зоны. Это позволит принять решение о необходимости дополнительных шагов на стадии препроцессинга данных.

```
from seaborn import display # импортируем метод для построения привлекательных гистограмм распределения
y = df['gap gllbsc'].values # извлекаем из df список значений в формате numpy.ndarray
ax = displot(y, bins-100) # строим гистограмму распределения с заданным количеством столбцов (100)
ax. set(xlabel="bandgap, ev") # добавляем подпись к оси абсцисс
```

Получаем вот такой график:

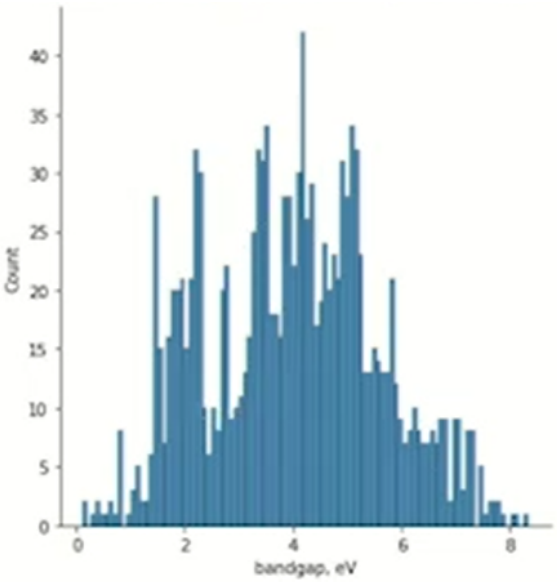


Рисунок 12.1. Результат расчета ширины запрещенной зоны

Разобьём случайным образом исходную выборку на два подмножества. Обучающая выборка используется непосредственно для оптимизации параметров (т.е., обучения) предсказательной модели, Тестовая выборка послужит для характеризации качества её работы. Соответсвующий метод импортируем из библиотеки scikit leam.

```
from sklearn.model_selection import train_test_split
x train, x test, y train, y test - train test_split
```

Двумерный массив Nxм, содержащий M-мерное векторое представление для структуру, и одномерный массив размерности n, содержаций значения целевой переменной для н структур test size 0.2, доля структур из исходной выборки, вошедших в тестовую выборку random state o и параметр для воспроизводимого случайного разбиения исходной выборки.

Теперь мы можем обучить модель машинного обучения. В иллюстративных целях, ограничимся одной из самых простых линейной регрессией.

```
from sklearn. linear model import LinearRegression
model = LinearRegression() # инициализируем модель линейной регрессии
model.fit(x_train, y_train) # обучаем модель
```

Осталось выяснять, насколько хорошо обученная нами модель воспроизводит целевую переменную. Для этого имплементируем две функции: get_regression_metrics() предназначена для расчёта основных регрессионных метрик (R2, MAE, RMSE), reg plot() позволяет сопоставить истинные и предсказанные значения (так называемый parityplot).

```
from matplotlib import pyplot as plt
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
def get_regression metrics(y_true, y pred):
metrics # инициализируем словарь и рассчитываем основные регрессионные метрики
metrics[‘$R^2$’] - dound(r2_score(y_true, y pred), 3) # коэффициент детерминации с точностью в 3 знака в дробной части
metrics["MAE"] = round(mean_absolute_error(y_true, y pred), 3) # MAE с точностью в 3 знака в дробной части
```

```
metrics['RMSE"] - round(np.sqrt(mean squared error(y true, y pred)), 3) # RMSE с точностью в 3 знака по дробной части
return metrics # возвращаем словарь со всеми метриками
def reg plot(y true, y pred, label=**, units=""):
plt. figure(figsize (7, 7)) # задаем размер графика
ax_min пр.min(пр. hstack((y_true, y_pred))) - 0.5 # рассчитываем минимальное значение, отложенное на обеих осях
ax max - пр.вах(пр. hstack((y true, y pred))) + 0.5 # рассчитываем максимальное значение, отложенное на обеих осях
plt. xlim(ax min, ax max) # задаем диапазон значений для оси абсцисс
plt.ylim(ax min, ax max) # задаем диапазон значений для оси ординат
plt.scatter(y_true, y_pred, marker ***) # наносим на график точки с координатами (y_true_i, y_pred_i)
plt.plot([ax min, ax max], [ax min, ax max], color 'k', linestyle"") # штрихпунктирная линия х у
plt.xlabel(f"(label} calculated, [(units)]") # подпись к оси абсцисс
plt.ylabel(f"(label} predicted, [{units}}") # подпись к оси ординат
plt.gca().set_aspect("equal", adjustable="box") # задаём равный маситаб по оси абсцисс и ординат
```

На выходе получим график регрессии.

Предсказание модуля сдвига неорганических кристаллов

Наш следующий пример посвящен предсказанию модуля сдвига для выборки неорганических кристаллов. Данный параметр характеризует отклик материала на приложенное сдвиговую нагрузку. На этой раз нам доступна кристаллическая структура материалов, которая хранится в виде CIF файлов в одной папке. Взглянем на её содержимое. Импортируем содержимое:

```
from os import listdir # импортируем функцию, которая возвращает список файлов и папок в указанном каталоге
listdir("/content/drive/hydrive/CollabNotebooks/12/datasets/brgoch_superhard_training/kits") # просмотр папки, содержащей CIF файлы
```

Преобразуем содержимое CIF файлов в экземпляры класса pymatgen.core. structure.Structure.

```
structures - list(map(Structure. from_file,
[f'/content/drive/MyDrive/ComNotebooks/12/datasets/brooch_super hard_training/kit/(i).if' for i in range(1000)]
))
```

Список значений модуля сдвига загружаем из файла с расширением про - формат, в котором обычно хранятся numpy массивы.

```
y=np.load("/content/drive/MyDrive/ComNotebooks/12/datasets/brooch_super hard_training/targets.npy")[:1000]
ax.display(y, benz 100) #строим гистограмму распределения целевой переменной
ax.set(xlabel-'shear modulus, GPa') # подпись к оси абсцисс
```

Модуль сдвига принимает значения в широком диапазоне (несколько десятичных порядков). На практике больший интерес представляет его десятичный логарифм.

```
y = np.log10(y) # берём логарифм от целевой переменной
ax. displot(y, bins-100) # строим гистограмму распределения новой целевой переменной
ax.set(xlabel="shear modulus, Gpa") # подпись к оси абсцисс
```

Bag of bones

Известная кристаллическая структура материала (в дополнение к химическому составу) принципиальным образом расширяет набор методов векторизации, доступных для построения предсказательной модели. В данном примере мы воспользуемся подходом Bag of Bonds (BoBs).

Библиотека matminer содержит реализацию и этого алгоритма. Элементы B os вектора рассчитываются как элементы кулоновской матрицы (Coulomb Matrix) - ещё один метод векторизации структуры, доступный в matminer.

```
from matt miner. featurizers. structure. bonding import BagofBonds
from matminer.featurizers.structure.matrix import coulombratrix
featurizer - Bag of Bones(coulomb matrix-Coulombeatrix(flatten-False)) # инициализируем bag of bones
featurizer.fit(structures) # формируем bag of bonds
np.array(featurizer featurize_many(structures)) # рассчитываем bag of bones для каждой структуры
```

Как и в первом примере, разбиваем исходную выборку на два подмножества.

```
x_train, x_test, y_train, y_test train_test_split(
x, # двумерный массив Nxm, содержащий M-мерное векторное представление для N структур
y, # одномерный массив размерности n, содержащий значения целевой переменной для структур
test size 0.2, # доля структур из исходной выборки, вошедших в тестовую выборку
random state # параметр для воспроизводимого случайного разбиения исходной выборки
```

В отличие от значений атомных долей, ограниченных диапазоном [0, 1], значения, полученные большинством других методов векторизации, не имеют строго установленных границ. Взглянем на распределение максимальных и минимальных значений отдельных признаков, составляющих Bag of Bones.

ax = display(np.min(x trail, axis 1), bins=100) # строим гистограмму распределения минимальных значений признаков из Bobs

ax.set(xlabel-"min feature value") # подпись к оси абсцисс

Чтобы привести к единому диапазону значения всех признаков, проведём нормализацию данных с помощью функции MinMaxScaler().

from sklearn.preprocessing import MinMaxScaler

scaler Minmaxscaler() # инициализируем экземпляр класса MinMaxscaler

scaler.fit(x train) # получаем параметры Mintaxscaler исходя из обучающей выборки

x train scaler.transform(x train) # нормируем обучающую выборку

x_test scaler.transform(X_test) # нормируем тестовую выборку

Размерность векторов, формирующих признаковое описание, прямым образом влияет на время обучения модели. Определим размерность полученного выше представления.

x train. shape[1] # количество признаков, т.е. столбцов в двухмерном массиве

Для ускорения процесса обучения снизим размерность представления. Для этого с помощью VarianceThreshold исключим из рассмотрения те признаки, для которых значение дисперсии ниже заданного порога.

from sklearn. feature_selection import VarianceThreshold

selector arianceThreshold(threshold 1e-2) # инициализируем экземпляр класса VarianceThreshold

selector. fit(x_train) #расчёт дисперсии признаков на обучающей выборке

x train = selector.transform(x train) # отсев признаков с низкой дисперсией для объектов из обучающей выборки

Предсказание энергии атомизации гибридных перовскитов

В третьем примере, посвящённом предсказанию энергии атомизации гибридных перовскитов, информация о структуре соединений и их составе хранится в файле формата .rar содержащий экземпляры класса ase.atoms.Atoms из библиотеки Atomic Simulation Environment (ASE).

from ase. io import read # импортируём металл для чтения файлов

загружаем содержимое файла в виде списка объектов ace.atoms.Atoms

Structures-read("/content/drive/HyDrive/collabNotebooks/12/datasets/hop/structures.troy", index=":") # загружаем массив значений целевой переменной

ynp. load("/content/drive/Drive/Club Notebooks/12/datasets/hop/targets.npu")

am - display(y, benz-100) # строим гистограмму распределения целевой переменной

ax. set(xlabel='atomization energy, ev/atom') # подпись к оси абсцисс

Использование предоплаченных моделей (на примере Magnet)

На практике зачастую проще воспользоваться уже готовой моделью, а не создавать собственную с нуля. Широкий набор пред обученных моделей для предсказания свойств (не)периодических структур доступен в библиотеке MatErials Graph Network((Magnet).

В частности, имеется доступ к модели, прогнозирующей ширину запрещённой зоны для структур из базы данных Materials Project. MAE на тестовой выборке составляет 0.33 эВ. Но можно ли использовать подобную модель для предсказания целевого свойства структур, существенно отличающихся от тех, что формируют обучающую выборку**?** Проверим на примере металл-органических каркасных структур.

```
df=pd.read_csv("/content/drive/MyDrive/Com Notebooks/12/datasets/qm of/data.csv") # загружаем исходный файл с данными
y_test - dff "bandgap" ] values #извлекаем значения в ширины запрещенной зоны
structures_test -  # инициализируем список
for refcode in df ["refcode"] #заполняем список объектами
pymatgen.core. structure.structure
structuretest.append(Structure.fromfile(f"/content/drive/myDrive/Comnotebooks/12/datasets/mof/cimc/frefcode).cif"))
y_test_prod = model.product_structures(structure_test) # предсказываем значения с помощью предобученной модели
reg plot(y_test, y test prep.flatten(), label="bandgap", units="ev") # parity plot
get_regression_metrics(y_test, y_test_prep. flatten()) #регрессионные метрики
```

Использование векторных представлений химических элементов в качестве самостоятельного признакового описания

В результате обучения упомянутой выше нейронной сети на графах, Magnet, в нашем распоряжении оказываются векторные представления химических элементов, которые могут быть использованы в качестве отдельного метода представления.

Для формирования глобального представления, характеризующего структуру в целом, усредним векторные представления элементов, входящих в его состав.

Чтобы предварительно оценить, может ли такое представление быть полезным для создания предсказательных моделей,визуализируем векторные представления отдельных химических элементов. Для этого предварительно снизим размерность до двух с помощью алгоритма стохастического вложения соседей с t-распределением.

Убедившись, что близкие по свойствам элементы расположены близко и в низкоразмерной проекции их векторного представления, используем усреднённые векторные представления элементов для предсказания ранее рассмотренного свойства – энергии атомизации гибридных перовскитов.

Структуры сохранены в виде списка экземпляров класса ase.atoms.Atoms, однако для дальнейшей работы нам понадобятся объекты pymatgen.core.structure.Structure. Библиотека pymatgen содержит класс AseAtomsAdaptor, предназначенный именно для данного (а также обратного ему) преобразования.

Занятие 13. Лекция. Генеративные модели в химии и материаловедении

Какого основная цель химии, как современной науки? Основная цель состоит в составлении функционально взаимосвязи между структурой и свойствами химического соединения. Что из этих двух сущностей: структура и свойство, выступает в роли аргумента и переменной? В данном случае мы имеем две комбинации: Входные данные – строение, а выходные – свойства. А вторая комбинация, наоборот:

Рисунок 13.1. Схематичное изображение прямой задачи химии.

В первом варианте мы имеем дело с прямой задачей (структура – свойство). Для этого используются обычно дискриминативные модели. Большую часть информации о моделях подобного рода уже была изложена в рамках этого курса. Оставшаяся часть курса будет рассматривать генеративных моделей (обратная задача структура – свойства).

Обратную задачу можно решать с использованием трех групп методов:

1) Высокопроизводительного виртуального скрининга (вариация применения предсказательных моделей с некоторыми ухищрений)
2) Использование методов глобальной оптимизации
3) Генеративные модели (разнообразные архитектуры нейронных сетей)

Высокопроизводительный скрининг

Оценка численности стехиометрических соединений. В наше рассмотрение входит только те соединения, которые содержат 2, 3, 4 элемента. Максимальное значение стехиометрических коэффициентов будет 8. В рамках этих ограничений количество бинарных, тройных и четверных систем будет исчисляться почти 5 триллионами.

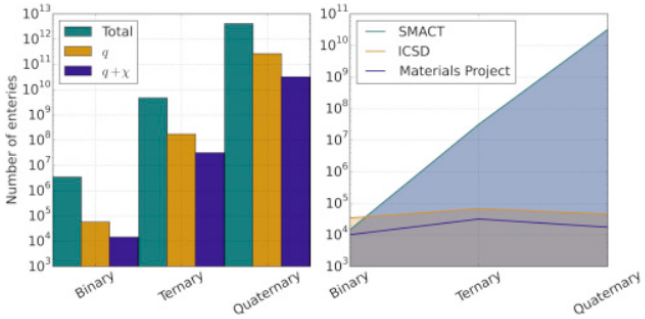


Рисунок 13.2. Пример высокопроизводительного скрининга.

Далеко не все комбинации данных химических составов далеко не всегда будут соответствовать в реальности существующих или синтезированных. Данное ограничение существует благодаря тому, что не все комбинации атомов и их степени окисления в сумме дают 0. Назовем данное ограничение q. Оно дает нам снижение выборки на более чем десятичный порядок. Если накладывать дополнительный фильтр на то, что электроотрицательность анионов выше электроотрицательности катионов, то численность выборки тоже снижается на порядок. Мы остаемся с 32 миллиардами химических соединений. Отметим, что в этом рассмотрений проигнорированы аморфные соединения и гибридные структуры. Таким образом мы не учитываем огромное количество структур. По этой причине высокопроизводительный скрининг уходит на второй план и уступает место другим вариантам обратного дизайна материалов.

Глобальная оптимизация

Пример, поиск сверхтвердых материалов. Задача сводилась к поиску параметров: постоянные решетки, координаты отдельных атомов на заранее определенной выборке кристаллических прототипов:

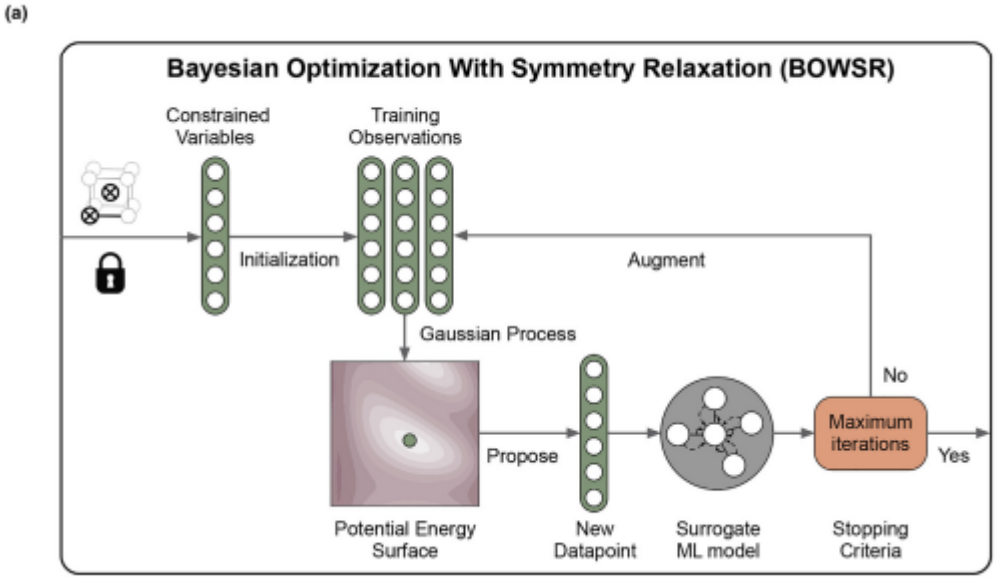


Рисунок 13.2. Схема работы глобальной оптимизации.

Оптимизация геометрии проводилась с помощью байесовской оптимизации (гауссовский процесс). Формулировка задачи (функция графов):

$$x \coloneqq \{a, b, c, \alpha, \beta, \gamma, \overrightarrow{c_1}, \overrightarrow{c_2}, \ldots\}; x_{opt} = argmin\, U(x), U: R^n \to R \qquad (13.1)$$

Данных пример считается совокупность трех задач: высокопроизводительного скрининга, глобальной оптимизации и генеративных моделей.

Схема генеративной модели новых материалов

Генеративные модели можно представить в виде следующей схемы:

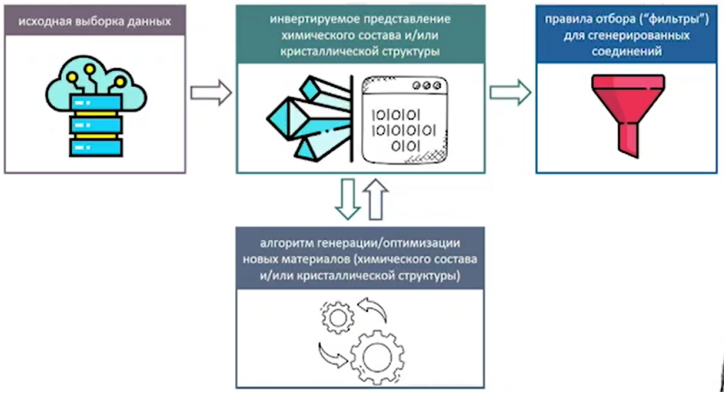


Рисунок 13.3. Схема генеративной модели новых материалов.

Первый компонент данной схемы – выборка данных. Исходя из этой выборки мы генерируем некое представление материалов, которое обладает инвертируемостью (возможность перевода данного метода представления в тот вид, который соответствует кристаллической структуре). В идеальном случае речь идет о параметрах ячейки и координатах атомов. Кроме того, данное представление должно быть пригодно для процесса оптимизации. Данный процесс осуществляется с помощью алгоритмов генерации/оптимизации новых материалов. Правила отбора (фильтры) представляют некий опциональный компонент, но обычно используется.

Рассмотрим примеры использования генеративной модели: генерация соединений произвольного химического состава:

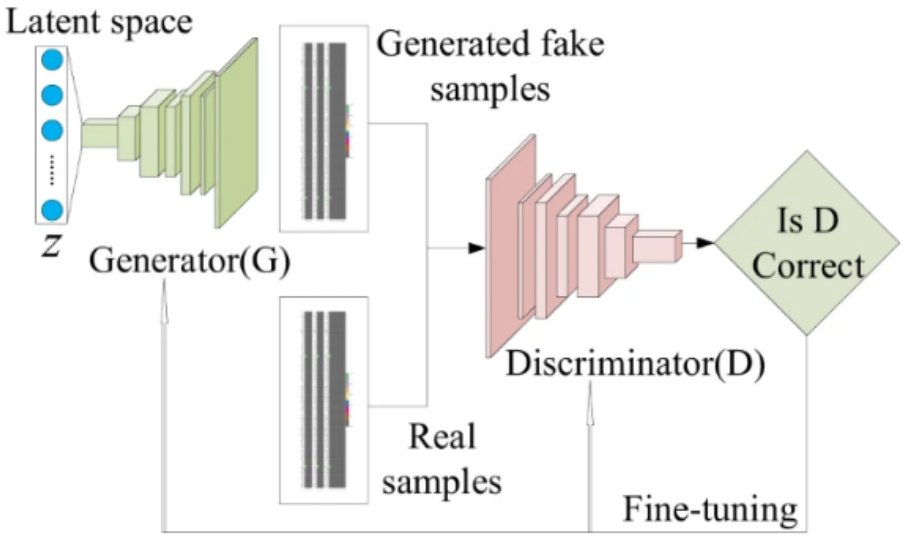


Рисунок 13.4. Пример использования генеративной модели.

В данном примере *исходная выборка данных:*

•291,884 соединения из OQMD
•65,922 соединения из MP
• 28,137 соединений из ICSO

инвертируемое представление:

унитарно-кодированные стехиометрические коэффициент: 2D-матрица размерностью 85*8

алгоритм генерации/оптимизации

генеративно-состязательная сеть Вассерштейна со сверточными и полносвязными слоями

правила отбора ("фильтры")

электронеитральность

баланс электроотрицательности

энергия образования

вариационный автокодировщик (вспомогательная сеть)

Другой *пример*, в котором закодировали не только химический состав, но и информацию о структуре соединения. Для этого можно использовать параметров ячейки и координат атомов. Данный способ кодирования информации о кристаллической структуре использовался в этом примере:

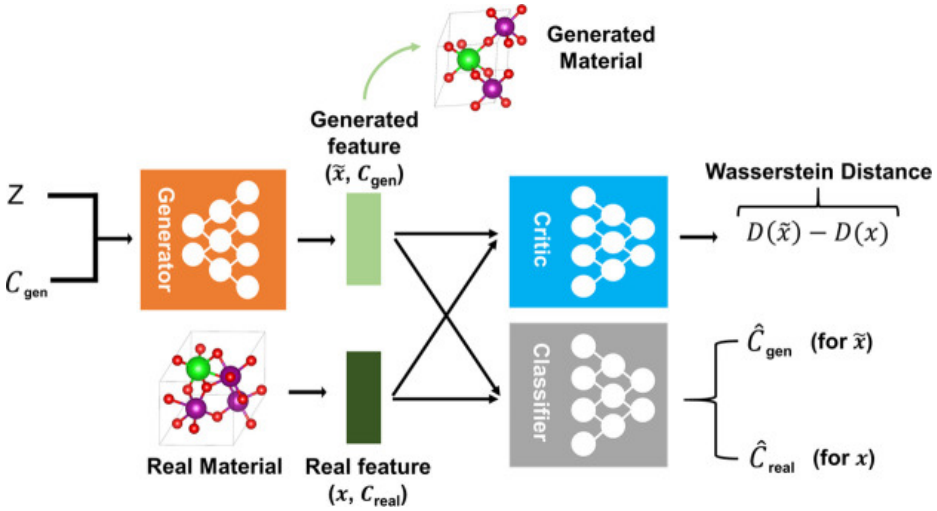


Рисунок 13.5. Пример кодирования структуры химического соединения.

исходная выборка данных

112.000 соединений $Mg_xMn_yO_z$

инвертируемое представление

дробные координат атомов и параметры ячейки

алгоритм генерации/оптимизации

генеративно-состязательная сеть Вассерштейна

правила отбора («фильтры»)

convex hull distance

электрохимическая устойчивость

Пример перевода вокселов в латентное пространство

Исходная выборка

10981 V_xO_y соединений (из MP и сгенерированное)

инвертируемое представление

воксельное представление элементарной ячейки и атомной плотности

снижение размерности с использованием автокодировщиков

алгоритм генерации/оптимизации

Сэмплирование латентного пространства условного вариационное автокодировщика:
из многомерного нормального распределение
линейная интерполяция на сфере
правила отбора («фильтры»)
стехиометрия
энергия образования

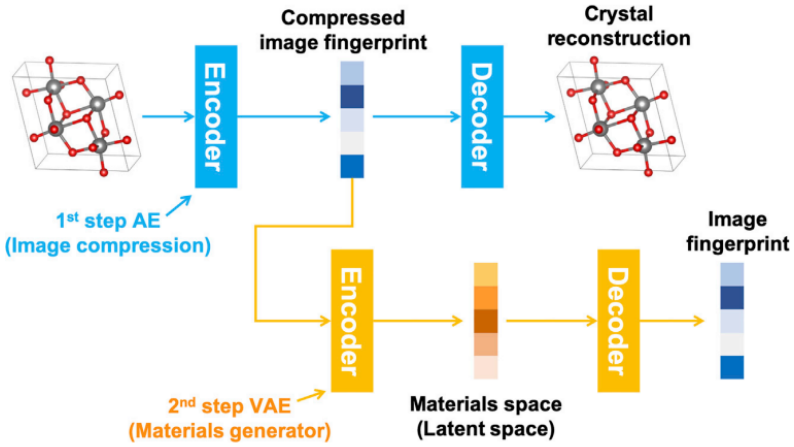


Рисунок 13.6. Схема автокодировщика.

Структурные признаки и их Фурье-сопряженные образы

В данной работе помимо информации, относящейся к отдельным атомам в струтуре, также была закодирована информация о кристаллических параметрах ячейки, заселенности узлов и физико-химических параметров:

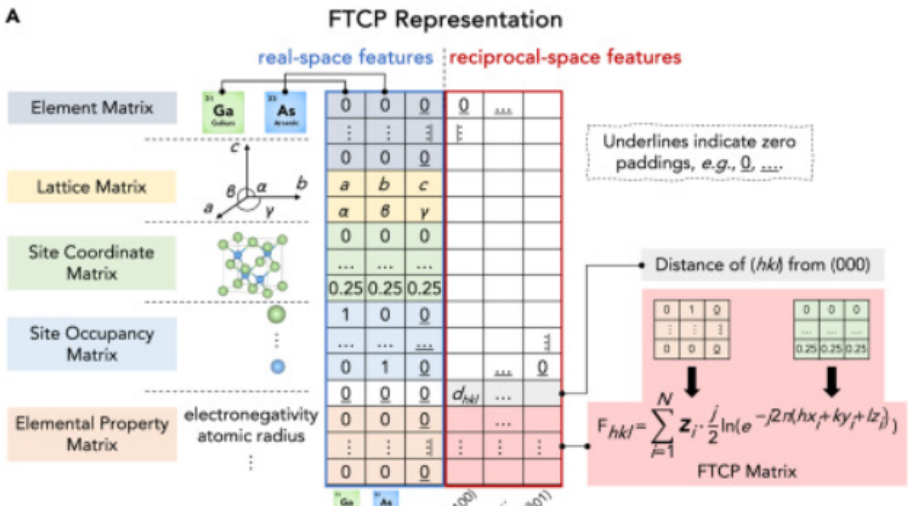


Рисунок 13.7. Работа нейронной сети с FTCP.

Авторы работы не стали ограничиваться вышеописанными дескрипторами, а также использовали Фурье-сопряженное образы (сгенерированы из физико-химических дескрипторов с использованием следующей аналитической формы, представленной на рисунке). Использование Фурье образов позволило улучшить качество воспроизведение

материалов и сходимость к тем значениям физико-химических свойств, которые интересны на практике.

исходная выборка данных

различные подмножества структур из MP

инвертируемое представление

структурное признаки в прямом и обратном (Фурье сопряженном) пространстве

алгоритм генерации/оптимизации

сэмплирование латентного пространства условного вариационного автокодировщика методом локального возмущения

правила отбора («фильтры»)

синтетическая доступность
энергия образования
convex hull distance

Байесовская оптимизация на конкатенированных представлениях

исходная выборка данных

133691 соединение из MP

инвертируемое представление

нормированные стехиометрические коэффициенты
дифрактограмма
перевод в континуальное представление пониженной размерности с помощью двух вариационных автокодировщиков

алгоритм генерации/оптимизации

байесовская оптимизация (метод Парзена)

правила отбора («фильтры»)

энергия образования
классификатор пространственной группы
электронейтральность
верификация дифрактограммы

Воксельное представление нанопористых материалов

исходная выборка данных

31,173 силикатных цеолита

инвертируемое представление

воксельные представления:

- атомная плотность кремния
- атомная плотность кислорода
- адсорбционный потенциал метана

алгоритм генерации/оптимизации

генеративно-состязательная сеть Вассерштейна с модифицированной функцией потерь (учет теплоты адсорбции)

правила отбора («фильтры»)

- энергий образования

• топологические характеристики

Обобщение

Обширные (курируемые) базы данных	**Универсальные инвертируемые представления**	**Разнообразные генеративные модели**
ChEMBL ZINC PubChem GDB-17 QM9 REAL	Последовательности символов SMILES SELFIES	Генеративно-состязательные сети
		Рекуррентные сети
	Графы Двухмерные Трехмерные	Автокодировщики
		Гибридные архитектуры

Таблица 13.1. Составляющие успеха обратного молекулярного дизайна

Занятие 14. Семинар. Генеративные модели в химии и материаловедении

Генерация распределения молекул с заданным свойством. Для генерации новых молекул мы воспользуемся подходом, предложенным в работе:

Nigam, A., Police, R., Krenn, M., dos Passos Gomes, G., & Aspuru-Guzik, A. (2021), Beyond generative models: super fast traversal, optimization,novelty, exploration and discovery (STONED) algorithm for molecules using SELFIES, Chemical science, 12(20), 7079-7090.doi.org/10.1039/D1SC00231G.

Исходная выборка данных составлена из молекул, входящих в базу ZINC. Используем подмножество 250 К молекул. Для начала импортируем содержимое файла с выборкой молекул:

```
import pandas as pd
df = pd.read_csv(os.path.join(FolDER, "datasets/zinc_250k_smiles.csv")) # загружаем содержимое .csv файла
```

Из полученного списка выбираем случайным образом 1000 молекул:

```
# случайная выборка 1000 строк SMILES в виде списка
init_smiles_list = df.sample(1000)['smiles'].values.to list()
#переводим строки SMILES в формат rd kit.Chem. rdchem. Mol
init tools_list = [Chem.MolfromSmiles(s) for s in init_smiles_list]
```

Вместо наиболее популярного строкового представления молекул (SMILES) мы воспользуемся Self-referencing embedded strings (**SELFIES**). Важной особенностью SELFIE является валидность произвольной строки данного формата, что позволяет генерировать новые молекулы с помощью операций, заданных на последовательности допустимых символов (словаря данного представления):

```
selfies_0 = encoder (smiles 0) # перевод SMILES в SELFIES
selfies_0
chars_selfie_0 = get_selfie_chart(series_0) # разбиение строки SELFIES на составляющие синполы
chars_selfie_0
```

Генерация новых молекул - "мутация" SELFIES строк

Ниже мы рассмотрим три операции, определенные на последовательности символов в SELFIES строке, и которые позволяют сгенерировать новые молекулы,

Insertion - вставка произвольного символа из словаря SELFIES на случайную позицию

```
# словарь допустимых Selfies сигналов
alphabet - list(selfies.got_semantic_robust_alphabet())
#случайный выбор позиции для добавленного символа
randon index np.random.randint(len(chars_selfie_0) + 1)
#случайный выбор символа SELFIES
random character - np.random.choice(alphabet, size-1)[0]
#вставка случайного SELFIES символа а исходную строку
```

selfie mutated_chars_o - chars_selfie_of:random_index] + [random character] + char_selfie_0[random index:]

Replacement - замена случайно выбранного символа в строке SELFIES на произвольный

```
# случайный выбор позиции для заменяемого символа
random_index - np.random.randint(len(chars_selfie_0))
# случайный пыбор симпола SELFIES
random_character = np.random.choice(alphabet, size=1)[0]
# взамена симпола в пыбранной позиции
if random index -0:
selfie_mutated_chars_0 = [random_character] + chars_selfie_0[random_index+1:]
else:
selfie mutated chars_0 - chars selfie_0[:random index] + [random character] + chars selfie_0[random index+1:]
```

Deletion - удаление случайно выбранного символа в строке SELFIES

```
# случайный выбор позиции для заменяемого символа
random index - np. random.randint(len(chars selfie 0))
# удаление символа в выбранной позиции
if random index 0:
selfie_mutated_chars_0 = chars_selfie_0[random_index+1:]
else:
selfie_mutated_chars_0=chars_selfie_0[:random_index]+chars_selfie_0[random_index+1:]
```

Теперь мы можем имплементировать функцию для "мутации" SELFIES строк с помощью трех рассмотренных выше операторов:

```
# вставки, замены и удаления символов.
def get new smiles(mol):
# для молекулы, поступиваей на вход, генерируем список неканонических строк SMILES
randomized_smile orderings-[randomize_smiles(mol) for in range(num_random_samples)]
# переводим SMILES в SELFIES
selfies_ls = [encoder(x) for x in randomized smile orderings]
#применяем num nutations раз случайным образом выбранные операторы мутации
selfies_mut - get_mutated_SELFIES(selfies_ls.copy(), num mutations num_mutations)
#конвертируем SELFIES обратно в SMILES
smiles back = [decoder(x) for x in selfies mut]
#инициализируем список для хранения сгенерированных SMILES в каноническом виде
canon_smi_is = []
#приведение сгенерированных SMILES в канонический вид
for ites in smiles back:
mol, smi_canon, did_convert = sanitize_smiles(iten)
```

```
if mol None or smi canon … or did convert … False:
raise Exception(" Invalid smile string found")
canon sai ls.append(smi canon)
canon_smi_ls - list(set(canon_smi_ls))
return canon smi ls
```

Оптимизируемое свойство - penalized logP

В данном примере мы рассмотрим лишь один аспект генерации молекул - оптимизацию (максимизацию) заданного свойства. На практике также следует учитывать следующие параметры выборки сгенерированных молекул: новизну, валидность, разнообразие и т.д. Детальный анализ качества работы генеративной модели может быть проведён с помощью специальных библиотек, таких как Guacamole и MOSES.

В качестве целевого свойства мы использовали penalized $logP$, который включает коэффициент липофильности $logP$. синтетическую доступность (synthetic accessibility) S.A и компонента cycle, накладывающую ограничения на размер макроциклов.

$$\text{penalized } log\ P\ =\ log\ P - SA - cycle$$

```
from sascorer import calculatescore as Molasses # импортируем функцию для расчёта синтетической доступности
from rd kit.Chem.Crippen import Moll op # импортируем функцию для расчёта коэффициента липофильности
def MolCycleScore(mol):
cycle_list = mol. GetRingInfo().AtomRings() # извлекаем циклы
large king size - max([len(size) font size in cycle list]) if cycle list else #определяем размер наибольшего цикла
cycle_score - max(targets ring size - 6, 0) # рассчитываем cycle penalty
return cycle_score
def get_penalized_logp(smiles):
mail - Chem.MolfromSmiles(smiles) #переводим строку SMILES в формат rdkit.Chem. rdchem.Mail
score - Malaga(mal) - Mal(mal) - MolCycleScore(mol) # рассчитыпаем penalized logP
return score
```

Оптимизация химического состава

Исходя из высокой размерности пространства поиска, как с формальной точки зрения (длина вектора признаков), так и с химической (многообразие химических соединений), простые подходы оптимизации (поиск по сетке, случайный поиск) потребуют крайне высокого числа итераций. По этой причине мы используем один из методов байесовской оптимизации.

```
from hyperopt import type, hp, fmin, STATUS_oK, Trials
# имплементируем функцию, значение которой необходимо минимизировать
def objective(elem dict):
```

```
        #инициализируем вектор атомных долей
        x = np.zeros(103)
        # заполняем его исходя из входных параметров (в виде словаря)
        for z, files dict.items():
        x[Element(z), z - 1)= f
        # нормализуем вектор, приводя сумму его компонентов к 1.0
        x = x / np. Sum(x)
        # возвращаем значение модуля сжатия, предсказанное ранее полученной моделью
        return {'loss': -model.predict([x])[0], 'status': STATus_ok)
        #имплементируем функцию для описания пространства поиска,
соответствукщего атомной доле одного элемента в диапазоне [0, 1]
        def jam space(z):
        return hp.uniform(Element. from_Z(z).symbol, 0.0, 1.0)
        # полное пространство поиска в формате словаря
        space - {Element. from_Z(z) symbol: elem space(z) for z in range(1, 104))
```

Оптимизация химического состава в латентном пространстве

Посредственный результат представленного выше подхода обусловлен в том числе разреженностью пространства поиска абсолютное большинство точек в нем не соответствует какому-либо стехиометрическому соединению. Это наводит на мысль о необходимости более "сжатого" пространства поиска, лишенного "мусорных" комбинаций параметров (атомных долей). Один из подходов для реализации подобного представления заключается в использовании планетного пространства вариационных автокодировщиков.

```
        import tensorflow as tf
        from keras.models import Model
        from keras.layers import Input, Dense, Lambda, Flatten
        from keras.metrics import binary cross entropy
input dim = 103
hidden dim = 64
latent dim = 32
def get_encoder(input_dim, hidden_dim, latent_dim):
x Input(shape (input_dim,))
hidden = Dense(hidden in, activation="relu")(x)
z_mean, z_log var - Dense(latent_dim)(hidden), Dense(latent_dim)(hidden)
        return Model(inputs x, outputs [z_mean, z_log var], name "encoder")
def get decoder(latent dim, hidden dim, input dim):
decoder input Input(shape (latent din,))
x = Dense(hidden_sim, activation "relu")(decoder_input)
x = Dense(input dim, activation 'sigmoid')(x)
        return Model(decoder_input, x, name="decoder")
def sampling(inputs):
2 mean, 2_log var inputs
```

```
batch_size tf.shape(z_mean)[0]
epsilon = tf.random.normal(shape (batch_size, latent_dim), mean-o., stddev-1.)
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
def get vae(input shape, encoder, decoder, sampling_layer):
input(shape-input shape, name="input")
z mean, z log var encoder(x)
z - sampling layer([z_mean, z log var])
x decoded - decoder(z)
vae = Model(x, x_decoded)
rec_loss - 1ei * input_dim " binary_cross entropy(Flatten()(x), Flatten()(x_decoded))
ki_loss - - 0.5 * tf.reduce_sum(1 + z log var - tf.square(z_ean) - tf.exp(z_log_var), axis--1)
vae_loss - tf.reduce_mean(rec_loss + kl_loss)
vae.add_loss(value_loss)
vae.compile(optimizer="adan")
        return value
```

Обучение будет проходить на выборке из базы данных material project. Главное не забыть провести разбиение выборки на обучающую и тестовую. С помощью автокодировщиков удается добиться существенного уменьшения функции потерь.

Занятие 15. Лекция. Основы параллельных вычислений.

Python – это в первую очередь интерпретируемый язык. Из это возникает основная проблема, он крайне нетороплив. Во вторую очередь, это язык более высокого уровня (можно реализовать многие команды всего в одну строчку). Кроме того, существует огромное количество библиотек, в которых практически все уже сделано за вас. Соответственно, можно взять уже готовое решение вместо того, чтобы придумывать решение самостоятельно.

Алгоритмы, плохой и хороший код

Интегрирование сектора, лишние операции. Рассмотрим пример:

Входные данные: положение центра, шаг по радиусу, размер сектора

Простой подход:

• Пересчитать координаты всех точек в полярные
• Пройти по всем точкам и выбрать нужные суммируя в массив
• ~16 млн сравнений

Разумный подход (ускорить вычисления)

• Пересчитать координаты всех точек в полярные
• Поскольку параметры сектора известны рассчитать координаты точки в столбце над сектором, суммировать пока не выйдем за пределы сектора
• Количество сравнений зависит от размера сектора (сектор 1* ~93 тысяч сравнений). Добавлено 2048 расчетный операций.

Другой пример: Поиск простых чисел, лишние операции

• Решение «в лоб»: делить на все числа до, смотреть остаток
• Решение каплю умнее: выкинуть все четные, использовать признаки делимости и т.д.
• Совсем разумно - использовать поиск готового решения: решето Эратосфена

Оптимизация времени выполнения python

Задача: перемножить два массива

Решение:

```
def mult_cycle ():
        for i in range(size):
         c[i] = a[i]*b[i]
def mult_direct ():
    c = a*b
def mult_numpy ():
     c = numpy. multiply (a,b)
```

Нужно отметить, что numpy может давать огромную оптимизацию кода и, соответственно, увеличить скорость программы в 100 раз.

Just-In-Time (JIT) компиляция

PyPy -трассировка в C

Numba -LLVM JIT компилятор, поддерживающий numpy

```
@jit(nopython=True,cache=True)
def single_cycle_np():
```

```
sum_val = 0
m a x v a l  = 0
min_val = 0
for i in range(size):
if (max_val < a[i]):
m a x _ v a l  = a[i]
 if ( m i n _ v a l  > a[i]):
min_val = a[i]
sum_val += a[i]
```

Какой код можно сделать параллельным?

Простой цикл:

```
for (int i=0;i<N;i++)
{c[i] = a[i]*b[i];}
```

Цикл с float параметром:

```
for (float a=0;a<N;a+=step)
{c[i] = b[i]*a;}
```

Сумма:

```
for (int i=0;i<N;i++)
{c += a[i];}
```

Конвейер:

```
for (int i=0;i<N-l;i++)
{a[i+l] += a[i];}
```

Закон Амдала

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}} \qquad (15.1)$$

Где α – доля от общего объема вычислений, которые могут быть получены только последовательным путем, p – число параллельных потоков, S_p – ускорение.

Закон показывает, насколько эффективно можно распараллелить код и эффективно ли нам это делать.

Python в параллельных вычислениях

Threading
• Конкурентный режим
• Быстрая инициализация
• Общая память потоков

Multiprocessing
• Создаются новые процессы
• Медленная инициализация
• Независимая память потоков

Numba (parallel=True)
• Экспериментальный режим
• Не поддерживает кэширование

MPI, OMP, pthread

POSIX threads (pthread)

• Запуск и управление потоками - задача программиста
• Как общая, так и собственная память у потоков
• Возможность запуска параллельных разнотипных задач

Open Multi-Processing (OMP)

• Простота использования
• «Общая» память потоков
• Количество потоков ограничено 1 узлом

Message Parsing Interface (MPI)

• Обеспечивает возможность использования нескольких узлов
• Поддержка гетерогенных систем
• Требуется обмен данными между потоками
• Ограничение со стороны шины передачи данных
• Гибридные варианты MPI/OMP, MPI/pthread

Отличия CPU и GPU

CPU

• Мало ядер но они производительные
• Доступ к RAM медленный
• RAM много
• Быстрый доступ к кешам (LI, L2, L3)

GPU

• Очень много ядер, но они медленные
• Вся память общая (ну почти), доступ для всех ядер быстрый
• Память заметно ограничена по объему
• Кеш есть но маленький, работа с ним своеобразна
• Очень муторный debug
• Низкая производительность FP64
• Необходимо копирование данных

Логика GPU, блоки, потоки

Как работает GPU? Данная программа работает по принципу SIMT, то есть одна инструкция выполняется одновременно сразу на множестве потоков (рис.15.1). Здесь есть ограничения в том, какие участки данных можно подавать одновременно, но это дает ограничения лишь в пару процентов, за которые не всегда стоит бороться.

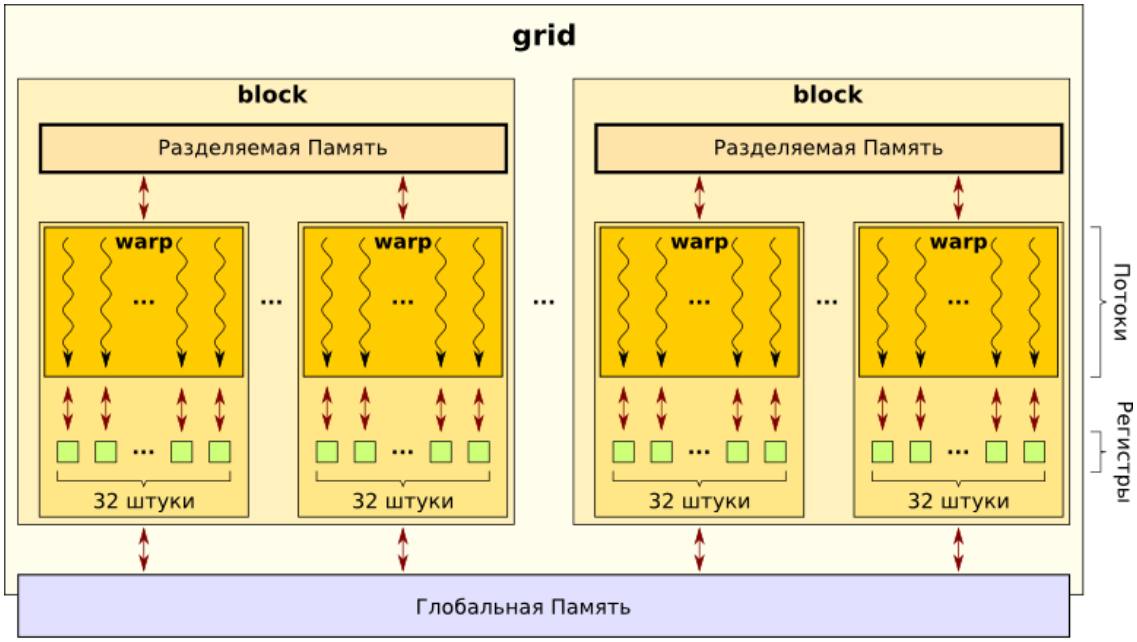


Рисунок 15.1. Иерархия потоков и доступ к частям памяти

Пример задачи: молекула глицерина ($C_3H_5(OH)_3$, есть координаты атомов в этой молекуле. Нужно посчитать расстояние между атомами.

Решение:

Вариант 1

```
for (int i=0;i<N;i++) { //parallel
    for (int j=0;j<N;j++) { //parallel
        if (i I=j)
          dist[i][j] = CalcDist(atom[i],atom[j]);}
```

Вариант 2

```
for (int i=0;i<N-1;i++) { //parallel
     for (int j-i+1;j<N;j*-*) ( //dependent
    dist[i][j] = CalcDist(atom(i],ato·n[ j]);}
```

1. Сложность N^2, макс. потоков N^2
2. Сложность $\frac{N^2}{2}$, макс. потоков N

Варианты переноса кода на GPU

pyCUDA

- Код встраивается напрямую в питон
- Необходимо преобразование типов переменных
- Минимум работы с памятью

OpenACC

- C/fortran
- Структура проекта практически не меняется
- Один компилятор (pci, ccc + offload) и он почти все делает за вас
- Прямая работа с памятью GPU и CPU, компилятор немного помогает

Native CUDA

- Код в отдельных файлах *.cu
- Отдельный компилятор (NVCC)
- Необходимо изменять структуру проекта
- Отдельно прописываются функции для «ядра»
- Прямая работа с памятью GPU и CPU

PyCUDA, пример

Инициализация:

```
import pycuda.autoinit
import pycuda.driver as drv
import nunpy
from pycuda.compiler import SourceModule
```

Код CUDA

```
 mod = SourceModule (***
_global_vold multiply_them(float *dest, float *a, float *b, int* size)
{
 const int i = threadIdx.x + blockDim.x * blockIdx.x;
if (i<size[o])
   dest[i] = a[i] * b[i];
}
***)
```

Загрузка функции

```
Multiply_them = mod.get_function("multiply_them")
```

Инициализация массивов

```
a = numpy.random.randn(size).astype(numpy.float32)
Ь = numpy.random.randn(size).astype(numpy.float32)
dest = numpy.zeros_like(a)
```

Запуск:

```
multiply_them(
drv.Out(dest), drv.In(a),
drv.In(b),drv.In(sent_size),
block=(400,1,1), grid=(250000,1))
```

Такая программа позволяет получать прирост производительности

Занятие 16. Семинар. Основы параллельных вычислений.

По умолчанию Numpy отправляет вычисления в высокоэффективные реализации BLAS (basic linear algebra subprograms) и LAPACK(Linear Algebra PACKage). BLAS и LAPACK – это библиотеки линейной алгебры, написанные на C и Fortran. Они поддерживаются группами высококвалифицированных разработчиков, способных получить от CPU максимально возможную скорость работы, используя низкоуровневые команды.

Использование JIT-компиляторов

Python код транслируется в LLVM байткод (Low Level Virtual Machine):

```
from numba import jit
mandelbrot_numba = njit(mandelbrot_python)
Xtime _ = mandelbrot_numba(size, iterations)
Xtime_ = mandelbrot_numba(size, iterations)
# в отличии от Xtime, два процента активируют операцию ко всей ячейки
# кроме того, timeit в отличии от time позволяет получить дополнительные сведения
 Xtimeit mandelbrot_numba(size, iterations)
```

jet = jet(nopython=True) - запрет использовать нетранслируемый код в байткод, так как nubma еще не поддерживает всех возможностей python (и numpy). Если встретится неподдерживаемая операция, @njit выбросит исключение, а @jit перейдет в обычный режим интерпретатора python до конца функции.

Сохранение байт кода на диск, кэширование

Прогрев коша - запуск функции, чтобы дать понять компилятору тип переменных:

```
@njit
def f(x, y):
      return x + y
f(1,1)
f.signatures
f.inspect_types()
```

При запуске запомнятся типы входных данных, функция сохранится на диск. Возможно придется установить переменную среды NUMBA_CACHE_DIR:

```
@njyt(cache=True)
def f(x, y):
return x + y
```

Ручное указание типов переменных

```
from numba import int16, int32
@njit(int32(int16, int16))
def f(x, j):
return x + y
f.inspect_types()
```

Указание локальных типов переменных:

```
@njit(int32(int16, int16), locals={'z': int32})
def f(x, y):
z = y + 10
return x + z
```

Превращение скалярной функции в векторную

```
from number import vectorize, int64
test_array = list(range(1000000))
def scalar_func(value):
If value % 2 0:
return 2
else:
return 1
x time it [scalar_func(x) for x in test_array]
@vectorize([int64(int64)))
def scalar_func(value):
if value % 2 == 0:
return 2
else:
return 1
Xtimeit scalar_func(test_array) # на google colab старая версия numba, работаетс лишь с float64
```

Параллельные вычисления

```
import time
def heavy_calculation():
print("Начали выполнение тяжелой задачи")
Time.sleep(2)
print("Закончили выполнение тяжелой задачи")
start = time.perf_counter()
heavy_calculation()
heavy_calculation()
finish = time.perf_counter()
print(f"Время выполнения: {round(finish-start, 2)} с")
```

GIL – global interpreter lock. Гарантирует то, что команда не будет теряться при выполнении нескольких команд. Способ синхронизации потоков, который позволяет избежать конфликтов при одновременном обращении разных потоков к одному участку памяти.

Multiprocessing и Multithreading

Multiprocessing:

- Новые процессы создается без зависимостей от основного
- Тяжеловесны, медленно запускаются
- Память одного процесса недоступна для других процессов

• По GIL на каждый процесс
• Инструменты синхронизации используются редко
Threading:
• Новые треды создаются в одном процессе
• Легковесны, быстро инициализируются
• Память доступна для всех тредов
• Один GIL на все треды
• Часто используются инструменты синхронизации (мьютексы)
Пример с аргументом тяжелой функции

```
Writefile multiproc.py
import multiprocessing as mp
import time
def heavy_calculation(duration):
    print(f"Начали выполнение тяхелой задачи: {duration} c")
    time.sleep(duration)
    print(f"Закончили выполнение тяжелой задачи: {duration} c")
if __name__** __main__:
start = time.perf_counter()
procs = []
for i in (x/2.0 for x in range(3, 8)):
p = mp.Process(target=heavy_calculation, args=[i])
p.start()
procs.append(p)
for p in procs:
p.join()
finish = time. perf_counter()
print(f"Время выполнения: {round(finish-start, 2)} c")
```

Посмотрим на время работы скрипта в зависимости от количества процессов (результаты будут отличаться от используемого ЦПУ):

```
import numpy as np
from multiprocessing import Pool
if _(variable) test_cases: list[int]
test_cases = list(range(1000000))
for n in range(1,8):
start = time perf_counter()
with Pool(processes=n) as WorkerPool:
WorkerPool. map(np.square, test_cases)
finish = time perf_counter()
print(f"время выполнения на пуле из {n} воркеров: (round(finish-start, 2)} c")
```

Мультитрелинг

Синтаксис аналогичен мультипроцессингу.

```
from threading import Thread
import time
def heavy_calculation(duration):
print(f"Начали выполнение тяхелой задачи: {duration} c")
time.sleep(duration)
print(f"Закончили выполнение тяжелой задачи: {duration} c")
start = time.perf_counter()
procs = []
for i in (x/2.0 for x in range(3, 8)):
p = Thread(target=heavy_calculation, args=[i])
p.start()
procs.append(p)
for p in procs:
p.join()
finish = time.perf_counter()
print(f"Время выполнения: {round(finish-start, 2)} c")
```

Можно ли еще быстрее? Доспустим у нас не 1000, а 10^6 операций, и мы хотим их выполнить быстро. Есть возможность увеличить производительность процессоров (но в настоящее время существует замедление производительности процессоров, и мы упираемся в физические ограничения). Один из вариантов, как можно ускорится – перейти на другую архитектуру. Например, выполнять наши действия не на процессоре, а на видеокарте.

Занятие 17. Лекция. Применение методов искусственного интеллекта в химии.

Краткий список того, что успели осветить в данном курсе:

- python3
- Анализ данных
- Методы машинного обучения
- Представление структур
- Нейронные сети
- Алгоритмы глобальной оптимизации
- Генеративные модели
- Оптимизация кода

Применение алгоритмов:

- Анализ данных
- QSAR/QSPR/QS...R
- Планирование экспериментов/оптимизация технологических процессов
- Оптимизация геометрии
- De novo design
- Создание новых численных методов (квантово-химические, атомистические, etc.)
- Обработка данных (спектральные, другие приборные, etc.)
- Автоматизированный сбор данных (NLP)
- Реакции, условия синтеза, ретросинтетический анализ, обход патентов
- Quantum machine learning

Новые квантово-химические методы (DFT)

Можно использовать методы глобальной оптимизации для разработки новых функционалов теории функционалов плотности. В примере использовали байесовскую оптимизацию с парсоновскими оценщиками для того, чтобы с существующими базами данных можно было перепараметризовать известные функционалы или искать какие-то комбинации известных функционалов для решения узкой задачи (здесь найти функционал для работы с актиноидами (менее изученная область химии). В данном случае использовали метод глобальной оптимизации, при этом выбор времени одного шага был ключевым.

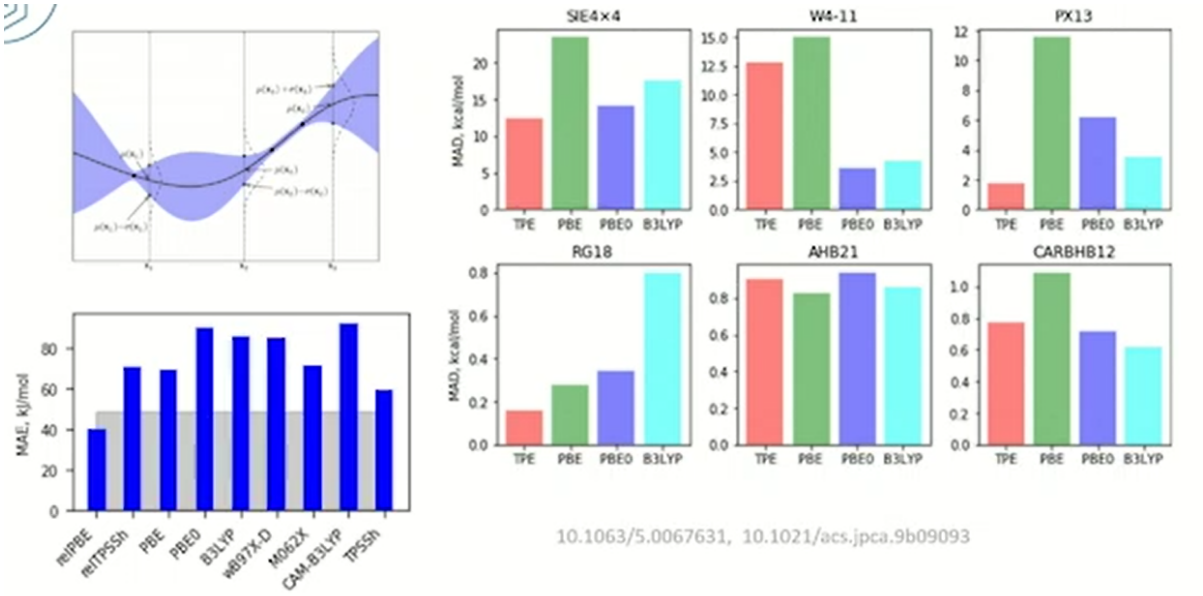


Рисунок 17.1. Пример оптимизации с парсоновскими оценщиками.

Новые атомистические методы

Поиск потенциала межатомного взаимодействия сейчас крайне актуален. И данную характеристику все чаще и чаще ищут с использованием методов машинного обучения и искусственного интеллекта. Например, в работе рассматривали переход от идеальных кристаллических структур к структурам более реальным:

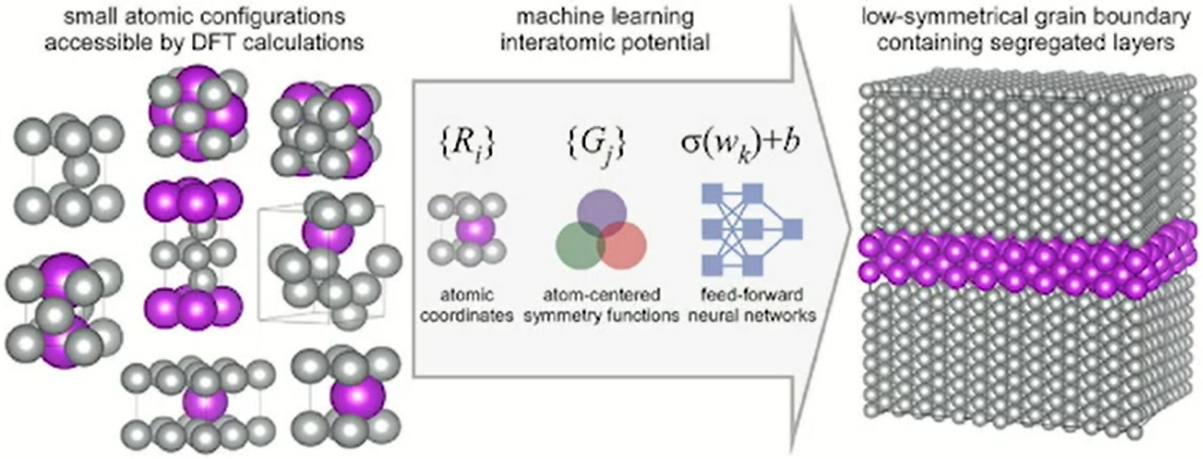


Рисунок 17.2. Пример перехода от идеальных структур к реальным.

А именно к таким структурам, где есть дефекты и изучали границы зерен и взаимодействие на границе зерен. Готовых потенциалов взаимодействия для того, чтобы моделировать эти свойства для некоторого интересующего класса соединений не существовало и в примере использовали совокупность квантово-механических расчетов в маленьких ячейках для имитации различных окружений. А также использовалось машинное обучение с использованием признаковых описаний, простых архитектур прямого распространения для оптимизации новых потенциалов взаимодействия.

ANN – reactions

Можно пойти немного дальше и предсказывать не просто свойства химического соединения, но и ход химической реакции. И предсказывать на основании машинного обучения выход реакции, иногда продукт или наоборот, из чего синтезировать какое-либо соединение. В данном подходе используются различные нейронные сети. Кроме того, сеть может помочь подобрать условия синтеза, оценки того, получится ли желательное соединение.

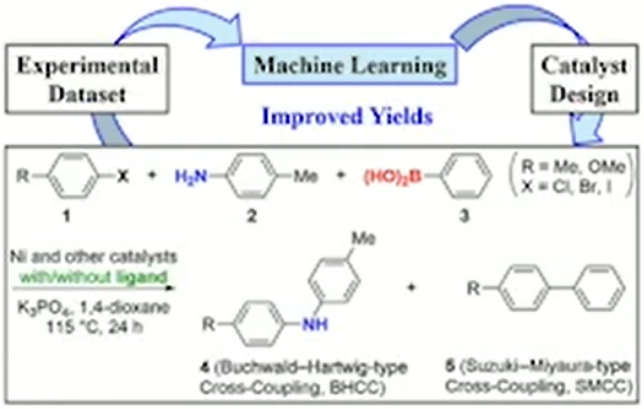


Рисунок 17.3. Пример обратной задачи в химии.

Natural language processing – сбор баз данных

Отдельно активно используются методы, связанные с обработкой естественного языка с использованием многочисленных архитектур нейронных сетей (с текстовыми файлами). При этом они могут быть использованы на разных этапах, в первую очередь для сбора базы данных. И далее собрать цельную картину в определенной области:

Рисунок 17.4. Пример использования обработки естественного языка.

Данные базы данных уже можно обрабатывать с человеческим участием (нивелировать ошибки), анализировать и делать какие-то выводы. В совокупности это позволяет гораздо быстрее, эффективнее и точнее обрабатывать информацию. Существуют решения, которые позволяют формировать базы данных.

Семантические сети

Семантические сети используются не для формирования баз данных, а для анализа данных и выявления общих закономерностей. Данные могут быть разные: экспериментальные, клинические случаи болезни. Можно найти семантические связь

между клиническим случаем и лекарством. Можно натренировать такие сети на уже известных специфичных лекарствах и на основании их искать неизвестные пары лекарство/болезнь. Это не генеративная модель для создания нового лекарства, а лишь способ обобщения информации.

Можно развить данную концепцию и усложнить вопросы, которые нас интересуют. По сути, размечать текст в соответствии с тем, где мы ожидаем увидеть ответ на вопрос. Например, искать те фрагменты, которые отвечают формуле, составу, продуктам, реагентам, условиям синтеза. Размечая текст на такие «сущности», можно получать короткую выжимку данных. При этом ускоряется процесс обработки данных.

Что делать, если данных не хватает?

В рамках одной экспериментальной группы сложно осуществить миллион экспериментов в одних и тех же условиях. Обычно имеется десяток или сотня точек данных с какими-либо характеристиками. Можно использовать QSPA модели (характеристика по структуре). Обычно она натренирована на большом количестве данных. И ее можно перетренировать только на последних слоях (уровень абстрактизации высокий) на маленьком количестве данных.

Компьютерное зрение (Computer vision)

Можно натренировать модель распознавать объекты, которые используются. В лаборатории обычно это не нужно, но, если работа сопряжена с радиоактивными элементами или чем-то опасным, это может быть полезно. И разработка помощника, который будет понимать, где-что находится и управлять этим, становится актуальной.

ХИМИЧЕСКИЙ
ФАКУЛЬТЕТ
МГУ ИМЕНИ
М.В. ЛОМОНОСОВА

teach-in
ЛЕКЦИИ УЧЕНЫХ МГУ