Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

К.И. Дементьева, А.А. Ракитский

Методы машинного обучения

Учебно-методическое пособие

Новосибирск
2023

УДК 002(075.8)

Утверждено редакционно-издательским советом СибГУТИ

Рецензент Перышкова Е.Н.

Ракитский А.А., Дементьева К.И. Методы машинного обучения: Практикум / Сибирский государственный университет телекоммуникаций и информатики; каф. Прикладной математики и кибернетики. – Новосибирск, 2023. – 35 с.

Данное учебно-методическое пособие включает в себя необходимые теоретические материалы по 4 лабораторным работам для изучения дисциплины «Методы машинного обучения» для направления подготовки 09.03.01 «Информатика и вычислительная техника» факультета Информатики и вычислительной техники. В пособии рассматриваются наиболее распространённые методы классификации, анализа и восстановления регрессии. Рассматриваются такие методы, как: линейная регрессия, метрические классификаторы и логические классификаторы, а также нейронные сети для задач распознавания объектов на изображениях, предсказание цены на недвижимость и т.д. Приводятся пояснения по применению указанных методов на практике и доказательства их корректности. Кроме того, описаны рекомендации и указания по выполнению лабораторных работ, связанных с данными темами.
Методические указания рекомендованы для студентов технических специальностей, изучающих методы машинного обучения на 4 курсе бакалавриата.

Содержание

Введение

В настоящий момент времени методы машинного обучения являются наиболее перспективным и развивающимся направлением в сфере информационных технологий. При помощи машинного обучения автоматизируются такие процессы, которые раньше невозможно было представить без человека: автопилоты для различных видов транспорта и техники, боты, выполняющие функции операторов поддержки, системы распознавания и классификации данных и многое другое. Очевидно, что для дальнейшего развития данного направления, необходимо готовить специалистов в области машинного обучения. Изучение дисциплины «Методы машинного обучения» является важной частью подготовки высококвалифицированных кадров технических специальностей. В представленном практикуме рассматриваются основные методы, использующиеся в машинном обучении, такие как линейная регрессия, метрические и логические классификаторы. Рассматриваются особенности этих методов как с точки зрения их непосредственной реализации, так и с точки зрения формирования признаков обучающей выборки. Кроме того описаны основные средства, используемые при реализации указанных методов, такие как пакет для обработки данных R и язык программирования Python. Основной целью ставится изучение базовых методов машинного обучения и применение их на реальных задачах.

1.Установка средств разработки Python

Курс основан на версии Python 3.7.0. Скачать необходимые средства можно на сайте https://www.python.org/. Установка на Windows/macOS не отличается от стандартной установки программы, за исключением того, что на первом экране необходимо указать флаг «Add Python 3.7 to PATH».

Рисунок 1.1 – Окно при установке Python.

После установки вам будет доступен интерпретатор языка Python через терминал (если не забыли поставить галочку), с помощью команды python

```
python
```

На Linux-base системах можно произвести установку пакетов python3, python3-pip (менеджер модулей Python) с помощью менеджера пакетов, что будет являться эквивалентом установки, представленной выше.
Так же программировать на Python можно использовать такие среды разработки как:

- Spyder
- The Jupyter Notebook
- PyCharm

Для перечисленных интегрированных сред разработки необходимо установить библиотеки Python.

Установка библиотек Python Установка библиотек, необходимых на протяжении курса, производится с помощью утилиты pip, которая является системой управления пакетами. Установка производится с помощью следующей команды:

```
pip install <имя пакета>
```

Примечание: если терминал выдал предупреждение или ошибку о том, что используется старая версия pip, то необходимо выполнить следующую команду:

```
python -m pip install –upgrade pip
```

Необходимые пакеты:

NumPy (pip install numpy) – библиотека, содержащая множество математических алгоритмов, такие как работа с матрицами, последовательностями, срезами и т.п.

Pandas (pip install pandas) – библиотека для обработки и анализа данных.

Matplotlib – библиотека для визуализации данных двумерной (2D) графикой (3D графика также поддерживается.

2. Теория программирования на языке Python

Списки

В языке Python существуют составные типы данных, использующихся для группировки прочих значений вместе. Наиболее популярный из них — список (list). Элементы списка могут быть разных типов. Пример задания списка:

```
>>> l = ['cats', 'frogs', 130, 54321]
>>> l
['cats', 'frogs', 130, 54321]
```

Подобно индексам в строках, индексы списков начинаются с нуля, списки могут быть срезаны, объединены и так далее:

```
>>> l[0]
'cats'
>>> l[3]
54321
>>> l[-3]
'frogs'
>>> l[1:-1]
['frogs', 130]
>>> l[:2] + ['banana', 3*3]
['cats', 'frogs', 'banana', 9]
>>> 3*l[:3] + ['Boom!!!']
['cats', 'frogs', 130, 'cats', 'frogs', 130, 'cats',
'frogs', 130, 'Boom!!!']
```

В списке, в отличие от неизменяемых строк, возможно изменить индивидуальные элементы списка:

```
>>> l = ['cats', 'frogs', 130, 54321]
>>> l[2] = l[2] + 23
>>> l
['cats', 'frogs', 153, 54321]
```

Изменить размер списка или полностью его очистить возможно с помощью присваивания срезу:

```
>>> # Заменим некоторые элементы:
... l[0:2] = [13, 15]
>>> l
```

```
[13, 15, 153, 54321]
>>> # Удалим первые 2 элемента:
... l[0:2] = []
>>> l
[153, 54321]
>>> # Вставим несколько значений начиная с 1го индекса:
... l[1:1] = ['qwerty', 'hohoho']
>>> l
[153, 'qwerty', 'hohoho', 54321]
>>> # Вставка копии списка в начало
>>> l[:0] = l
>>> l
[153, 'qwerty', 'hohoho', 54321, 153, 'qwerty', 'hohoho',
54321]
>>> # Замена всех значений пустым списком (очистка)
>>> l[:] = []
>>> l
[]
```

Встроенная функция len() применима к спискам для нахождения количества элементов:

```
>>> l = ['q', 'w', 'e', 'r', 't', 'y']]
>>> len(l)
6
```

Возможно встраивание списка в другой:

```
>>> k = [1, 7]
>>> l = [1, k, 4]
>>> len(l)
3
>>> k[1]
[1, 7]
>>> l[1][0]
1
```

Добавление элемента в конец списка:

```
>>> l[1].append('dogs')
>>> l
```

```
[1, [1, 7, 'dogs'], 4]
```

Оператор if

```
  if x < 100:
  ...       x = 100
  ...       print(Значение меньше ста, изменено на 100')
  ... elif x == 100:
  ...       print('Сто')
  ... else:
  ...       print('Больше ста')
```

Блок else не является обязательным, запись elif является сокращенной записью else if.

Оператор for

Оператор `for` в Python отличается от оператора for в C/C++. Без указания шага итерации и возможности оператор `for` в Python проходит по всем элементам любой последовательности в том порядке, в котором они в ней располагаются. Например:

```
... a = ['one', 'two', 'three']
>>> for i in a:
...     print(i)
```

Для перебора последовательности чисел следует использовать встроенную функцию `range()`. С её помощью генерируются арифметические прогрессии в формате списка:

```
>>> for num in range(5):
...    print(num)
```

Последний элемент интервала **никогда** не включается в сгенерированный список. Примеры вызова функции:

```
range(3, 11)
   3, 4, 5, 6, 7, 8, 9, 10

range(0, 30, 3)
   0, 3, 6, 9, 12, 15, 18, 21, 24, 27

range(-20, -100, -30)
```

```
    -20, -50, -80
```

Определение функции

Зарезервированное слово def осуществляет определение функции. За ним следуют имя функции и заключённый в скобки список параметров. Выражения, составляющие тело функции, начинаются со следующей строки и должны иметь отступ.

```
>>> def fib(n):# вывести числа Фибоначчи вплоть до n
...     a, b = 0, 1
...     while b < n:
...         print(b, end=' ')
...         a, b = b, a+b
>>> # Вызов функции:
... fib(2000)
```

Параметры функции (один или несколько) могут быть заданы по умолчанию. В итоге создаётся функция, которая может быть вызвана с меньшим количеством параметров, чем в её определении: при этом неуказанные при вызове параметры примут значения, данные в определении функции. Например:

```
def repeat(message, times = 1):
    print(message * times)

repeat('Hello!')
#Hello!
repeat('Hi', 5)
#HiHiHiHiHi
```

Контрольные вопросы

1. Дан список L = ['один', 'два', 'три', 'десять', 'пятнадцать']. Каков будет вывод при выполнении функции print(L[-1])?
2. Какие существуют параметры функции range()?
3. Осуществите реализацию функции factorial(n) нахождения факториала числа n. Каков вывод будет при вызове factorial(23)?

3. Метрические классификаторы

3.1 Метод k ближайших соседей

Теоретический базис:

Пусть на множестве объектов X задана функция расстояния $\rho: X \times X \to [0, \infty)$. Существует целевая зависимость $y^*: X \to Y$, значения которой известны только на объектах обучающей выборки $X^\ell = (x_i, y_i)_{i=1}^\ell, y_i = y^*(x_i)$. Множество классов Y конечно. Требуется построить алгоритм классификации $a: X \to Y$, аппроксимирующий целевую зависимость $y^*(x)$ на всём множестве X.

Для произвольного объекта $u \in X$ расположим элементы обучающей выборки $x_1, \ldots, x_\ell$ в порядке возрастания расстояний до u: $\rho(u, x_u^{(1)}) \leq \rho(u, x_u^{(2)}) \leq \cdots \leq \rho(u, x_u^{(\ell)})$, где через x_u^i обозначается i-й сосед объекта u. Соответственно, ответ на i-м соседе объекта u есть $y_u^i = y^*(x_u^i)$. Таким образом, любой объект $u \in X$ порождает свою перенумерацию выборки.

Определение: *Метрический алгоритм классификации с обучающей выборкой X^ℓ относит объект u к тому классу $y \in Y$, для которого суммарный вес ближайших обучающих объектов $\Gamma_y(u, X^\ell)$ максимален:*

$$a(u; X^\ell) = \arg\max_{y \in Y} \Gamma_y(u, X^\ell); \; \Gamma_y(u, X^\ell) = \sum_{i=1}^\ell [y_u^i = y] w(i, u); \qquad (3.1.1)$$

где весовая функция $w(i, u)$ оценивает степень важности -го соседа для классификации объекта u. Функция $\Gamma_y(u, X^\ell)$ называется оценкой близости объекта u к классу y.

Алгоритм k ближайших соседей (k nearest neighbors, kNN).

Чтобы сгладить влияние выбросов, будем относить объект u к тому классу, элементов которого окажется больше среди k ближайших соседей $x_u^i, i = 1, \ldots, k$:

$$w(i, u) = [i \leq k]; \; a(u; X^\ell, k) = \arg\max_{y \in Y} \sum_{i=1}^k [y_u^i = y]. \qquad (3.1.2)$$

При $k = 1$ этот алгоритм совпадает с предыдущим, следовательно, неустойчив к шуму. При $k = \ell$, наоборот, он чрезмерно устойчив и вырождается в константу.

Таким образом, крайние значения k нежелательны. На практике оптимальное значение параметра k определяют по критерию скользящего контроля с исключением объектов по одному (leave-one-out, LOO). Для каждого объекта $x_i \in X^\ell$ проверяется, правильно ли он классифицируется по своим k ближайшим соседям.

$$LOO\left(k, X^{\ell}\right) = \sum_{i=1}^{\ell}\left[a\left(x_i\,;\, X^{\ell}\setminus\{x_i\}, k\right) \neq y_i\right] \to \min_{k} \quad . \qquad (3.1.3)$$

Заметим, что если классифицируемый объект x_i не исключать из обучающей выборки, то ближайшим соседом x_i всегда будет сам x_i , и минимальное (нулевое) значение функционала $LOO(k)$ будет достигаться при $k = 1$. Существует и альтернативный вариант метода kNN: в каждом классе выбирается k ближайших к u объектов, и объект u относится к тому классу, для которого среднее расстояние до k ближайших соседей минимально.

3.2 Алгоритм k взвешенных ближайших соседей.

Недостаток kNN в том, что максимум может достигаться сразу на нескольких классах. В задачах с двумя классами этого можно избежать, если взять нечётное k. Более общая тактика, которая годится и для случая многих классов — ввести строго убывающую последовательность вещественных весов w_i , задающих вклад i-го соседа в классификацию:

$$w(i,u) = [i \leq k]w_i\,;\; a\left(u;\, X^{\ell}, k\right) = arg\max_{y\in Y} \sum_{i=1}^{k}\left[y_u^{(i)} = y\right]w_i\,. \quad (3.2.1)$$

Выбор последовательности w_i является эвристикой. Если взять линейно убывающие веса $w_i = \frac{k+1-i}{k}$, то неоднозначности также могут возникать, хотя и реже (пример: классов два; первый и четвёртый сосед голосуют за класс 1, второй и третий — за класс 2; суммы голосов совпадают). Неоднозначность устраняется окончательно, если взять нелинейно убывающую последовательность, скажем, геометрическую прогрессию: $w_i = q^i$, где знаменатель прогрессии $q \in (0,1)$ является параметром алгоритма. Его можно подбирать по критерию LOO, аналогично числу соседей k.

3.3 Метод парзеновского окна

Ещё один способ задать веса соседям — определить w_i как функцию от расстояния $\rho(u, x_u^{(i)})$, а не от ранга соседа i. Введём функцию ядра $K(z)$, невозрастающую на $[0,\infty)$. Положив $w(i,u) = K(\frac{1}{h}\,\rho\left(u, x_u^{(i)}\right))$ в общей формуле, получим алгоритм

$$a\left(u;\, X^{\ell}, h\right) = arg\max_{y\in Y}\sum_{i=1}^{\ell}\left[y_u^{(i)} = y\right]K\left(\frac{\rho\left(u, x_u^{(i)}\right)}{h}\right) \qquad (3.3.1)$$

Параметр h называется шириной окна и играет примерно ту же роль, что и число соседей k. «Окно» — это сферическая окрестность объекта u радиуса h, при попадании в которую обучающий объект x_i «голосует» за отнесение

объекта u к классу y_i . Мы пришли к этому алгоритму чисто эвристическим путём, однако он имеет более строгое обоснование в байесовской теории классификации, и, фактически, совпадает с методом парзеновского окна. Параметр h можно задавать априори или определять по скользящему контролю. Зависимость $LOO(h)$, как правило, имеет характерный минимум, поскольку слишком узкие окна приводят к неустойчивой классификации; а слишком широкие — к вырождению алгоритма в константу.

Фиксация ширины окна h не подходит для тех задач, в которых обучающие объекты существенно неравномерно распределены по пространству X. В окрестности одних объектов может оказываться очень много соседей, а в окрестности других — ни одного. В этих случаях применяется окно переменной ширины. Возьмём финитное ядро — невозрастающую функцию $K(z)$, положительную на отрезке $[0,1]$, и равную нулю вне его. Определим h как наибольшее число, при котором ровно k ближайших соседей объекта u получают ненулевые веса: $h(u) = \rho(u, x_u^{(k+1)})$. Тогда алгоритм принимает вид

$$a(u;\, X^\ell, k) = arg\max_{y \in Y} \sum_{i=1}^{k} [y_u^{(i)} = y]\, K\left(\frac{\rho\left(u, x_u^{(i)}\right)}{\rho\left(u, x_u^{(k+1)}\right)}\right) . \qquad (3.3.2)$$

Заметим, что при финитном ядре классификация объекта сводится к поиску его соседей, тогда как при не финитном ядре (например, гауссовском) требуется перебор всей обучающей выборки.

Выбор ядра следует осуществлять из вариантов, представленных на рисунке:

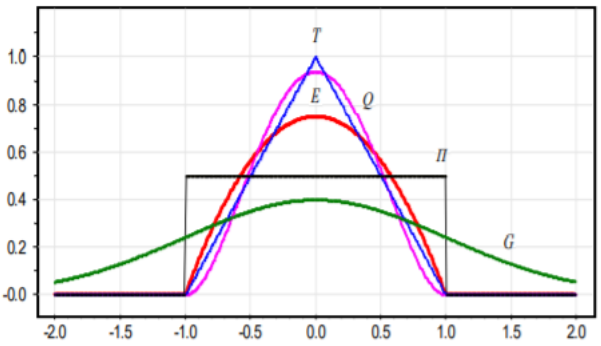

Рис. Часто используемые ядра:
E — Епанечникова;
Q — квартическое;
T — треугольное;
G — гауссовское;
$П$ — прямоугольное.

Рисунок 3.3.1 – Часто используемые ядра классификации

Входные данные:

К заданию на лабораторную работу прилагаются файлы, в которых представлены наборы данных из $\sim 10^4$ объектов. Каждый объект описывается двумя признаками ($f_j(x) \in R$) и соответствующим ему классом ($y \in \{0,1\}$).

Задание на лабораторную работу:

Суть лабораторной работы заключается в написании классификатора на основе метода k ближайших соседей. Данные из файла необходимо разбить на обучающую и тестовую выборки. На основе этих данных необходимо обучить разработанный классификатор и протестировать его на обеих выборках. В качестве отчёта требуется представить работающую программу и таблицу с результатами тестирования для каждого из 10 разбиений. Разбиение выборки необходимо выполнять программно, случайным образом. Разбивать рекомендуется по следующему правилу: делим выборку на 3 равных части, 2 части используем в качестве обучающей, одну в качестве тестовой. Кроме того, обучающая выборка должна быть сгенерирована таким образом, чтобы минимизировать разницу между количеством представленных объектов разных классов, т.е. $abs\left(\left|\{(x_i, y_i) \in X^l \middle| y_i = -1\}\right| - |\{(x_i, y_i) \in X^l | y_i = 1\}|\right) \to min.$

Варианты:

Выполнение лабораторной работы разбито на несколько пунктов, в каждом из которых есть несколько вариантов, выбор варианта опирается на N_c – индивидуальный номер студента (для студентов очной формы обучения это номер в журнале).

Первый пункт отвечает за выбор типа классификатора. Вариант выбирается по формуле $N_\text{В} = (N_c \ mod\ 3) + 1$:

1. Метод k взвешенных ближайших соседей
2. Метод парзеновского окна с фиксированным h
3. Метод парзеновского окна с относительным размером окна

Для первого варианта задания необходимо использовать весовую функцию w_i по формуле $N_w = (N_c mod\ 2) + 1$. Параметр q подбирается методом скользящего контроля.

1. $w_i = q^i, q \in (0,1)$
2. $w_i = \left(\frac{k+1-i}{k}\right)^q, q \in \{2,3,4\}$

В случае 2-го и 3-го вариантов, необходимо использовать функцию ядра $K(z)$ из списка по следующей формуле $N_\text{я} = \left((N_\text{с} * 6 + 13)\ mod\ 8\ mod\ 3\right) + 1$:

1. *Q –квартическое* $K(x) = (1 - r^2)^2[r \le 1]$
2. *T – треугольное* $K(x) = (1 - r)[r \le 1]$
3. *П – прямоугольное* $K(x) = [r \le 1]$

Кроме того, к лабораторной работе прилагаются 5 файлов с данными для классификации, файл выбирается по следующей формуле

$$N_\text{ф} = \left((N_c + 2) mod\ 5\right) + 1$$

Контрольные вопросы

1. Что понимается под классифицированием объекта?
2. Что называют тестовой выборкой?
3.Что такое валидационная выборка?
4. Как оценить точность метода ближайщих соседей?
5. Чем классификация отличается от кластеризации?

4. Логические классификаторы

Пусть $\phi: X \to \{0,1\}$ — некоторый предикат, определённый на множестве объектов X. Говорят, что предикат ϕ выделяет или покрывает (cover) объект x, если $\phi(x) = 1$. Предикат называют закономерностью, если он выделяет достаточно много объектов какого-то одного класса c, и практически не выделяет объекты других классов (более строгое определение будет дано ниже). Особую ценность представляют закономерности, которые описываются простой логической формулой. Их называют правилами (rules). Процесс поиска правил по выборке называют извлечением знаний из данных (knowledge discovery). К знаниям предъявляется особое требование — они должны быть интерпретируемы, то есть понятны людям. На практике логические закономерности часто ищут в виде конъюнкций небольшого числа элементарных высказываний. Именно в такой форме люди привыкли выражать свой житейский и профессиональный опыт.

Напомним основные обозначения. Имеется пространство объектов X и конечное множество имён классов $Y = \{1, \dots, M\}$. Целевая зависимость $y^*: X \to Y$ известна только на объектах обучающей выборки $X^\ell = (x_i, y_i)_{i=1}^\ell, y_i = y^*(x_i)$. Требуется построить алгоритм классификации $a: X \to Y$, аппроксимирующий y^* на всём X. [1]

Понятие информативности

Интуитивно предикат ϕ тем более информативен, чем больше он выделяет объектов «своего» класса $c \in Y$ по сравнению с объектами всех остальных «чужих» классов. Свои объекты называют также позитивными (positive), а чужие — негативными (negative). Введём следующие обозначения:

- P_c — число объектов класса c в выборке X^ℓ;
- $p_c(\phi)$ — из них число объектов, для которых выполняется условие $\phi(x) = 1$;
- N_c — число объектов всех остальных классов Y в выборке X^ℓ;
- $n_c(\phi)$ — из них число объектов, для которых выполняется условие $\phi(x) = 1$.

Для краткости индекс c и аргумент (ϕ) будем иногда опускать.
Предполагается, что $P \geq 1, N \geq 1$ и $P + N = \ell$.

Итак, задача построения информативного предиката ϕ сводится к оптимизации по двум критериям: $p_c(\phi) \to max$ и $n_c(\phi) \to min$. Наименее интересны те предикаты, которые либо выделяют слишком мало объектов, либо выделяют позитивные и негативные объекты примерно в той же пропорции, в которой они были представлены во всей выборке, $n : p \approx N : P$. Введём обозначение E_c для доли негативных среди всех выделяемых объектов, и D_c для доли выделяемых позитивных объектов:

$$E_c(\phi, X^\ell) = \frac{n_c(\phi)}{p_c(\phi) + n_c(\phi)}, D_c(\phi, X^\ell) = \frac{p_c(\phi)}{\ell}. \quad (4.1)$$

Опр. Предикат $\phi(x)$ будем называть логической ε,δ-закономерностью для класса $c \in Y$, если $E_c(\phi, X^\ell) \leq \varepsilon$ и $D_c(\phi, X^\ell) \geq \delta$ при заданных достаточно малом ε и достаточно большом δ из отрезка [0, 1].

Если $n_c(\phi) = 0$, то закономерность ϕ называется чистой или непротиворечивой. Если $n_c(\phi) > 0$, то закономерность ϕ называется частичной.

Статистическое определение информативности

Адекватную скалярную характеристику информативности даёт техника проверки статистических гипотез. Пусть X — вероятностное пространство, выборка X^ℓ — простая, то есть случайная, независимая, одинаково распределённая (independent, identically distributed — i.i.d.), $y^*(x)$ и $\phi(x)$ — случайные величины. Допустим, справедлива гипотеза о независимости событий $\{x: y^*(x) = c\}$ и $\{x: \phi(x) = 1\}$. Тогда вероятность реализации пары (p, n) подчиняется гипергеометрическому распределению:

$$h_{P,N}(p,n) = \frac{C_P^p C_N^n}{C_{P+N}^{p+n}}, 0 \leq p \leq P, 0 \leq n \leq N, \quad (4.2)$$

где $C_m^k = \frac{m!k!}{(m-k)!}$ — биномиальные коэффициенты, $0 \leq k \leq m$. Если вероятность $h_{P,N}(p,n)$ мала, и тем не менее пара (p,n) реализовалась, то гипотеза о независимости должна быть отвергнута. Чем меньше значение вероятности, тем более значимой является связь между y^* и ϕ. Можно сказать и так: если реализовалось маловероятное событие, то, скорее всего, оно не случайно, а закономерно.

Опр. Информативность предиката $\phi(x)$ относительно класса $c \in Y$ по выборке X^ℓ есть

$$I_c(\phi, X^\ell) = -\ln h_{P_c,N_c}\left(p_c(\phi), n_c(\phi)\right). \quad (4.3)$$

Предикат $\phi(x)$ будем называть статистической закономерностью для класса c, если $I_c(\phi, X^\ell) \geq I_0$ при заданном достаточно большом I_0. Порог информативности I_0 выбирается так, чтобы ему соответствовало достаточно малое значение вероятности, называемое уровнем значимости. Для каждой задачи он должен выбираться индивидуально.

Энтропийное определение информативности

Ещё один способ определения информативности вытекает из теории информации. Напомним, что если имеются два исхода ω_0, ω_1 с вероятностями q_0 и $q_1 = 1 - q_0$, то количество информации, связанное с исходом ω_i, по определению равно $-\log_2 q_i$. Это математическое ожидание числа бит, необходимых для записи информации о реализации исходов ω_i при

использовании оптимального (наиболее экономного) кодирования. Энтропия определяется как матожидание количества информации:

$$H(q_0, q_1) = -q_0 \log_2 q_0 - q_1 \log_2 q_1. \quad (4.4)$$

Будем считать появление объекта класса с исходом ω_0, а появление объекта любого другого класса исходом ω_1. Тогда, подставляя вместо вероятностей частоты, можно оценить энтропию выборки X^ℓ:

$$\widehat{H}(P, N) = H(\frac{P}{P+N}, \frac{N}{P+N}). \quad (4.5)$$

Допустим, стало известно, что предикат ϕ выделил p объектов из P, принадлежащих классу c, и n объектов из N, не принадлежащих классу c. Тогда энтропия выборки $\{x \in X^\ell \mid \phi(x) = 1\}$ есть $\widehat{H}(p, n)$. Вероятность появления объекта из этой выборки оценивается как $\frac{p+n}{P+N}$. Аналогично, энтропия выборки $\{x \in X^\ell \mid \phi(x) = 0\}$ есть $\widehat{H}(P-p, N-n)$, а вероятность появления объекта из неё оценивается как $\frac{P-p+N-n}{P+N}$. Таким образом, энтропия всей выборки после получения информации ϕ становится равна

$$\widehat{H}_\phi(P, N, p, n) = \frac{p+n}{P+N}\widehat{H}(p, n) + \frac{P+N-p-n}{P+N}\widehat{H}(P-p, N-n). \quad (4.6)$$

В итоге уменьшение энтропии составляет

$$IGain_c(\phi, X^\ell) = \widehat{H}(P, N) - \widehat{H}_\phi(P, N, p, n). \quad (4.7)$$

Это и есть информационный выигрыш (information gain) — количество информации об исходном делении выборки на два класса «с» и «не с», которое содержится в предикате ϕ. Таким образом, появляется ещё одно, альтернативное, определение закономерности.

Опр. Предикат ϕ является закономерностью по энтропийному критерию информативности, если $IGain_c(\phi, X^\ell) > G_0$ при некотором достаточно большом G_0.

Решающие деревья

Решающее дерево (decision tree, DT) — это логический алгоритм обучения, основанный на поиске конъюнктивных закономерностей, который используется как для задач классификации, так и для задач регрессии. Он имеет иерархическую древовидную структуру, которая состоит из корневого узла, ветвей, внутренних узлов и конечных узлов. это ещё один логический алгоритм классификации, основанный на поиске конъюнктивных закономерностей. Но, в отличие от решающего списка, при синтезе дерева все конъюнкции строятся одновременно. дерево решений начинается с корневого узла, который не имеет входящих ветвей. Исходящие ветви от корневого узла затем направляются во

внутренние узлы, также известные как узлы принятия решений. На основе доступных функций оба типа узлов выполняют оценку для формирования однородных подмножеств, которые обозначаются конечными узлами или конечными узлами. Листовые узлы представляют все возможные результаты в наборе данных. Дерево называется бинарным, если из любой его внутренней вершины выходит ровно два ребра. Выходящие рёбра связывают каждую внутреннюю вершину v с левой дочерней вершиной L_v и с правой дочерней вершиной R_v.

Опр. Бинарное решающее дерево — это алгоритм классификации, задающийся бинарным деревом, в котором каждой внутренней вершине $v \in V$ приписан предикат $\beta_v : X \to \{0,1\}$, каждой терминальной вершине $v \in V$ приписано имя класса $c_v \in Y$. При классификации объекта $x \in X$ он проходит по дереву путь от корня до некоторого листа, в соответствии с Алгоритмом

Алгоритм Классификация объекта $x \in X$ бинарным решающим деревом

1: $v := v_0$;
2: **пока** вершина v внутренняя
3: **если** $\beta_v(x) = 1$ **то**
4: $v := R_v$; (переход вправо)
5: **иначе**
6: $v := L_v$; (переход влево)
7: **вернуть** c_v.

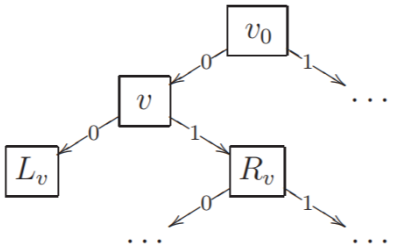

Рисунок 4.1 –Алгоритм классификации объекта $x \in X$ бинарным решающим деревом

Объект x доходит до вершины v тогда и только тогда, когда выполняется конъюнкция $K_v(x)$, составленная из всех предикатов, приписанных внутренним вершинам дерева на пути от корня v_0 до вершины v. Пусть T — множество всех терминальных вершин дерева. Множества объектов $\Omega_v = \{x \in X : K_v(x) = 1\}$, выделяемых терминальными конъюнкциями $v \in T$, попарно не пересекаются, а их объединение совпадает со всем пространством X (это легко доказывается индукцией по числу вершин дерева). Отсюда следует, что решающее дерево никогда не отказывается от классификации, в отличие от решающего списка. Отсюда также следует, что алгоритм классификации $a: X \to Y$, реализуемый бинарным решающим деревом, можно представить в виде простого голосования конъюнкций:

$$a(x) = \underset{y \in Y}{\operatorname{argmax}} \sum_{v \in T} [c_v = y] K_v(x), \tag{4.8}$$

причём для любого $x \in X$ одно и только одно слагаемое во всех этих суммах равно единице. Вместо суммирования можно было бы использовать и дизъюнкцию. Естественное требование максимизации информативности конъюнкций $K_v(x)$ означает, что каждая из них должна выделять как можно

больше обучающих объектов, допуская при этом как можно меньше ошибок. Для повышения обобщающей способности решающего дерева число листьев должно быть как можно меньше, и они должны покрывать подвыборки примерно одинаковой мощности $|\Omega_v \cap X^\ell|$.

Задание на лабораторную работу

Данная работа наиболее творческая и призвана показать, насколько студент подготовлен к реальному применению полученных знаний на практике. Как известно, в реальной работе никаких вводных данных не предоставляется, тем не менее, мы слегка пренебрегли данным правилом и предоставили теорию и предпочтительный метод для применения.

На сайте kaggle по ссылке располагаются реальные данные “Heart Attack Prediction”, которые несут информацию по сердечной заболеваемости, собранные различными медицинскими учреждениями. Каждый человек представлен 14ю характеристиками и полем goal, которое показывает наличие болезни сердца, поле принимает значение от 0 до 4 (0 – нет болезни).

Требуется имеющиеся данные разбить на обучающую и тестовую выборки в процентном соотношении 70 к 30. После чего по обучающей выборке необходимо построить решающее дерево. Для построения дерева можно пользоваться любыми существующими средствами. Кроме того, для построения дерева необходимо будет решить задачу выделения информативных решающих правил относительно имеющихся числовых признаков.

Разрешается использовать уже реализованные решающие деревья из известных библиотек (например, scikit-learn для Python), либо реализовывать алгоритм построения дерева самостоятельно (все необходимые алгоритмы представлены в теории по ссылке).

В качестве результата работы необходимо сделать 10 случайных разбиений исходных данных на обучающую и тестовую выборки, для каждой построить дерево и протестировать, после чего построить таблицу, в которой указать процент правильно классифицированных данных. Полученную таблицу необходимо включить в отчёт по лабораторной работе.

Контрольные вопросы

1. Какой метод нужно использовать для построения дерева решений?

2. Возможно ли расширить модель дерева решений?

3. Какой параметр метода RandomForestClassifier отвечает за максимальную глубину дерева?

4. В какой библиотеке находится метод для построения дерева решений?

5. Восстановление регрессии

Регрессия—это метод, используемый для моделирования и анализа отношений зависимой и независимыми переменными, а также для того, чтобы осуществить анализ того, как эти переменные вместе влияют на получение определенного результата. Проще говоря, метод осуществляет подгонку функции из выбранного семейства функций к выборочным данным при некоторой функции ошибки. Регрессионный анализ — один из самых основных инструментов в области машинного обучения, используемых для прогнозирования. Используя регрессию, возможно подобрать функцию к доступным данным и попытаться предсказать результат на будущее или задержку данных.

Задачу обучения по прецедентам при Y = R принято называть задачей восстановления регрессии. Основные обозначения остаются прежними. Задано пространство объектов X и множество возможных ответов Y . Существует неизвестная целевая зависимость y ∗ : X → Y , значения которой известны только на объектах обучающей выборки Xℓ = (xi , yi) ℓ i=1, yi = y ∗ (xi). Требуется построить алгоритм a: X → Y , аппроксимирующий целевую зависимость y ∗ .

5.1. Линейная регрессия

Линейная регрессия — это алгоритм, принадлежащий контролируемому машинному обучению. Он пытается применить отношения, которые будут предсказывать исход события на основе точек данных независимой переменной. Отношение обычно представляет собой прямую линию, которая наилучшим образом соответствует различным точкам данных как можно ближе. Выходные данные имеют непрерывную форму, т. е. числовое значение. Например, результатом может быть доход или продажи в валюте, количество проданных продуктов и т. д.

Целевая функция линейной регрессионной модели ($y = f(x, b) + \varepsilon, E(\varepsilon) = 0$, где b — параметры модели, ε — случайная ошибка модели) $f(x, b)$ имеет вид $f(x, b) = b_0 + b_1 x_1 + b_2 x_2 + \cdots + + b_k x_k$, где — b_j параметры (коэффициенты) регрессии, x_j — регрессоры (факторы модели), k — количество факторов модели.

5.2. ElasticNet

Эластичная сеть — модель регрессии с двумя регуляризаторами l1, l2.

Пусть известны измерения n объектов. Каждый объект представим в виде пары (x_i, y_i), $x_i \in \mathrm{R}^k$, $y_i \in \mathrm{R}$. Для удобства будем записывать это в матричном виде: (X, y) . Классическая задача регрессии ставится следующим образом:

$$|y - Xw|^2 \rightarrow \min_w \quad . \qquad (5.2.1)$$

В силу неточности измерений данных или каких либо еще ошибок с целью построения наилучшей модели вводят регуляризатор или несколько регуляризаторов. Тогда получается следующая задача оптимизации:

$$|y - Xw|_2^2 + \lambda_1|w|_1 + \lambda_2|w|_2^2 \rightarrow \min_w \quad . \qquad (5.2.2)$$

Частными случаями являются модели лассо ($\lambda_2 = 0$) и гребневой регрессии ($\lambda_1 = 0$). Для каждого фиксированного λ_2 можно перебрать все возможные значения параметра λ_1 (поскольку по этому параметру это кусочно-линейная функция с конечным числом точек, в которых она не дифференцируема).

5.3. LASSO

Lasso (Least absolute shrinkage and selection operator) - метод оценивания коэффициентов линейной регрессионной модели.

Метод заключается во введении ограничения на норму вектора коэффициентов модели, что приводит к обращению в 0 некоторых коэффициентов модели. Метод приводит к повышению устойчивости модели в случае большого числа обусловленности матрицы признаков *X*, позволяет получить интерпретируемые модели - отбираются признаки, оказывающие наибольшее влияние на вектор ответов**.**

Задана выборка $D = \{(x_n), y_n\}^N n = 1, x \in \mathrm{R}^m$.

Пусть $X = \{x_1, x_2, \dots, x_m\}$ - матрица значений свободных переменных (матрица признаков), y - вектор ответов.

Пусть столбцы *X* нормированы, математическое ожидание каждой из свободных переменных равно 0, так что выполнены следующие условия:

$$\sum_{i=1}^{n} x_{ij}^2 = 1,$$

$$\frac{1}{n}\sum_{i=1}^{n} x_{ij} = 0,$$

$$\frac{1}{n}\sum_{i=1}^{n} y_j = 0,$$

где j = 1, 2, …, m.

Назначена линейная модель. Регрессионные коэффициенты $\beta = (\beta_1, \beta_2, \dots, \beta_m)'$ определяют вектор μ:

$$\mu = \sum_{j=1}^{m} x_j\beta_j = X\beta \qquad (5.3.1)$$

Критерием качества модели считается среднеквадратичная ошибка:

$$S(\beta) = ||y - \mu||^2 = \sum_{i=1}^{n}(y_i - \mu_i)^2 \qquad (5.3.2)$$

Результатом минимизации среднеквадратичной ошибки по β методом наименьших квадратов является вектор $\hat{\beta}$.

Пусть $T(\hat{\beta})$ - сумма модулей регрессионных коэффициентов, $T(\hat{\beta})=\sum_{j=1}^{m}|\hat{\beta}_j|$.

В Lasso выбирается $\hat{\beta}$ из условия минимизации $S(\beta)$ при ограничении $T(\hat{\beta}) \leq t$, где $t \geq 0$ – параметр регуляризации.

Чтобы найти коэффициенты можно воспользоваться методом квадратичного программирования с линейным ограничением-неравенством.

При больших значениях *t* решение совпадает с решением метода наименьших квадратов. Чем меньше t, тем большее число коэффициентов $\hat{\beta}_i$ принимают нулевое значение. Таким образом, Lasso осуществляет отбор информативных признаков.

5.4. LARS

Регрессия по наименьшему углу (LARS) - это алгоритм для подбора модели линейной регрессии для многомерных данных, разработанные Брэдли Эфроном , Тревором Хасти , Иэном Джонстоном и Робертом Тибширани .

Предположим, ожидается, что переменная ответа будет определена линейной комбинацией подмножества потенциальных ковариат. Затем алгоритм LARS предоставляет средства для оценки того, какие переменные необходимо включить, а также их коэффициенты.

Вместо того, чтобы давать векторный результат, решение LARS состоит из кривой, обозначающей решение для каждого значения L1 norm вектора параметров. Алгоритм аналогичен прямому пошаговой регрессии, но вместо включения переменных на каждом шаге оцениваемые параметры увеличиваются в направлении, равносильном корреляциям каждого из них с остатком.

Алгоритм

Основные шаги Алгоритма регрессии по наименьшему углу:

1. Начать со всех коэффициентов β, равных нулю.

2. Найти предиктор x_j, который наиболее коррелирует с y.

3. Увеличить коэффициент β_j в направлении знака его корреляции с y. По пути взять остатки $r = y - \hat{y}$. Остановиться, когда какой-либо другой предиктор x_k имеет такую же корреляцию с r, как и с x_j .

4. Увеличить (β_j, β_k) в их совместном направлении наименьших квадратов, пока какой-либо другой предиктор x_m не будет иметь такой же корреляции с остатком r.

5.Увеличить (β_j, β_k, β_m) в их объединенном направлении наименьших квадратов, пока какой-либо другой предиктор x_n не будет иметь такой же корреляции с остатком r.

6. Продолжайте до тех пор, пока все предикторы не будут в модели.

5.5. Пример реализации регрессии в Scikit-Learn

На практике предлагается использовать проверенную и широко используемую библиотеку *Scikit-Learn* для реализации регрессии.

Следующая команда импортирует набор данных CSV, используя библиотеку *pandas*:

```
dataset = pd.read_csv('Weather.csv')
```

Чтобы увидеть статистические данные набора данных, можно использовать метод *describe*():

```
dataset.describe()
```

Затем разделяем данные: 80% данных это обучающий набор, а 20% данных – это набор тестовых данных. Переменная *test_size* задаёт пропорцию тестового набора:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

После разделения данных на обучающие и тестовые наборы следует обучить алгоритм. Для этого нужно импортировать класс *LinearRegression*, создать его экземпляр и вызвать *fit()* метод вместе с данными для обучения.

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

После обучения алгоритма пришло время сделать прогнозы. Для этого можно использовать тестовые данные чтобы посмотреть, насколько точно алгоритм предсказывает процентную оценку. Для прогноза на тестовых данных выполните следующий скрипт:

```
y_pred = regressor.predict(X_test)
```

Задание на лабораторную работу

Целью данной лабораторной работы является разработка программы, реализующей применение метода линейной регрессии к заданному набору данных.

Набор данных содержит в себе информацию о вариантах португальского вина "Винью Верде". Входные переменные представляют собой 13 столбцов со значениями, полученными на основе физико-химических тестов, а именно:

0 – цвет вина (“red” / ”white”)

1 - фиксированная кислотность

2 - летучая кислотность

3 - лимонная кислота

4 - остаточный сахар

5 - хлориды

6 - свободный диоксид серы

7 - общий диоксид серы

8 - плотность

9 - pH

10 - сульфаты

11 - спирт

Выходная переменная (на основе сенсорных данных):

12 - качество (оценка от 0 до 10, целое число)

Классы упорядочены и не сбалансированы (например, нормальных вин гораздо больше, чем отличных или плохих). В предоставленных данных есть пропуски и неточности. Задания выполняются согласно варианту. Чтобы определить номер варианта, воспользуйтесь следующей формулой:

Nварианта = (Nпо списку mod 4) + 1

Варианты заданий:

1) Использовать классическую модель LinearRegression

2) Использовать модель LASSO

3) Использовать модель LARS

4) Использовать модель ElasticNet

Задание: Данные необходимо рассматривать как три набора. Данные для красного вина, данные для белого, общие данные вне зависимости от цвета. Необходимо построить модель для каждого из наборов, обучить её и сравнить полученные при помощи модели результаты с известными. Для обучения использовать 70% выборки, для тестирования 30%. Разбивать необходимо случайным образом, а, следовательно, для корректности тестирования качества модели, эксперимент необходимо провести не менее 10 раз и вычислить среднее значение качества регрессии.

Особенности работы с данными:

1) Данные разнотипные, поэтому необходимо все столбцы привести к одному типу. Все данные должны быть вещественными числами. В данных есть пропуски, а это означает, что при считывании они будут записаны как NaN (либо произойдёт ошибка).

2) Результат работы модели будет тоже вещественным числом. Поэтому для оценки качества работы модели, необходимо использовать не прямое сравнение, а учитывать разницу между настоящим значением и смоделированным.

3) Данные в столбцах имеют разную размерность. Поэтому необходимо их нормализовать. Можно воспользоваться, например, методом preprocessing.normalize().

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \in [0,1]$$

В качестве результата выполненной лабораторной работы должна быть разработанная программа, решающая поставленную задачу и отчёт с содержанием текста программы, краткими комментариями и результатами работы программы.

Контрольные вопросы

1. В какой библиотеке находится метод для построения линейной регрессии?
2. Регрессионный анализ – это?
3. Является ли определение дохода клиента банка задачей регрессии?
4. Приведите практический пример задачи регрессионного анализа.

6. Нейронные сети

Нейронная сеть — это последовательность нейронов, соединенных между собой синапсами. Нейронные сети представляют собой набор алгоритмов, примерно смоделированных по образцу человеческого мозга, которые предназначены для распознавания закономерностей. Они интерпретируют сенсорные данные посредством своего рода машинного восприятия, маркируя или группируя исходные данные. Образцы, которые они распознают, являются числовыми, содержащимися в векторах, в которые должны быть переведены все данные реального мира, будь то изображения, звук, текст или временные ряды.

Среди распространённых задач, где применяются нейронные сети выделяются:

- Классификация — возможность предсказания категории объекта и разделение объектов согласно заданным заранее признакам.
- Прогнозирование — возможность предсказания будущих значений данных.
- Распознавание – возможность обнаружения экземпляров визуальных объектов определенных классов в цифровых изображениях.

Нейрон — это вычислительная единица, называемая нейронным узлом, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Типичная нейронная сеть состоит из слоев нейронов, называемых нейронными узлами. Эти слои бывают следующих трех типов: входной слой (одиночный), скрытый слой (один или несколько) и выходной слой (одиночный). Входной слой получает информацию, n скрытых слоев, которые обрабатывают эту информацию и выходной слой, который выводит результат. У каждого из нейронов есть 2 основных параметра: входные и выходные данные. В случае входного нейрона: входные данные являются выходными. В остальных, во входные данные попадает суммарная информация всех нейронов с предыдущего слоя, после чего она нормализуется с помощью функции активации и попадает в поле выхода. Математическая функция, содержащаяся в нейроне, может варьироваться в зависимости от типа искусственной нейронной сети — это могут быть простые функции регрессии, нелинейные сигмовидные функции и так далее.

Синапс – это связь между двумя нейронами. У синапсов есть 1 параметр — **вес**. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. Допустим, есть 3 нейрона, которые передают информацию следующему. Тогда есть 3 веса, соответствующие каждому из этих нейронов. У того нейрона, у которого вес будет больше, та информация и будет доминирующей в следующем нейроне (пример — смешение цветов). Совокупность весов нейронной сети или матрица весов — это своеобразный мозг всей системы. Именно благодаря этим весам, входная информация обрабатывается и превращается в результат.

Функция активации — это способ нормализации входных данных. То есть, если на входе будет большое число, пропустив его через функцию активации, можно получить на выходе число в нужном диапазоне. Функций активации достаточно много, но наиболее распространённые: Линейная, Сигмоид (Логистическая) и Гиперболический тангенс. Главные их отличия — это диапазон значений.

Ошибка — это процентная величина, отражающая расхождение между ожидаемым и полученным ответами. Ошибка формируется каждую эпоху и должна идти на спад. Если этого не происходит, значит, вы что-то делаете не так. Ошибку можно вычислить разными путями, самые распространённые: Mean Squared Error (далее MSE), Root MSE и Arctan. Здесь нет какого-либо ограничения на использование, как в функции активации, и можно выбрать любой метод, который будет приносить наилучший результат. Стоит лишь учитывать, что каждый метод считает ошибки по-разному.

Для разных данных и приложений используются различные типы нейронных сетей. Различные архитектуры нейронных сетей специально разработаны для работы с этими конкретными типами данных или предметной областью.
Среди типов нейронных сетей выделяют:

- Персептрон
  Самая простая форма нейронных сетей. Нейрон выполняет обработку добавления входных значений с их весами. Полученная сумма затем передается в функцию активации для создания двоичного вывода.
- Сети с прямой связью
  Сети с прямой связью состоят из множества нейронов и скрытых слоев. Чем больше количество слоев, тем больше может быть настройка весов. И, следовательно, больше будет способности сети к обучению. Веса не обновляются, так как нет обратного распространения. Результат умножения весов на входы подается на функцию активации, которая действует как пороговое значение.
- Многослойный персептрон
  В отличии от сетей с прямой связью многослойные персептроны включают в себя несколько скрытых слоев и функций активации. Обучение происходит под наблюдением, когда веса обновляются с помощью градиентного спуска.
- Сверточные нейронные сети
  CNN содержит несколько слоев свертки, которые отвечают за извлечение важных функций из изображения. Операция свертки использует пользовательскую матрицу, также называемую фильтрами, для свертки входного изображения и создания карт. Эти фильтры инициализируются случайным образом, а затем обновляются путем обратного распространения. После слоя свертки идет слой объединения, который отвечает за агрегацию карт, созданных на основе свертки.

- Сети долгой краткосрочной памяти
  Нейронные сети с долгой краткосрочной памятью преодолевают проблему исчезающего градиента, добавляя специальную ячейку памяти, которая может хранить информацию в течение длительных периодов времени. Такие сети использует фильтры, чтобы определить, какой вывод следует использовать или забыть.

  Далее показан процесс разработки и запуска нейронной сети.

```
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras import utils
from tensorflow.keras.preprocessing import image
from google.colab import files
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

Рисунок 6.1 – Подключение необходимых пакетов.

```
classes = ['футболка', 'брюки', 'свитер', 'платье', 'пальто', 'туфли', 'рубашка', 'кроссовки', 'сумка', 'ботинки']
(x_train,y_train),(x_test,y_test)=fashion_mnist.load_data()
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train=x_train/255
x_test=x_test/255
y_train = utils.to_categorical(y_train, 10)
y_test = utils.to_categorical(y_test, 10)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

Рисунок 6.2 – Загрузка данных, объявление списка с названиями классов, нормализация данных, преобразование классов в метки категорий.

```
from tensorflow.keras import activations

model=Sequential()
model.add(Dense(1600,input_dim=784,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1600)              1256000
_________________________________________________________________
dense_1 (Dense)              (None, 10)                16010
=================================================================
Total params: 1,272,010
Trainable params: 1,272,010
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 6.3 – Создание последовательной модели, добавление уровней сети, компиляция модели.

```
model.fit(x=x_train,y=y_train,batch_size=50,epochs=15,validation_split=0.2)

Epoch 1/15
960/960 [==============================] - 2s 3ms/step - loss: 0.3405 - accuracy: 0.8754 - val_loss: 0.3500 - val_accuracy: 0.8709
Epoch 2/15
960/960 [==============================] - 2s 3ms/step - loss: 0.3124 - accuracy: 0.8851 - val_loss: 0.3528 - val_accuracy: 0.8715
Epoch 3/15
960/960 [==============================] - 2s 3ms/step - loss: 0.2912 - accuracy: 0.8924 - val_loss: 0.3476 - val_accuracy: 0.8752
Epoch 4/15
960/960 [==============================] - 2s 3ms/step - loss: 0.2772 - accuracy: 0.8970 - val_loss: 0.3217 - val_accuracy: 0.8832
Epoch 5/15
960/960 [==============================] - 2s 3ms/step - loss: 0.2596 - accuracy: 0.9023 - val_loss: 0.3171 - val_accuracy: 0.8884
Epoch 6/15
960/960 [==============================] - 2s 3ms/step - loss: 0.2445 - accuracy: 0.9092 - val_loss: 0.3173 - val_accuracy: 0.8871
Epoch 7/15
960/960 [==============================] - 2s 3ms/step - loss: 0.2354 - accuracy: 0.9116 - val_loss: 0.3731 - val_accuracy: 0.8785
```

Рисунок 6.4 – Обучение сети.

```
L=len(y_test)
correct=0
YP=model.predict(x_test)
for i in range(L):
  y1=np.argmax(y_test[i])
  ypred=np.argmax(YP[i])
  if ypred==y1:
    correct+=1
print(correct,' ',L)
print(correct/L*100)

8941   10000
89.41
```

Рисунок 6.5 – Оценка доли верных ответов на тестовых данных

```
prediction = model.predict(x)
```

Рисунок 6.6 – Запуск распознавания.

```
[ ] prediction

array([[0.0000000e+00, 8.6703852e-31, 4.3886688e-01, 3.0877629e-15,
        0.0000000e+00, 2.0248261e-23, 1.6152668e-25, 0.0000000e+00,
        5.6112343e-01, 9.6952454e-06]], dtype=float32)

prediction = np.argmax(prediction)
print("Номер класса:", prediction)
print("Название класса:", classes[prediction])
```

Рисунок 6.7 – Результаты распознавания.

Задание на лабораторную работу

Целью данной лабораторной работы является разработка нейронной сети для решения задачи классификации или регрессии в зависимости от набора данных в рамках варианта. Лабораторная работа предполагает разработку на языке программирования Python с использованием библиотеки Keras.

Варианты заданий:

1) Распознавание цифр на изображении (MNIST digits classification dataset)
2) Распознавание 10 предметов на уменьшенных изображениях
(CIFAR10 small images classification dataset)
3) Определение эмоционального окраса рецензии фильма (IMDB movie review sentiment classification dataset)
4) Классификация ленты новостей (Reuters newswire classification dataset)
5) Определение цены недвижимости
(Boston Housing price regression dataset)

Все наборы данных доступны по ссылке : https://keras.io/api/datasets/

При разработке нейронной сети следует соблюсти наличие необходимых составляющих исходя из следующих вариантов:

1) Нейросеть должна состоять из трёх полносвязных слоёв, обязательное использование Dropout, в качестве оптимизатора использовать Adam;
2) Нейросеть должна состоять из четырех полносвязных слоёв, обязательное использование GaussianDropout, в качестве оптимизатора использовать SGD;
3) Нейросеть должна состоять из пяти полносвязных слоёв, обязательное использование ActivityRegularization, в качестве оптимизатора использовать RMSprop.

Выбор количества нейронов на всех внутренних слоях, функций активации и других параметров должен быть обусловлен оптимальностью работы модели.

Формула вычисления варианта:

$$N_{в1} = N_{сп} \, mod \, 5 + 1$$

$$N_{в2} = (N_{сп} + 2) \, mod \, 3 + 1$$

где $N_{сп}$ - номер студента в списке.

Для защиты лабораторной работы следует обосновать выбор значений дополнительных параметров и продемонстрировать работу обученной нейронной сети. Обоснование должно включать в себя демонстрацию качества работы сети на валидационном наборе данных в процессе обучения. Параметры выбираются те, на которых валидация даёт наилучший результат.

Контрольные вопросы

1. Какая библиотека Python позволит составить графики любого типа?
2. Какой метод позволяет изменить размерность входных данных?
3. Чем отличается свёрточная модель от полносвязной модели нейронной сети?
4. Каких видов бывают нейронные сети?
    а) Полносвязные и рекуррентные
    б) Рекуррентные, сверточные и полносвязные
    в) Рекуррентные, сверточные, полносвязные и трансформеры

Заключение

В результате освоения данного курса обучающийся получает как теоретические, так и практические навыки применения методов машинного обучения, в том числе и на реальных данных. В ходе выполнения лабораторных работ, студент приобретает знания о средствах разработки, наиболее подходящих для машинного обучения, об уже существующих решениях и возможностях их использования, учится самостоятельно искать недостающую информацию и анализировать полученные данные. В дальнейшем полученные навыки пригодятся в профессиональной деятельности, так как в реальных задачах специалистам приходится постоянно сталкиваться с неполными и неточными данными, уметь их анализировать и грамотно обрабатывать. Применение подобных методов позволит стать специалистом широкого профиля и работать в любой сфере деятельности человека, от анализа банковских операций до прогнозирования урожайности закупленного фермой зерна.

Список литературы

1. Воронцов К. В. Лекции по логическим алгоритмам классификации //Режим доступа: http://www.machinelearning.ru/wiki/images/3/3e/Voron-ML-Logic.pdf– 2007.
2. Воронцов К. В. Математические методы обучения по прецедентам (теория обучения машин) //Москва. – 2011. – С. 119-121.
3. Золотых Н. Ю. Машинное обучение и анализ данных. – 2013.
4. Дмитриев Е. А. Метрические классификаторы. – 2017.
5. Рашид Т. Создаем нейронную сеть. – Litres, 2022.
6. Галанов А. Э., Селюкова Г. П. Нейронные сети и нейронные технологии //Актуальные вопросы науки и хозяйства: новые вызовы и решения. – 2019. – С. 399-405.

Учебное издание

Дементьева Кристина Игоревна
Ракитский Антон Андреевич

Методы машинного обучения

Редактор
Корректор

Подписано в печать
Формат бумаги 62 × 84/16, отпечатано на ризографе, шрифт № 10,
п. л. – 2,2, заказ № , тираж – 100.

Отдел рекламы и PR СибГУТИ
630102, г. Новосибирск, ул. Кирова, 86, офис 107, тел. (383) 269-83-18