МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению лабораторных работ
по дисциплине
«МАШИННОЕ ОБУЧЕНИЕ»
для студентов направления подготовки
02.03.01 Математика и компьютерные науки
направленность (профиль)
«Анализ данных и искусственный интеллект»

Введение

Целью изучения дисциплины «Машинное обучение» является формирование профессиональной компетенции (ПК-1) у будущего бакалавра по направлению подготовки 02.03.01 Математика и компьютерные науки.

Задачи дисциплины:

- формирование у студентов теоретических знаний и практических навыков по основам машинного обучения;
- овладение студентами инструментарием, моделями и методами машинного обучения;
- приобретение навыков исследователя данных (datascientist) и разработчика математических моделей, методов и алгоритмов анализа данных.

Перечень планируемых результатов обучения по дисциплине, соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	**Код, формулировка индикатора**	**Планируемые результаты обучения по дисциплине, характеризующие этапы формирования компетенций, индикаторов**
ПК-1. Способен демонстрировать базовые знания математических и естественных наук, основ программирования и информационных технологий	**ИД-2$_{ПК-1}$** применяет информационные технологии и среды программирования для решения профессиональных задач	Способен применять инструменты, модели и методы машинного обучения для решения профессиональных задач

Данные методические указания к лабораторным работам разработаны в соответствии с рабочей программой по дисциплине «Машинное обучение» для студентов направления подготовки 02.03.01 «Математика и компьютерные науки».

Лабораторная работа 1

Тема: «Введение в машинное обучение».

Цель работы
Проверить знания базовых инструментов, которые используются для обращения с данными в проектах по машинному обучению - библиотек numpy, pandas, средств визуализации matplotlib, seaborn.

Методические указания
Для успешного выполнения заданий и проектов по машинному обучению необходимо постоянное манипулирование данными в определенных форматах. Очень облегчают эту задачу специализированные библиотеки. В экосистеме Python почти всегда используются модули numpy и pandas для анализа и преобразования данных. Поэтому для эффективной работы с моделями машинного обучения необходимо виртуозно владеть этими базовыми библиотеками.

Среди базовых возможностей языка программирования Python при работе с анализом данных и машинным обучением чаще всего используются следующие:

- индексирование массивов и срезы
- генераторные выражения с условиями и циклами
- анонимные функции

Среди средств библиотеки numpy чаще всего применяются следующие инструменты:

- функции генерации массивов arange, linspace, logspace
- функция генерации сетки mgrid
- функции создания матриц diag, zeroes, ones
- создание случайных значений - пакет random
- форма массива - shape
- изменение формы массива - reshape
- оператор многоточия - ...
- сложное индексирование
- индексные маски
- поэлементные операции

Библиотека pandas особенна полезна для манипуляции двумерными массивами информации. К самым часто используемым ее возможностям в машинном обучении относятся следующие:

- сохранение и чтение файлов в формате csv, xls, xlsx
- аналитические функции info, describe, head
- итерация по строкам и столбцам, обращение к строкам и столбцам по номерам и индексам/названиям
- объединения и соединения таблиц

- агрегатные функции и группировка
- замена отдельных значений в таблице в том числе по условию

Кроме того, нам часто придется визуализировать данные в виде различных графиков, гистограмм. Средства визуализации тоже очень важны и полезны. Аналитик должен владеть разными типами графиков и выбирать наиболее подходящий для решения конкретной задачи. Помните, что визуализация нужна для того, чтобы сделать видимыми какие-то скрытые зависимости в данных. То есть визуализация должна быть простой, наглядной и понятной, с явными целями и выводами.

Эту и все последующие работы следует выполнять в ноутбучном редакторе Python, который позволяет выполнять код по ячейкам. Такой формат работы больше подходит для прототипирования и построения проекта по анализу данных и машинному обучению. Рекомендуется использовать Jupyternotebook или Google Colab.

Задания для самостоятельного выполнения

1. С помощью массивов numpy создайте таблицу умножения.
2. Создайте функцию, которая принимает как аргументы целое число N и первый элемент (вещественное число), и разность (вещественное число) и создает матрицу numpy по диагонали, которой располагаются первые N членов арифметической прогрессии.
3. Сгенерируйте средствами numpy матрицу A 5 на 5, содержащую последовательные числа от 1 до 25. Используя срезы извлеките в плоский массив все нечетные элементы этой матрицы.
4. Создайте двумерный массив, содержащий единицы на границе и нули внутри.
5. Создайте две матрицы размером (5,5). Одна матрица содержит 5 в шахматном порядке как в задаче домашнего задания, другая имеет треугольную форму содержащую 5 на основной диагонали и в позициях выше ее, а ниже все 0. Посчитайте их детерминант и найдите обратные матрицы.
6. С помощью pandas загрузите датасет для предсказания цены квартиры, прилагающийся к этой работе.
7. Выведите на экран несколько первых и несколько последний строк файла.
8. Выведите с помощью методов pandas основные количественные параметры датасета: количество строк и столбцов, тип данных каждого поля, количество значений в каждом столбце, шкала измерения каждого численного поля.
9. Удалите из таблицы столбцы, содержащие идентификаторы, переименуйте все оставшиеся названия колонок на русском языке.
10. Выведите отдельно столбец, содержащий цену, по номеру и названию. Выведите первую, десятую и предпоследнюю строку таблицы по номеру и по индексу.
11. Выделите в отдельную таблицу последние десять строк. Уберите в ней столбец с ценой. Склейте ее с первоначальной таблицей при помощи append. Заполните отсутствующие значения цены средним по таблице.
12. Выделите пять последних колонок в отдельную таблицу. Удалите в ней строки, в которых цена ниже среднего. Присоедините эту таблицу к изначальной (выберите самый подходящий тип соединения).

13. Выведите таблицу, содержащую среднюю цену и количество квартир на каждом этаже из первоначального набора данных.
14. Сохраните получившуюся таблицу в файлы формата csv и xlsx. Прочитайте их и убедитесь, что данные отображаются корректно.
15. Создайте в Excel или другом табличном редакторе таблицу, содержащую несколько численных и текстовых полей. Прочитайте ее в программу при помощи pandas.
16. Дальнейшие задания производите, используя изначальную версию датасета. Должны быть подписаны названия графиков, названия осей, указаны значения на осях. Оцениваться будет использование количества различных атрибутов при построении графиков и визуальная красота.
17. Постройте круговую диаграмму для признака Rooms, иллюстрирующую количество квартир в процентах в зависимости от количества комнат. Сделайте сектор с наибольшим числом квартир выдвинутым.
18. Постройте гистограмму по целевой переменной Price. Оцените визуально, по какой цене продаётся наибольшее количество квартир.
19. Постройте диаграммы рассеяния для признаков Rooms, Square, HouseFloor, HouseYear в зависимости от целевой переменной Price в одной области figure. Оцените визуально, есть ли среди них такие, на которых разброс точек близок к линейной функции.
20. Постройте ядерную оценку плотности целевой переменной Price. Оцените визуально, напоминает ли полученный график нормальное распределение. Постройте двумерную ядерную оценку плотности для целевой переменной Price и признака HouseFloor, затем оцените визуально на каких этажах и по какой цене продаётся основная масса квартир.
21. Постройте ящиковую диаграмму признака Square. Оцените визуально имеются ли выбросы, и, если да, то начиная с какого размера площади значение признака можно считать выбросом.
22. При помощи сетки графиков PairGrid визуализируйте попарные отношения признаков Rooms, Square, HouseFloor, HouseYear, Price следующим образом: на диагонали - гистограммы, под диагональю - ядерные оценки плотности, над диагональю - диаграммы рассеяния. По результатам визуализации сделайте выводы.
23. Постройте тепловую карту матрицы корреляции (df.corr()) признаков Rooms, Square, HouseFloor, HouseYear, Price. По ней определите, какие признаки являются зависимыми (у таких признаков коэффициент корреляции близок к единице).

Контрольные вопросы

1. Какие структуры данных используются в Numpy? В чем их отличие от списков Python?
2. Какие функции для генерации массивов использует Numpy?
3. Какие способы предлагает Numpy для извлечения данных из массивов?
4. Что такое векторизация кода и почему это ускоряет работу программ?
5. Какие виды матричных операций реализованы в Numpy?
6. Какие функции используются для преобразования формы, размера и соединения массивов?
7. Какие две главные структуры данных используются в pandas? В чем их отличие?
8. Как происходит объединение двух таблиц в pandas?

9. Зачем нужны и как работают индексы в pandas.
10. Построение каких основных видов графиков используется при анализе данных в машинном обучении?
11. В чём разница между библиотеками matplotlib и seaborn? Каковы преимущества каждой из них?
12. Как задать размер графика в matplotlib?
13. Как установить стили в seaborn?
14. Для чего используют подграфики subplots?
15. Какие основные типы графиков реализованы в matplotlib? Что изображается на ящиковой диаграмме?
16. Как поменять палитру цветов у тепловой карты?
17. Дополнительные задания
18. Изучите документацию модулей numpy, pandas, matplotlib по тем темам, которые упомянуты в методических указаниях.
19. Изучите приемы работы с датой и временем: модуль strftime.
20. Попробуйте загружать как локальный файл, содержащий данные, так и файл, размещенный на хостинге (например, GitHub) по URL.
21. Продемонстрируйте разные типы соединения таблиц в pandas.
22. Напишите SQL-аналоги всех операций, которые мы проводили в лабораторной с помощью pandas.
23. В ноутбуке
24. Постройте график jointplot (гибрид scatterplot и histogram) из библиотеки seaborn. Постройте график violinplot (гибрид boxplot и ядерной оценки плотности) из библиотеки seaborn.
25. Продемонстрируйте работу с сеткой подзаголовков FacetGrid.
26. Продемонстрируйте работу с трехмерными графиками. Создайте интерактивный график.

Лабораторная работа 2

Тема: «Метрические классификаторы»

Цель работы

Познакомиться с основными приемами работы с моделями классификации в scikit-learn.

Задания для выполнения

1. Загрузите данные о диагностике сахарного диабета.
2. Постройте модель классификации для предсказания наличия заболевания.
3. Оцените качество построенной модели с помощью отчета о классификации и матрицы классификации.
4. Постройте альтернативную полиномиальную модель, сравните ее с предыдущей.

Методические указания

Для начала работы обратимся к набору данных diabetes. Это довольно известный датасет, собравший информацию о медицинских показателях более 700 пациентов, обследованных на предмет наличия сахарного диабета. На нем мы потренируемся строить классификационные модели.

Сперва загрузим исходный набор данных. Это можно сделать, как скопировав файл csv в локальную папку, так и по общедоступному URL:

data =
pd.read_csv("https://raw.githubusercontent.com/koroteevmv/ML_course/2023/ML2.2%20real%20classification/data/diabetes.csv")

Обратите внимание, что в библиотеке *sklearn* встроен очень похожий датасет pima-indian-diabetes. Имейте в виду, что в данной работе используется немного другой датасет.

Как и ранее, хорошей идеей перед началом анализа будет познакомиться с составом набора данных визуально. Выведем датасет на экран:

data.head()

Познакомьтесь с основной структурой датасета:

index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

При проведении серьезного анализа перед построением модели машинного обучения нужно провести тщательную обработку и очистку набора данных - удаление пропущенных значений, анализ шкал, нормализация, удаление выбросов и аномалий. В учебных целях ограничимся обязательными проверками критических ошибок в данных.

В первую очередь проверим данные на наличие пропущенных значений:

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Видим, что пропусков в данных нет. Кроме того, видно, что все данные выражены в численных шкалах. Значит, особенной обработки данный датасет не требует, он уже достаточно чистый. Теперь можно вывести основную статистику по датасету:
data.describe()

index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0
mean	3.8450520833333335	120.8945312 5	69.10546875	20.536458333333332	79.79947916666667	31.992578124999998	0.4718763020833332 5	33.240885416666664	0.3489583333333333
std	3.3695780626988694	31.9726181951362 2	19.355807170644777	15.952217567727637	115.24400235133817	7.884160320375446	0.3313285950127749	11.760231540678685	0.47695137724279896
min	0.0	0.0	0.0	0.0	0.0	0.0	0.078	21.0	0.0
25%	1.0	99.0	62.0	0.0	0.0	27.3	0.24375	24.0	0.0
50%	3.0	117.0	72.0	23.0	30.5	32.0	0.3725	29.0	0.0

index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
75%	6.0	140.25	80.0	32.0	127.25	36.6	0.62625	41.0	1.0
max	17.0	199.0	122.0	99.0	846.0	67.1	2.42	81.0	1.0

Здесь мы видим шкалу измерения каждого признака. Можно прийти к выводу, что явных видимых аномалий в данных нет. отдельно обратим внимание на столбец "Outcome" - он содержит целевую переменную. В данном случае она также выражается числом (0 - здоров, 1 - болен), но это не всегда так.
Теперь выделим целевую переменную и факторы:
y = data.Outcome
X = data.drop(["Outcome"], axis=1)
Выведем форму получившихся массивов:
y.shape, X.shape
((442,), (442, 10))
Данные выглядят полностью готовыми к началу машинного обучения. Для начала импортируем нужный класс и создадим его экземпляр:

```
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression()
```

Обучим созданную модель на имеющихся у нас данных:
logistic.fit(X, y)
После выполнения этой инструкции мы можем увидеть специальное предупреждение:

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (sta
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression()
```

Смысл этого сообщения в том, что процесс обучения завершился по условию достижения максимального количества итераций, а не по условию стабилизации функции ошибки. Это значит, что модель обучается трудно и медленно. Это может свидетельствовать о том, что результаты могут быть не очень удовлетворительными.
Но давайте посмотрим, что за модель мы получили после такого обучения. В первую очередь выведем коэффициенты модели:
print("Coefficients: \n", logistic.coef_[0])
Coefficients:
[1.17252338e-01 3.35998450e-02 -1.40873757e-02 -1.27047756e-03
-1.24032162e-03 7.72023360e-02 1.41904174e+00 1.00353608e-02]

В линейных моделях коэффициенты имеют физический смысл - они показывают значимость соответствующих признаков. Поэтому представляет особый интерес посмотреть коэффициенты вместе с названиями признаков.
Для этого соединим массив названий колонок из датасета и массив коэффициентов. Можно использовать, например, генераторное выражение для прохода по получившемуся массиву. Конструкция "_ = [...]" нужна только в ноутбуке для того, чтобы подавить автоматический вывод выражения:

```
_ = [print(k, v) for k, v in zip(X.columns, logistic.coef_[0])]
```

```
Pregnancies 0.11725233809260834
Glucose 0.03359984495281752
BloodPressure -0.014087375706652597
SkinThickness -0.0012704775628293837
Insulin -0.001240321623795555
BMI 0.07720233600112382
DiabetesPedigreeFunction 1.4190417369640222
Age 0.010035360773350831
```

Самостоятельно проанализируйте эти данные и сделайте вывод, какие атрибуты оказывают большее влияние на значение целевой переменной.
Как и в модели линейной регрессии, данный вектор не включает в себя свободный коэффициент. Он хранится в отдельном поле класса:

```
print("Intercept: \n", logistic.intercept_)
```

```
Intercept:
 [-7.70291388]
```

Теперь можно построить по полученной модели прогноз. Для этого передадим в соответствующий метод нашу матрицу признаков:

```
y_pred = logistic.predict(X)
```

Сформировав вектор предсказанных значений целевой переменной, можно сравнить его с реальными значениями:

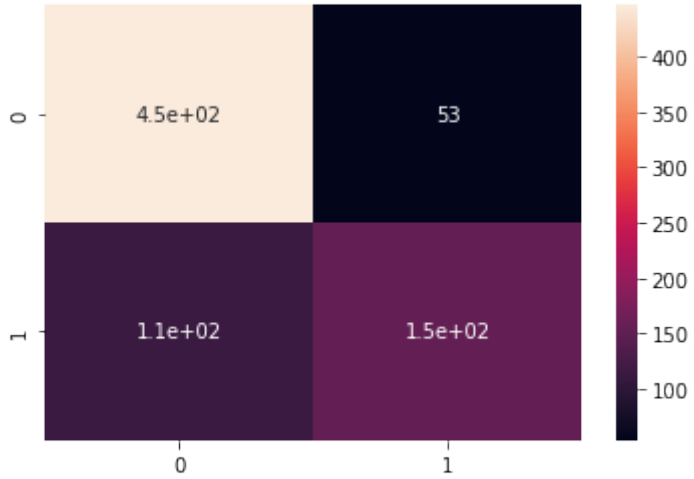
```
_ = [print(a, b) for a, b in list(zip(y, y_pred))[:10]]
```

Можно видеть, что большинство значений совпадает, но есть и ошибки - различия в значениях. Но так сравнивать все значения в ручном режиме очень неудобно. Поэтому лучше использовать специальные функции - метрики. Самая простая из них подсчитывает количество правильно и неправильно распознанных объектов и представляет результат в виде матрицы классификации:

```
from sklearn import metrics
metrics.confusion_matrix(y, y_pred)
```

Матрица классификации показывает нам очень полезную информацию: совместное распределение числа объектов предсказанных и реальных классов. Рассматривая эту матрицу, мы можем получить важную информацию: сколько объектов мы классифицировали правильно, сколько неверно, к каким классам наша модель тяготеет, какие классы распознаются хорошо, какие - плохо
Гораздо удобнее анализировать ту же информацию в графической форме. Для этого воспользуемся специальной библиотекой *seaborn*, которая позволяет создавать полезные для машинного обучения визуализации очень просто:

```
import seaborn as sns
sns.heatmap(metrics.confusion_matrix(y, y_pred), annot=True)
```



Кроме матрицы классификации весьма полезно использовать численные метрики эффективности классификации. Самая простая и распространенная из них - метрика точности предсказания - показывает долю правильно распознанных объектов. Расчет этой метрики встроен в сам объект модели и доступен с помощью специального метода:

logistic.score(X, y)

0.7825520833333334

Эту же метрику можно рассчитать и по-другому - через отдельную функцию из пакета *metrics*. Обратите внимание на другую сигнатуру метода:

metrics.accuracy_score(y_test, y_pred)

Значение метрики (0,78) показывает, что модель в среднем делает ошибки в 22% процентов случаев. Это основной показатель качества модели. В дальнейших работах мы покажем, как его замерять более правильно.

Если же такой уровень эффективности нас не устраивает, то мы можем попробовать использовать другие классы моделей классификации и среди них выбрать наиболее качественную. Например, можно попробовать построить полиномиальную модель. В библиотеке *sklearn* не предусмотрено отдельного класса полиномиальной модели. Ее можно создать через специальный объект *PolynomialFeature*, который добавляет полиномиальные признаки к данным. Для его использования сначала импортируем его:

from sklearn.preprocessing import PolynomialFeatures

Теперь можно создать объект преобразования (точно также как мы создавали объект модели):

poly = PolynomialFeatures(2)

Здесь мы указываем, что будем создавать полиномиальные признаки второго порядка. Теперь можно использовать этот объект для создания собственно самих признаков:

poly = poly.fit_transform(X)

poly

Этот код выводит следующие данные на экран:

```
array([[1.00000e+00, 6.00000e+00, 1.48000e+02, 7.20000e+01, 3.50000e+01,
        0.00000e+00, 3.36000e+01, 6.27000e-01, 5.00000e+00, 3.60000e+01,
        8.88000e+02, 4.32000e+02, 2.10000e+02, 0.00000e+00, 2.01600e+02,
        3.76200e+00, 3.00000e+02, 2.19040e+04, 1.06560e+04, 5.18000e+03,
        0.00000e+00, 4.97280e+03, 9.27960e+01, 7.40000e+03, 5.18400e+03,
        2.52000e+03, 0.00000e+00, 2.41920e+03, 4.51440e+01, 3.60000e+03,
        1.22500e+03, 0.00000e+00, 1.17600e+03, 2.19450e+01, 1.75000e+03,
        0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 1.12896e+03,
        2.10672e+01, 1.68000e+03, 3.93129e-01, 3.13500e+01, 2.50000e+03],
```

[1.00000e+00, 1.00000e+00, 8.50000e+01, 6.60000e+01, 2.90000e+01,
0.00000e+00, 2.66000e+01, 3.51000e-01, 3.10000e+01, 1.00000e+00,
8.50000e+01, 6.60000e+01, 2.90000e+01, 0.00000e+00, 2.66000e+01,
3.51000e-01, 3.10000e+01, 7.22500e+03, 5.61000e+03, 2.46500e+03,
0.00000e+00, 2.26100e+03, 2.98350e+01, 2.63500e+03, 4.35600e+03,
1.91400e+03, 0.00000e+00, 1.75560e+03, 2.31660e+01, 2.04600e+03,
8.41000e+02, 0.00000e+00, 7.71400e+02, 1.01790e+01, 8.99000e+02,
0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 7.07560e+02,
9.33660e+00, 8.24600e+02, 1.23201e-01, 1.08810e+01, 9.61000e+02]])

Теперь эти данные можно использовать как исходные для моделирования. А строить мы будем обычную логистическую регрессию:

```
polynomial = LogisticRegression()
polynomial.fit(poly, y)
y_pred_poly = polynomial.predict(poly)
```

Исследуйте точность этой модели и сравните ее с линейной самостоятельно.

Задания для самостоятельного выполнения

1. Изучите документацию *sklearn*, посвященную классу LogisticRegression. Какую еще информацию можно вывести для обученной модели? Попробуйте изменить аргументы при создании модели и посмотрите, как это влияет на качество предсказания.
2. Попробуйте применить к той же задаче другие модели классификации. Для каждой из них выведите матрицу классификации и оценку точности. Рекомендуется исследовать следующие модели:
    i. Метод опорных векторов
        a. Без ядра
        b. С линейным ядром
        c. С гауссовым ядром
        d. С полиномиальным ядром
    ii. Метод ближайших соседей
    iii. Многослойный перцептрон
    iv. Дерево решений
    v. Наивный байесовский классификатор
    vi. (*) Другие методы:
        a. Пассивно-агрессивный классификатор
        b. Гребневый классификатор
        c. Случайный лес
        d. Беггинг
        e. Другие модели по желанию
3. Напишите функцию, которая автоматически обучает все перечисленные модели и для каждой выдает оценку точности.
4. Повторите полностью анализ для другой задачи - распознавание вида ириса по параметрам растения (можно использовать метод sklearn.datasets.load_iris()).

Контрольные вопросы

1. Чем отличается применение разных моделей классификации в библиотеке *sklearn*?
2. Что показывает метрика точности регрессии?
3. Какое значение имеют коэффициенты логистической регрессии?

4. Что показывает матрица классификации?
5. Какие параметры имеет конструктор объекта логистической регрессии?
6. Какие атрибуты имеет объект логистической регрессии?
7. Какие параметры и атрибуты имеют объекты других моделей машинного обучения библиотеки *sklearn*?

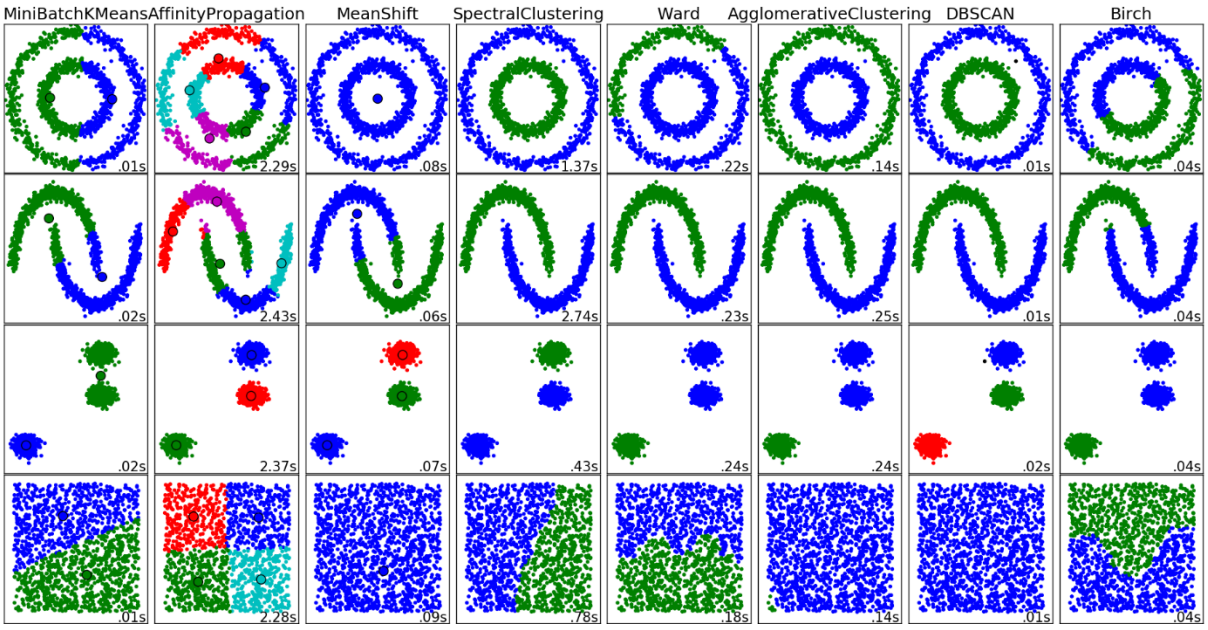
Лабораторная работа 3

Тема: Алгоритмы кластеризации

Цель работы: закрепление знаний, умений и навыков по работе с алгоритмами кластеризации.

Методические указания

Интуитивная постановка задачи кластеризации довольно проста и представляет из себя наше желание сказать: "Вот тут у меня насыпаны точки. Я вижу, что они сваливаются в какие-то кучки вместе. Было бы круто иметь возможность эти точки относить к кучкам и в случае появления новой точки на плоскости говорить, в какую кучку она падает." Из такой постановки видно, что пространства для фантазии получается много, и от этого возникает соответствующее множество алгоритмов решения этой задачи. Перечисленные алгоритмы ни в коем случае не описывают данное множество полностью, но являются примерами самых популярных методов решения задачи кластеризации.



Примеры работы алгоритмов кластеризации из документации пакета scikit-learn

Задание 1. Реализовать алгоритм K-means

Алгоритм К-средних, наверное, самый популярный и простой алгоритм кластеризации и очень легко представляется в виде простого псевдокода:

1. Выбрать количество кластеров inlinekinline, которое нам кажется оптимальным для наших данных.
2. Высыпать случайным образом в пространство наших данных inlinekinline точек (центроидов).
3. Для каждой точки нашего набора данных посчитать, к какому центроиду она ближе.
4. Переместить каждый центроид в центр выборки, которую мы отнесли к этому центроиду.
5. Повторять последние два шага фиксированное число раз, либо до тех пор покацен-

троиды не "сойдутся" (обычно это значит, что их смещение относительно предыдущего положения не превышает какого-то заранее заданного небольшого значения).

В случае обычной евклидовой метрики для точек лежащих на плоскости, этот алгоритм очень просто расписывается аналитически и рисуется. Давайте посмотрим соответствующий пример:

In [ ]:

```
# Начнём с того, что насыпем на плоскость три кластера точек
X =np.zeros((150, 2))

np.random.seed(seed=42)
X[:50, 0] =np.random.normal(loc=0.0, scale=0.3, size=50)
X[:50, 1] =np.random.normal(loc=0.0, scale=0.3, size=50)

X[50:100, 0] =np.random.normal(loc=2.0, scale=0.5, size=50)
X[50:100, 1] =np.random.normal(loc=-1.0, scale=0.2, size=50)

X[100:150, 0] =np.random.normal(loc=-1.0, scale=0.2, size=50)
X[100:150, 1] =np.random.normal(loc=2.0, scale=0.5, size=50)

plt.figure(figsize=(5, 5))
plt.plot(X[:, 0], X[:, 1], "bo");
```

In [ ]:

```
# В scipy есть замечательная функция, которая считает расстояния
# между парами точек из двух массивов, подающихся ей на вход
from scipy.spatial.distance import cdist

# Прибьём рандомность и насыпем три случайные центроиды для начала
np.random.seed(seed=42)
centroids =np.random.normal(loc=0.0, scale=1.0, size=6)
centroids =centroids.reshape((3, 2))

cent_history= []
cent_history.append(centroids)

for i in range(3):
# Считаем расстояния от наблюдений до центроид
distances=cdist(X, centroids)
# Смотрим, до какой центроиде каждой точке ближе всего
labels=distances.argmin(axis=1)

# Положим в каждую новую центроиду геометрический центр её точек
centroids =centroids.copy()
centroids[0, :] =np.mean(X[labels == 0, :], axis=0)
centroids[1, :] =np.mean(X[labels == 1, :], axis=0)
```

```
centroids[2, :] =np.mean(X[labels == 2, :], axis=0)

cent_history.append(centroids)
```
In [ ]:
```
# А теперь нарисуем всю эту красоту
plt.figure(figsize=(8, 8))
for i in range(4):
    distances =cdist(X, cent_history[i])
    labels =distances.argmin(axis=1)

plt.subplot(2, 2, i+ 1)
plt.plot(X[labels == 0, 0], X[labels == 0, 1], "bo", label="cluster #1")
plt.plot(X[labels == 1, 0], X[labels == 1, 1], "co", label="cluster #2")
plt.plot(X[labels == 2, 0], X[labels == 2, 1], "mo", label="cluster #3")
plt.plot(cent_history[i][:, 0], cent_history[i][:, 1], "rX")
plt.legend(loc=0)
plt.title("Step {:}".format(i+ 1));
```
Также стоит заметить, что хоть мы и рассматривали евклидово расстояние, алгоритм будет сходиться и в случае любой другой метрики, поэтому для различных задач кластеризации в зависимости от данных можно экспериментировать не только с количеством шагов или критерием сходимости, но и с метрикой, по которой мы считаем расстояния между точками и центроидами кластеров.
Другой особенностью этого алгоритма является то, что он чувствителен к исходному положению центроид кластеров в пространстве. В такой ситуации спасает несколько последовательных запусков алгоритма с последующим усреднением полученных кластеров.

Задание 2. Выбор числа кластеров для kMeans
В отличие от задачи классификации или регресии, в случае кластеризации сложнее выбрать критерий, с помощью которого было бы просто представить задачу кластеризации как задачу оптимизации. В случае kMeans распространен вот такой критерий – сумма квадратов расстояний от точек до центроидов кластеров, к которым они относятся.

$$J(C)=\sum k=1K\sum i \in Ck||xi-\mu k||\rightarrow minC,$$

здесь C – множество кластеров мощности K, μk – центроид кластера Ck.
Понятно, что здравый смысл в этом есть: мы хотим, чтобы точки распологались кучно возле центров своих кластеров. Но вот незадача: минимум такого фнукционала будет достигаться тогда, когда кластеров столько же, сколько и точек (то есть каждая точка – это кластер из одного элемента). Для решения этого вопроса (выбора числа кластеров) часто пользуются такой эвристикой: выбирают то число кластеров, начиная с которого описанный функционал J(C) падает "уже не так быстро". Или более формально:

$$D(k)=|J(Ck)-J(Ck+1)||J(Ck-1)-J(Ck)|\rightarrow mink$$

Рассмотрим пример.
In [ ]:
```
from sklearn.cluster import KMeans
```
In [ ]:
```
inertia = []
```

```
for k in range(1, 8):
kmeans=KMeans(n_clusters=k, random_state=1).fit(X)
inertia.append(np.sqrt(kmeans.inertia_))
In [ ]:
plt.plot(range(1, 8), inertia, marker="s")
plt.xlabel("$k$")
plt.ylabel("$J(C_k)$");
```

Видим, что J(Ck) падает сильно при увеличении числа кластеров с 1 до 2 и с 2 до 3 и уже не так сильно – при изменении k с 3 до 4. Значит, в данной задаче оптимально задать 3 кластера.

Сложности

Само по себе решение задачи K-means NP-трудное (NP-Hard, статья "Еще немного про P и NP" на Хабре), и для размерности d, числа кластеров k и числа точек n решается за O(ndk+1). Для решения такой боли часто используются эвристики, например MiniBatch K-means, который для обучения использует не весь датасет целиком, а лишь маленькие его порции (batch) и обновляет центроиды используя среднее за всю историю обновлений центроида от всех относящихся к нему точек. Сравнение обычного K-means и его MiniBatch имплементации можно посмотреть в документации scikit-learn.

Реализация алгоритма в scikit-learn обладает массой удобных плюшек, таких как возможность задать количество запусков через параметр n_init, что даст более устойчивые центроиды для кластеров в случае скошенных данных. К тому же эти запуски можно делать параллельно, не жертвуя временем вычисления.

Задание 3. Реализовать алгоритм кластеризации AffinityPropagation

Ещё один пример алгоритма кластеризации. В отличие от алгоритма К-средних, данный подход не требует заранее определять число кластеров на которое мы хотим разбить наши данные. Основная идея алгоритма заключается в том, что нам хотелось бы, чтобы наши наблюдения кластеризовались в группы на основе того, как они "общаются" или насколько они похожи друг на друга.

Заведём для этого какую-нибудь метрику "похожести", определяющуюся тем, что $s(x_i,x_j)>s(x_i,x_k)$ если наблюдение x_i больше похоже на наблюдение x_j, чем на x_k. Простым примером такой похожести будет отрицательный квадрат расстояния $s(x_i,x_j)=-||x_i-x_j||^2$.

Теперь опишем сам процесс "общения" для этого заведём две матрицы, инициализируемые нулями, одна из которых $r_{i,k}$ будет описывать насколько хорошо k-тое наблюдение подходит для того, чтобы быть "примером для подражания" для i-того наблюдения, относительно всех остальных потенциальных "примеров", а вторая — $a_{i,k}$ будет описывать насколько правильным было бы для i-того наблюдения выбрать k-тое в качестве такого "примера". Звучит немного запутанно, но чуть дальше увидим пример "на пальцах".

После этого данные матрицы обновляются по очереди по правилам:

$$r_{i,k} \leftarrow s(x_i,x_k) - \max_{k' \neq k}\{a_{i,k'} + s(x_i,x_{k'})\}$$

$$a_{i,k} \leftarrow \min\left(0, r_{k,k} + \sum_{i' \notin \{i,k\}} \max(0, r_{i',k})\right), \quad i \neq k$$

$$a_{k,k} \leftarrow \sum_{i' \neq k} \max(0, r_{i',k})$$

Задание 4. Реализовать спектральную кластеризацию

Спектральная кластеризация объединяет несколько описанных выше подходов, чтобы получить максимальное количество профита от сложных многообразий размерности меньшей исходного пространства.
Для работы этого алгоритма нам потребуется определить матрицу похожести наблюдений (adjacencymatrix). Можно это сделать таким же образом, как и для AffinityPropagation выше: Ai,j=−||xi−xj||2. Эта матрица также описывает полный граф с вершинами в наших наблюдениях и рёбрами между каждой парой наблюдений с весом, соответствующим степени похожести этих вершин. Для нашей выше выбранной метрики и точек, лежащих на плоскости, эта штука будет интуитивной и простой — две точки более похожи, если ребро между ними короче. Теперь нам бы хотелось разделить наш получившийся граф на две части так, чтобы получившиеся точки в двух графах были в общем больше похожи на другие точки внутри получившейся "своей" половины графа, чем на точки в "другой" половине. Формальное название такой задачи называется Normalizedcutsproblem и подробнее про это можно почитать тут.

Лабораторная работа 4

Тема: Деревья решений, линейные классификаторы

Цель работы: Научиться применять модель дерева принятия решений для задач классификации и регрессии.

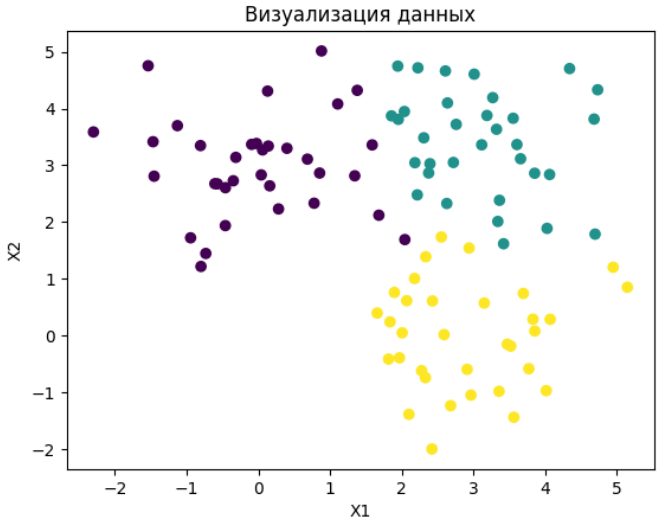
Методические указания

Задача классификации

Создадим и визуализируем датасет:

```
X, y = make_blobs(n_samples=100, centers=[(0,3),(3,3),(3,0)],
n_features=2, random_state=RANDOM_SEED,
cluster_std=(0.9,0.9,0.9))
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.title('Визуализацияданных')
plt.xlabel('X1')
plt.ylabel('X2')
```

Видим набор данных из трех классов:



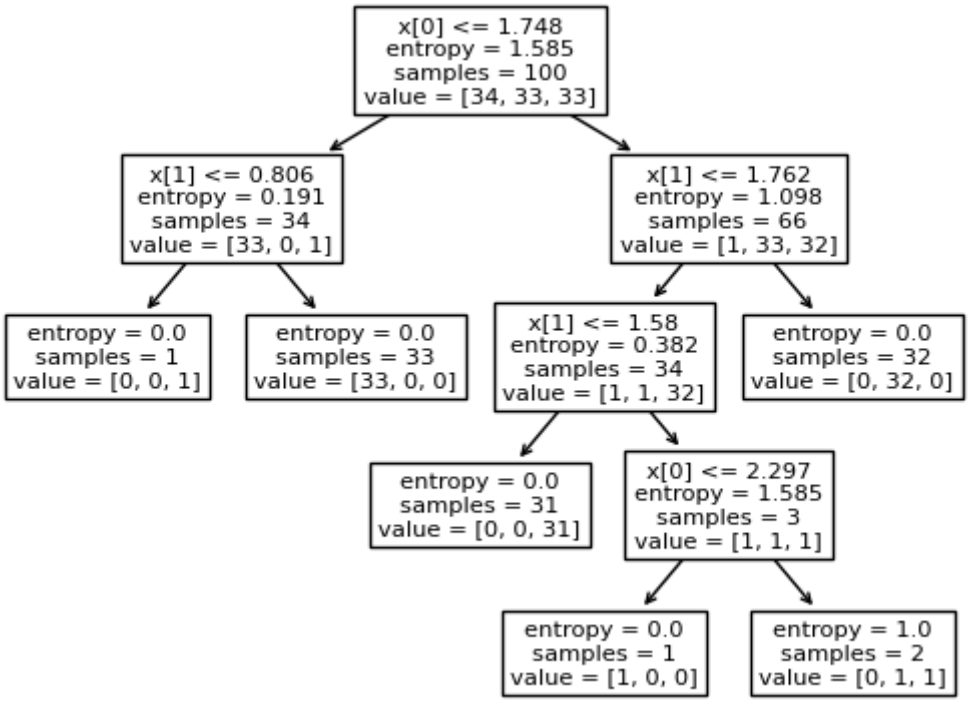
Также, как и перцептрон, деревья решений могут легко применяться к задачам разной размерности. Количество классов не играет роли для дерева. Содадим и обучим модель:

```
depth=4
clf_tree = DecisionTreeClassifier(criterion='entropy', max_depth=depth,
random_state=RANDOM_SEED)
clf_tree.fit(X, y)
```

Здесь мы задаем вид функции, которая будет использоваться для нахождения оптимальной границы разбиения выборки. По умолчанию используется критерий Джини, а мы сейчас будем использовать критерий информационной энтропии. Также мы задает максимальную глубину дерева. Дерево не будет "расти" дальше этого количества уровней. Вы самостоятельно можете изменить значения этих параметров и проанализировать, как это повлияет на рабоут модели.

После обучения данной модели мы можем вывести собственно само дерево решений, то есть полное внутреннее устройство модели. Для этого в объекте существует специальный

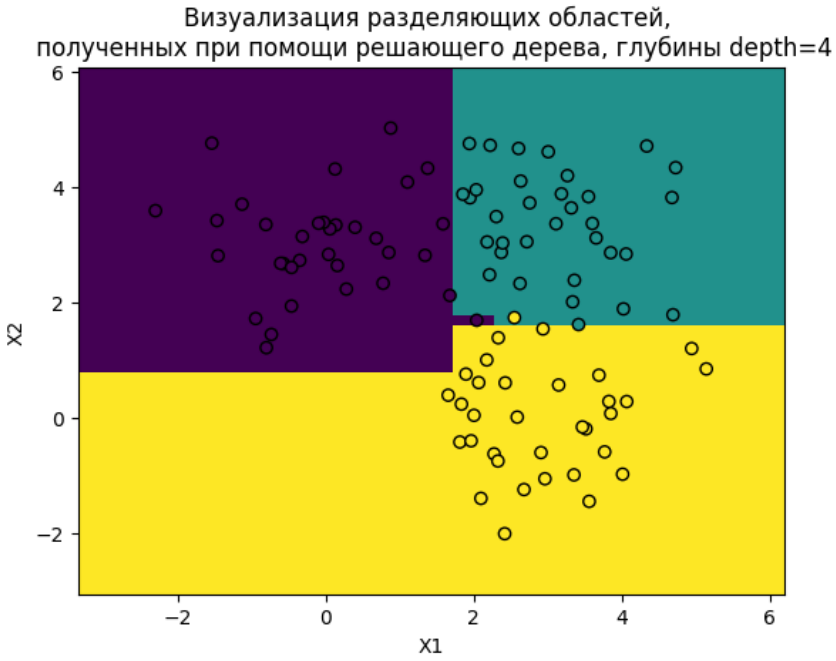
метод *plot_tree()*:
tree.plot_tree(clf_tree)
plt.show()
В этом объекте есть еще один метод визуализации дерева - втекстовом виде. Самостоятельно найдите и примените его. А использованный нами метод выводит дерево в графическом виде:



Проинтерпретируйте изображенную на этом графике информацию.
Теперь можно визуализировать сами границы принятия решений:

```
X0 = np.linspace(X[:, 0].min()-1,X[:, 0].max()+1, X.shape[0])
X1 = np.linspace(X[:, 1].min()-1,X[:, 1].max()+1, X.shape[0])
X0_grid, X1_grid = np.meshgrid(X0, X1)

y_predict = clf_tree.predict(np.c_[X0_grid.ravel(),X1_grid.ravel()]).reshape(X0_grid.shape)
plt.pcolormesh(X0_grid, X1_grid, y_predict)

plt.scatter(X[:, 0], X[:, 1], c=y,  edgecolors='black',linewidth=1)

plt.title('Визуализация разделяющих областей, \n полученных при помощи решающего дерева, глубины depth={}'.format(depth))
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```

Мы видим очень характерную для деревьев форму границы:



Как всегда оценим качество работы модели и с помощью метрик:

```
y_pred = clf_tree.predict(X)
print(confusion_matrix(y, y_pred))
print('Accuracy =', accuracy_score(y, y_pred))
print('F1_score =', f1_score(y, y_pred, average='micro'))
```

```
[[34  0  0]
[ 0 33  0]
[ 0  1 32]]
Accuracy = 0.99
F1_score = 0.99
```

Мы видим, что модель допустила всего одну ошибку на 100 примерах, что довольно неплохо. Но увеличив глубину дерева можно еще повысить его эффективность и добиться абсолютной точности.

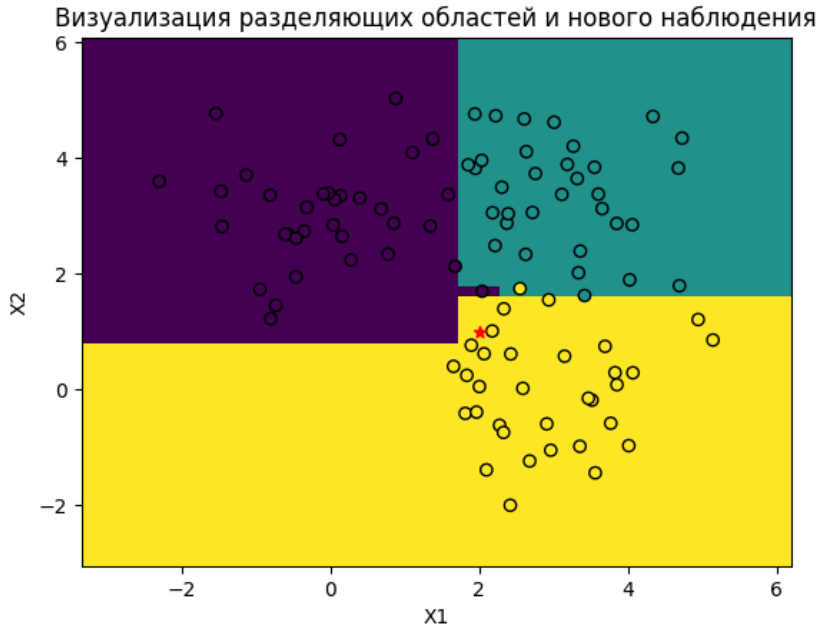
Теперь создадим новое наблюдение:

```
observation_new = [[2, 1]]
```

Иклассифицируемего:

```
clf_tree.predict(observation_new)
```

Визуализируем на графике его вместе с границей принятия решений:

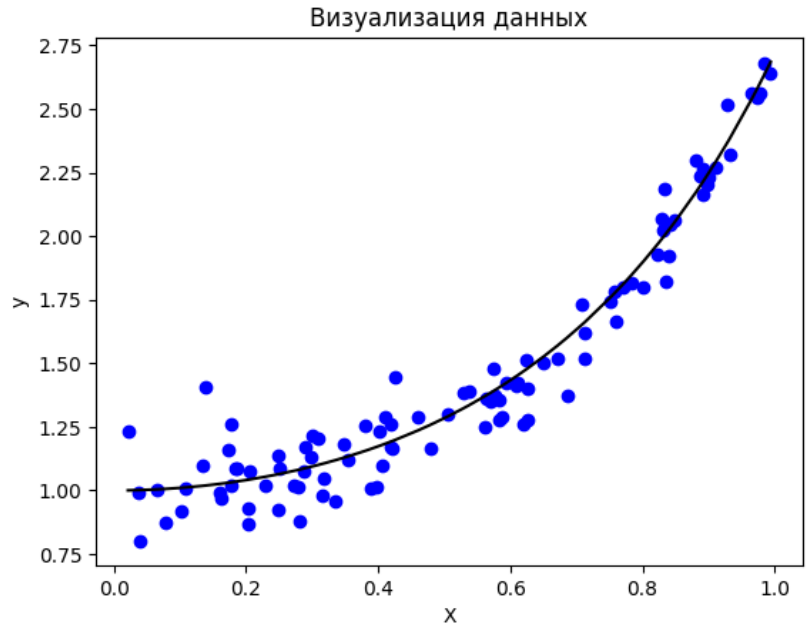


Задача регрессии

Сгенерируем простой датасет для задачи регрессии. Для этого получим 100 упорядоченных случайных чисел, а затем значения целевой переменной вычислим как результат какой-нибудь функции (например возьмем экспоненту) и прибавим к результату случайный шум, для имитации разброса:

```
n_samples = 100
X = np.sort(np.random.rand(n_samples))
y = np.exp(X ** 2)+np.random.normal(0.0, 0.1, X.shape[0])
```

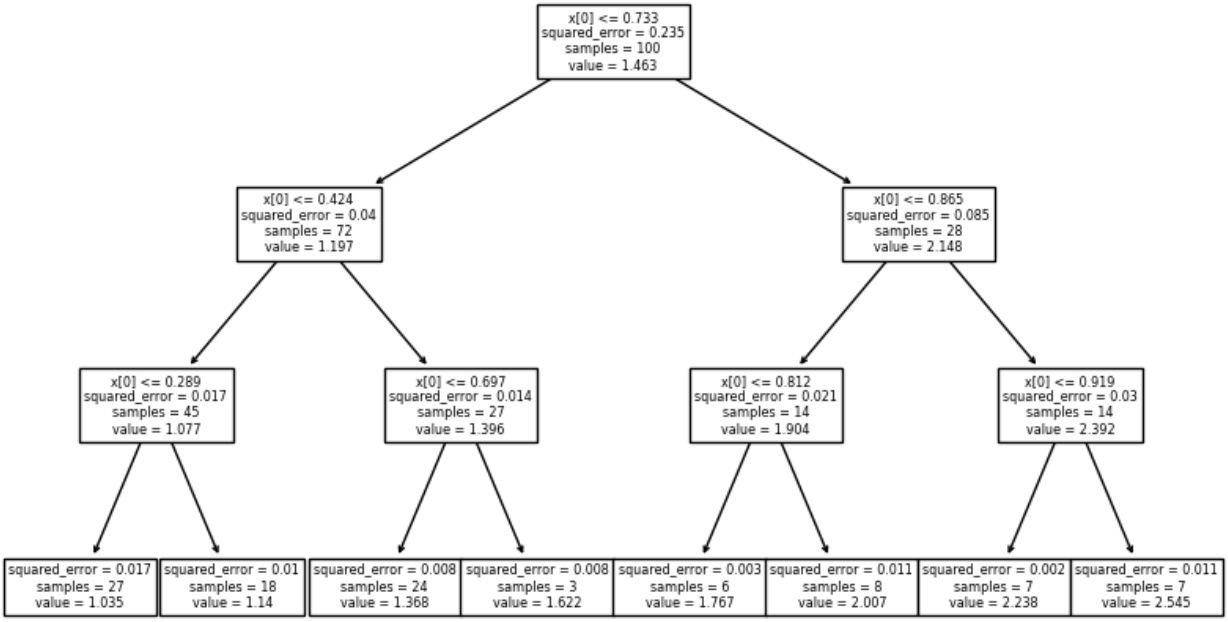
Мы получим примерно такой датасет для парной регрессии (линия здесь добавлена только для информации и наглядности, она не является частью данных):



Создадим объект регрессора на основе деревьев решений и обучим его:

```
depth=3
reg_tree = DecisionTreeRegressor(max_depth=depth, random_state=RANDOM_SEED)
reg_tree.fit(X, y)
```

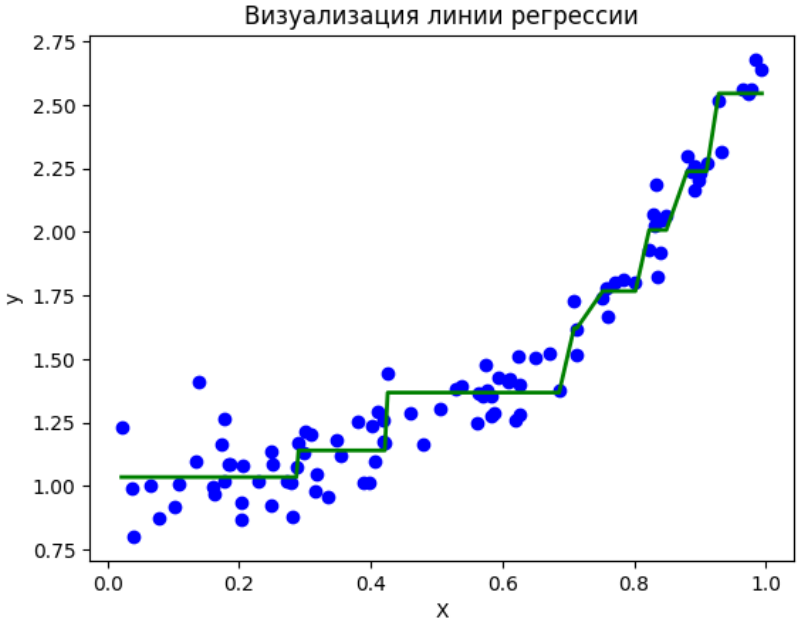
Выведем полученное дерево:



Самостоятельно проинтерпретируйте информацию в этом графе. А мы построим линию регрессии на графике::
plt.scatter(X, y, c="b")

plt.plot(X, y_pred_reg, "g", lw=2)

plt.title('Визуализация линии регрессии')
plt.xlabel('X')
plt.ylabel('y');
Опять видим характерную кусочно-линейную линию:



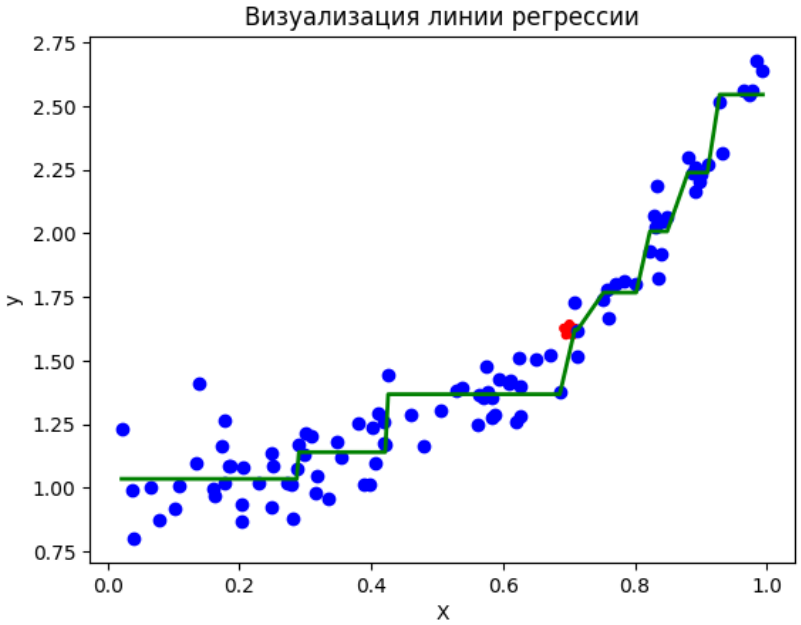
Тперь как всегда оценим качество работы модели:

```
print('r2= ', r2_score(y, y_pred_reg))
print('MSE= ', mean_squared_error(y, y_pred_reg))
r2=  0.9559340585697235
MSE=  0.010347452773751169
```

Данные метрики свидетельствуют о довольно точной работе модели. Лишь на графике открывается определенная условность и неглабкость полученных предсказаний. Давайте создадим новое предсказание:

```
observation_new = [[0.7]]
```

И изобразим его на графике:



Самостоятельно подвигайте данную точку. Обратите внимание, что есть участки, где небольшое изменение положения точки не изменяет результат предсказания.

Задания для самостоятельного выполнения

1. Загрузите встроенные данные sklearn.datasets.load_iris, взяв только последние два столбца (длина и ширина лепестков). Изобразите их на диаграмме рассеяния, подкрасив каждый класс некоторым цветом.
2. Обучите модель классификационного дерева принятия решений sklearn.tree.DecisionTreeClassifier глубины 4, используя энтропию. Визуализируйте соответствующий граф дерева принятия решений.
3. Обучите модель классификационного дерева принятия решений sklearn.tree.DecisionTreeClassifier с разными глубинами (1, 2, 3, 4, 10), используя энтропию, и визуализируйте в каждом случае полученные разделяющие области.
4. Выведите необходимые метрики для оценки работы моделей с разными глубинами. Сделайте вывод о том, какая модель лучше классифицирует данные.
5. Загрузите весь датасет load_iris. Обучите модель классификационного дерева принятия решений sklearn.tree.DecisionTreeClassifier глубины 4, используя энтропию. Визуализируйте соответствующий граф дерева решений. Оцените качество работы модели.
6. Загрузите встроенные данные sklearn.datasets.california_housing, взяв только столбец AveBedrms в качестве единственного признака. Изобразите данные на диаграмме рассеяния так, чтобы на одной оси были отмечены значения признака, а на другой - целе-

вой переменной.

7. Обучите модель регрессионного дерева принятия решений sklearn.tree.DecisionTreeRegressor, зафиксировав random_state=0, а остальными гиперпараметрами по умолчанию.
8. Визуализируйте соответствующий граф дерева решений и получившуюся кусочную линию регрессии.
9. Оцените качество работы модели. Создайте новое наблюдение и сделайте предсказание на нём.
10. Загрузите весь датасетfetch_california_housing. Обучите ту же модель. Визуализируйте соответствующий граф дерева решений и оцените качество работы модели.

Контрольные вопросы

1. Почему граница принятия решений у деревьев решений имеет такую характерную форму?
2. Как глубина дерева влияет на сложность модели?
3. Почему глубина дерева на разных ветках может быть разная?
4. Что такое критерий в деревьях решений и как он влияет на работу модели?

Дополнительные задания

1. Проверьте работу модели на ненормализованных данных с очень разной величиной. Сделайте вывод о применимости модели дерева решений без нормализации
2. Примените дерево решений на большом датасете (не менее 10 000 строк и 10 столбцов) по вашему выбору.
3. Повторите моделирование с другим критерием. Найдите датасеты, на которых лучше работают разные критерии.
4. Используйте на первом датасете вместо деревьев решений случайный лес. Визуализируйте его границу принятия решений.

Лабораторная работа5

Тема: Нейронные сети

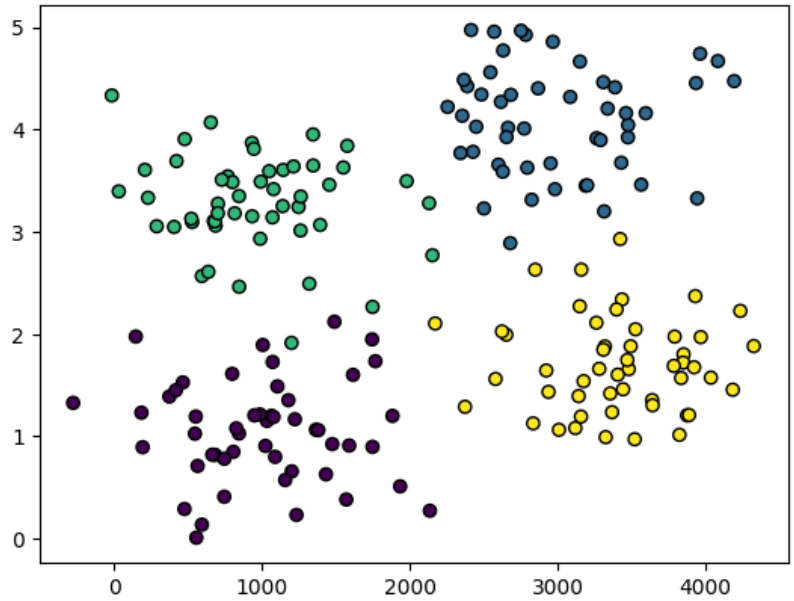
Цель работы: Познакомиться с перцептроном как с моделью обучения с учителем в библиотеке sklearn.

Методические указания

Для тренировки работы с перцептроном создадим набор данных. Воспользуемся уже знакомыми функциями генерации данных. Сразу после создания визуализируем этот набор данных на диаграмме рассеяния:

```
blob_centers = ([1, 1], [3, 4], [1, 3.3], [3.5, 1.8])
X, y = make_blobs(n_samples=200,
                  centers=blob_centers,
cluster_std=0.5,
random_state=0)

X[:, 0] *= 1000

plt.scatter(X[:, 0], X[:, 1], c=y,  edgecolors='black',linewidth=1)
```

Вы должны увидеть набор данных, состоящий из четырех кластеров, принадлежащий четырем классам:



То есть мы имеем дело с задачей множественной классификации. Для перцептрона это не проблема, нейросеть это очень универсальная модель, она может работать нативно и в множественном режиме. Давайте создадим первую версию нашей нейросети:

```
clf = MLPClassifier(hidden_layer_sizes=(6,),
random_state=1).fit(X, y)
```

Обратите внимание, что при создании мы задаем количество слоев и количество нейронов в каждом из них с помощью кортежа *hidden_layer_sizes*. В данном случае имеем один

скрытый слой в 6 нейронов. Если мы не зададим этот аргумент, то его значение по умолчанию - (100,), то есть один слой со ста нейронами. Этого для первой модели будет довольно много.
Так как мы уже обучили модель, мы можем посмотреть значения весов нейронов, то есть внутренние параметры нейросети. Все они хранятся в поле *coefs_*, аналогично другим моделям машинного обучения в sklearn. Например, можно вывести веса отдельно для каждого слоя:

```
print("Веса между входным и скрытым слоем:")
print(clf.coefs_[0])
print("\nВеса между скрытым и выходным слоем:")
print(clf.coefs_[1])
```

```
Веса между входным и скрытым слоем:
[[-0.13186096  0.36276684 -0.87753229 -0.3544923  -0.62342741 -0.69458326]
 [-0.55527141 -0.28633324 -0.1793465   0.07807555 -0.14370807  0.32689274]]

Веса между скрытым и выходным слоем:
[[-0.4955391  -0.41463226  0.41301641  0.65040935 -0.25197168  0.26165608]
[ 0.5134373   0.53925562 -0.57325313 -0.6379063  -0.4542433   0.51596208]
 [-0.55456344 -0.10522807  0.63360879  0.03543427  0.28300565 -0.25076137]
[ 0.25356345  0.4604906  -0.66711811  0.3423877   0.70293671  0.33948522]
 [-0.29956981  0.3969661  -0.54770982 -0.0851425   0.58948128 -0.28122352]
 [-0.28935468 -0.51007606 -0.6656005   0.24156416 -0.39625967 -0.3203444 ]]
```

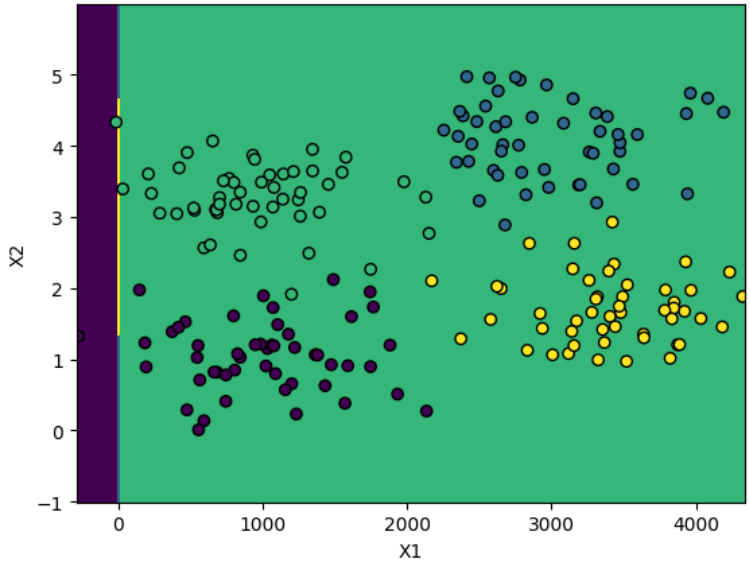
Либо по каждому нейрону отдельно:

```
for i in range(len(clf.coefs_)):
number_neurons_in_layer = clf.coefs_[i].shape[1]
    for j in range(number_neurons_in_layer):
        weights = clf.coefs_[i][:,j]
print(i, j, weights, end=", ")
print()
print()
```

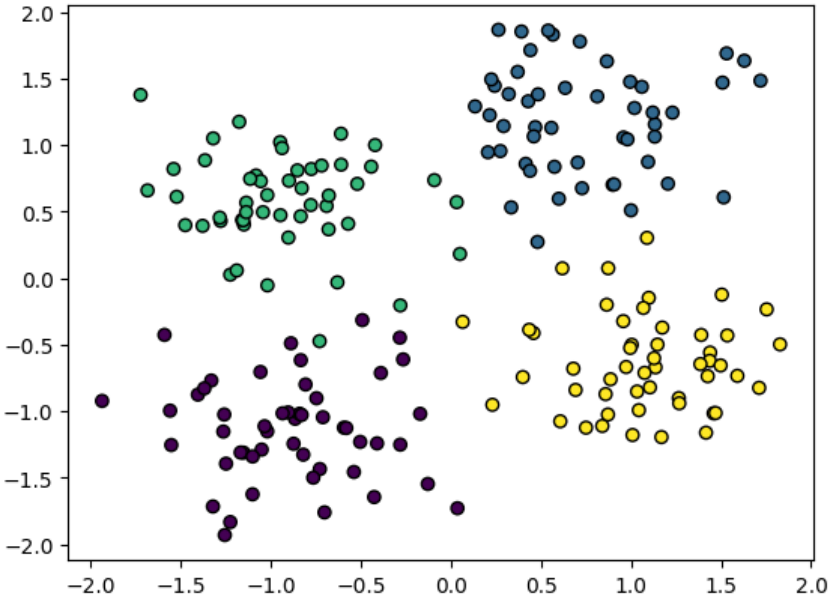
```
0 0 [-0.09809877 -0.58903611],
0 1 [ 0.33667051 -0.31219238],
0 2 [-0.82024119 -0.22424296],
0 3 [-0.29720839  0.11423652],
0 4 [-0.56852789 -0.18324369],
0 5 [-0.66056109  0.37321781],

1 0 [-0.51316658 -0.24370005 -0.59713359 -0.5769702   0.34285864 -0.7006119 ],
1 1 [-0.43365467  0.34328608 -0.67678056 -0.10221922 -0.25645017  0.35515533],
1 2 [ 0.43189995  0.57835494 -0.46609801  0.75476969  0.33417991  0.80295119],
1 3 [0.68148414 0.57889271 0.54011704 0.00610289 0.47281254 0.33880635],
```

Обратите внимание, что судя по выведенной информации, на выходном слое у нас четыре нейрона. Это обусловлено тем, что в датасете четыре класса. Количество классов всегда определяет количество нейронов на выходном слое. А на входном слое у нас два нейрона. Это потому, что в датасете всего две атрибута.

Веса нейронов смещения (bias) как и в других моделях хранятся в отдельном поле:
```
print("Веса смещения для скрытого слоя:")
print(clf.intercepts_[0])
print("\nВеса смещения для выходного слоя:")
print(clf.intercepts_[1])
```
```
Веса смещения для скрытого слоя:
[-0.55752645  0.60978582 -0.8640854   0.33258994 -0.18652436  0.05696655]

Веса смещения для выходного слоя:
[-0.29413473  0.49414359 -0.60792984 -0.11888525]
```
Давайте проанализируем, насколько хорошо работает наша модель. Для этого воспользуемся уже знакомой матрицей классификации:
```
confusion_matrix(y, clf.predict(X))
array([[ 1,  0, 49,  0],
[ 0,  0, 50,  0],
[ 1,  0, 49,  0],
[ 0,  0, 50,  0]])
```
Мы видим, что как будто модель работает не совсем корректно и делает много ошибок. Чтобы убедиться в этом, изобразим классификацию на графике:
```
X0 = np.linspace(X[:, 0].min()-1,X[:, 0].max()+1, X.shape[0])
X1 = np.linspace(X[:, 1].min()-1,X[:, 1].max()+1, X.shape[0])
X0_grid, X1_grid = np.meshgrid(X0, X1)

y_predict = clf.predict(np.c_[X0_grid.ravel(),X1_grid.ravel()]).reshape(X0_grid.shape)
plt.pcolormesh(X0_grid, X1_grid, y_predict)

plt.scatter(X[:, 0], X[:, 1], c=y,  edgecolors='black',linewidth=1)
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```
На примере данного кода познакомьтесь со способом изображения границы принятия решений для множественной классификации. Мы видим довольно странную картину:

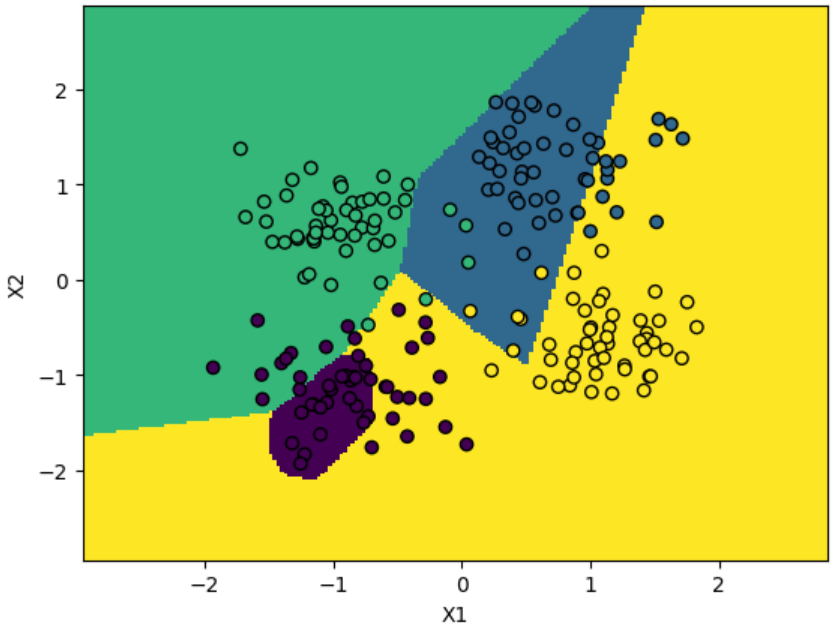


Почти все точки модель относит к одному классу. А граница принятия решений почти всегда вертикальна. Что мы сделали не так? Для ответа на этот вопрос надо обратить внимание на подпись осей координат. Дело в том, что атрибуты в исходных данных имеют очень разные значения. X1 измеряется в тысячах, а X2 - в единицах. Это может стать проблемой для некоторых моделей машинного обучения. И перцептрон - одна из них. Для корректной работы перцептрона данные нужно обязательно нормализовать.
Для этого импортируем и воспользуемся объектом нормализации данных. Например, таким:
from sklearn.preprocessing import StandardScaler
X_scaled = scaler.transform(X)
X_scaled = scaler.transform(X)
Давайте еще раз изобразим точки на графике, чтобы понять, как нормализация преобразовала наш датасет:

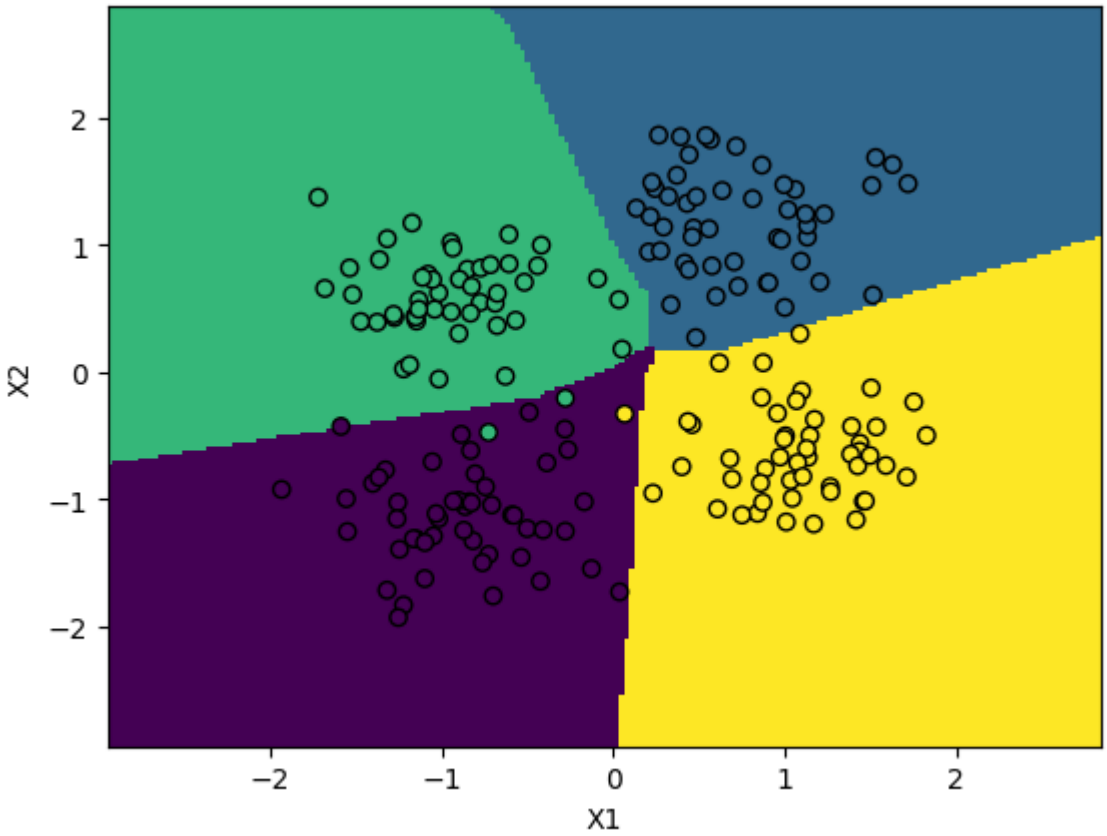


Само распределение как будто не изменилось, но обратите внимание на значения по осям. Теперь наш датасет имеет одинаковую размерность. Данные как будто "ужались" по горизонтали. Давайте посмотрим, повлияет ли это на качество обучения. Переобучим модель и выведем матрицу классификации:

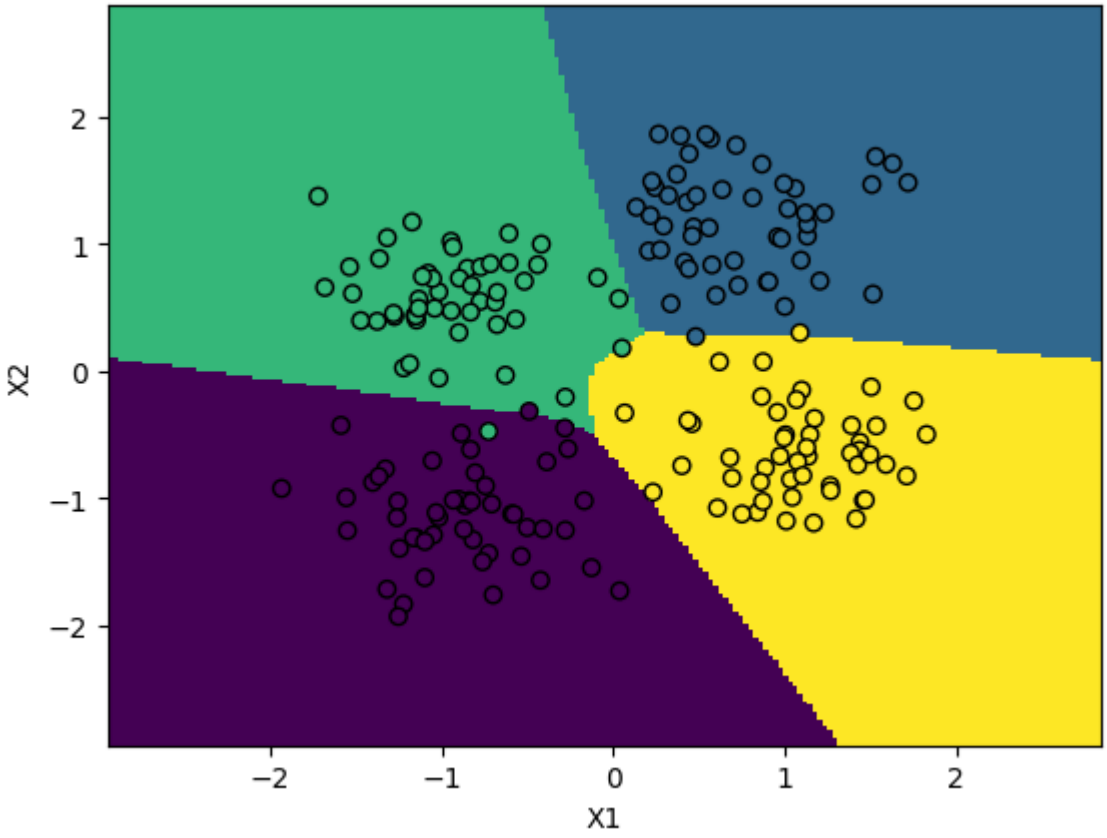
array([[22, 0, 11, 17],
[0, 38, 0, 12],
[0, 3, 45, 2],
[0, 5, 0, 45]])
Мы видим совершенно другую картину. Стало значительно лучше. Для большей уверенности изобразим классификацию на графике:



Мы видим, что регионы соответствующих классов уже гораздо ближе к точкам обучающей выборки. При этом модель все еще делает довольно много ошибок. Как можно просто ее улучшить? вы могли обратить внимание, что при обучении модель выдавал предупреждение о раннем прерывании обучения. Мы уже сталкивались с такой ситуацией. Можно просто увеличить лимит итераций алгоритма обучения:
clf = MLPClassifier(hidden_layer_sizes=(6,), max_iter=10_000, verbose=True).fit(X_scaled, y)
Заодно здесь мы применим еще один аргумент конструктора объекта перцептрона - *verbose*. Он позволяет вывести подробную информацию про обучение модели. Проанализируйте полученную информацию и сделайте вывод.
Теперь можно вывести матрицу классификации. Сразу видно, что опять стало значительно лучше:
array([[50, 0, 0, 0],
[0, 50, 0, 0],
[2, 3, 45, 0],
[0, 1, 0, 49]])
Этот вывод подтверждает и визуализация модели:



Для дальнейшего повышения точности можно увеличить количество скрытых слоев. Тогда получится уже глубокая нейросеть. Например, обучим перцептрон с тремя слоями:
clf = MLPClassifier(hidden_layer_sizes=(6, 6, 6), max_iter=10_000).fit(X_scaled, y)



Из графика видно, что граница принятия решений еще больше соответствует обучающей

выборке. Постройте матрицу классификации этой модели самостоятельно.

Задания для самостоятельного выполнения

1. Создайте однослойный перцептрон с 1, 2, 10 и 100 нейронами. Сравние их точность и сделайте вывод о достаточном количестве нейронов.
2. Создайте и оцените модель с двумя, тремя и десятью скрытыми слоями с одинаковым количеством нейронов. Сравните их точность и сделайте вывод о достаточном количестве слоев.
3. Для глубокой модели выведите веса всех нейронов на всех слоях. Выведите значения векторов весов смещения.
4. Постройте и оцените модель с большим количеством нейронов и слоев. Замерьте время выполнения обучения, сравните со временем обучения более простых моделей.
5. Постройте и оцените модель классификации с помощью перцептрона на датасете, который вы использовали на контрольной по классификиации (если вы ее не выполняли, возьмите любой датасет из раздела "realworlddatasets" в библиотеке sklearn).
6. Постройте и оцените модель регрессии с помощью перцептрона на датасете, который вы использовали на контрольной по регрессии.

Контрольные вопросы

1. Что называют глубокой нейронной сетью?
2. Что такое архитектура нейронной сети?
3. Как количество нейронов и слоев влияет на качество моделирования?
4. Как нейронная сеть решает задачи множественной классификации?
5. В каких случаях следует применять перцептрон?

Дополнительные задания

1. Повторите моделирование на одном из датасетов, но с использованием библиотек Tensoflow, PyTorch или Keras. Сравните результаты с реализацией перцептрона в библиотеке sklearn.
2. Поэкспериментируйте с разными функциями активации и методами оптимизации. Найдите датасеты, на которых лучше работают разные функции активации и методы оптимизации.

Лабораторная работа 6

Тема: Регрессионный анализ

Цель работы

Познакомиться с основными приемами работы с моделями регрессии в scikit-learn.

Задания для выполнения

1. Загрузите встроенные датасет о ценах на недвижимость в Калифорнии.
2. Постройте модель регрессии для предсказания цены конкретного объекта.
3. Оцените качество построенной модели с помощью визуализации и коэффициента детерминации.
4. Постройте альтернативную полиномиальную модель, сравните ее с предыдущей.

Методические указания

В качестве исходных данных будем использовать датасет о ценах на объекты недвижимости в Калифорнии. Это один из известных обучающих наборов данных. Он встроен в библиотеку *sklearn*, так что его не нужно загружать или скачивать отдельно. Для начала работы импортируем стандартные необходимые библиотеки:

importnumpyasnp
import matplotlib.pyplot as plt
import pandas as pd

Сперва загрузим исходный набор данных:

from sklearn.datasets import fetch_california_housing
california = fetch_california_housing()

Сперва следует ознакомиться со структурой тех данных, которые мы получили. Для этого выведем тип полученного объекта:

type(california)
sklearn.utils.Bunch

Это специальный тип данных библиотеки *sklearn*, который похож по своему устройству на обычный словарь. Поэтому посмотрим, какие ключи есть в этом словаре:

california.keys()
dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESCR'])

Особый интерес здесь представляют поля *data* и *target*, которые содержат именно исходные атрибуты и вектор значений целевой переменной. Выведемихтип:

print(type(california.data), type(california.target))
<class 'numpy.ndarray'><class 'numpy.ndarray'>

Так как это массивы *numpy*, то можно посмотреть их форму:

print(california.data.shape, california.target.shape)
(20640, 8) (20640,)

Получается, что в данных более 20 тысяч строк и 8 атрибутов. Дополнительно можно еще вывести описание датасета для получения дополнительной информации.

Теперь с данными можно работать разными способами. Для удобства анализа мы объединим все массивы в датафрейм:

data = pd.DataFrame(california.data, columns = california.feature_names)
data['Price'] = california.target
data.head()

index	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	8.3252	41.0	6.984126984126984	1.0238095238095237	322.0	2.5555555555555554	37.88	-122.23	4.526
1	8.3014	21.0	6.238137082601054	0.9718804920913884	2401.0	2.1098418277768014	37.86	-122.22	3.585
2	7.2574	52.0	8.288135593220339	1.0734463276836160	496.0	2.8022598870056497	37.85	-122.24	3.521
3	5.6431	52.0	5.817351598173515	1.0730593607305936	558.0	2.5479452054794520	37.85	-122.25	3.413
4	3.8462	52.0	6.281853281853282	1.0810810810810811	565.0	2.1814671814671813	37.85	-122.25	3.422

Проверим данные на наличие пропущенных значений:

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   Price       20640 non-null  float64
dtypes: float64(9)
memoryusage: 1.4 MB
```

Видим, что пропусков в данных нет. Кроме того, видно, что все данные выражены в численных шкалах. Значит, особенной обработки данный датасет не требует, он уже достаточно чистый. Теперь можно вывести основную статистику по датасету:

```
data.describe().round(2)
```

index	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
count	20640.0	20640.0	20640.0	20640.0	20640.0	20640.0	20640.0	20640.0	20640.0
mean	3.87	28.64	5.43	1.1	1425.48	3.07	35.63	-119.57	2.07
std	1.9	12.59	2.47	0.47	1132.46	10.39	2.14	2.0	1.15
min	0.5	1.0	0.85	0.33	3.0	0.69	32.54	-124.35	0.15
25%	2.56	18.0	4.44	1.01	787.0	2.43	33.93	-121.8	1.2
50%	3.53	29.0	5.23	1.05	1166.0	2.82	34.26	-118.49	1.8
75%	4.74	37.0	6.05	1.1	1725.0	3.28	37.71	-118.01	2.65
max	15.0	52.0	141.91	34.07	35682.0	1243.33	41.95	-114.31	5.0

Теперь выделим целевую переменную и факторы:

```
y = data['Price']
X = data.drop('Price', axis=1)
```

Выведем форму получившихся массивов:

```
y.shape, X.shape
((442,), (442, 10))
```

Приступим к обучению и оценке качества модели. Из набора линейных моделей библиотеки *sklearn* импортируем линейную регрессию:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X, y)
```

Как и в предыдущих работах выведем коэффициенты модели, так как в линейных моделях они имеют некоторый смысл:

```
print("Coefficients: \n", model.coef_)
Coefficients:
 [ 4.36693293e-01  9.43577803e-03 -1.07322041e-01  6.45065694e-01
-3.97638942e-06 -3.78654265e-03 -4.21314378e-01 -4.34513755e-01]
```

Выведем коэффициенты вместе с названиями соответствующих атрибутов:

```
_ = [print(k, v) for k, v in zip(X.columns, model.coef_)]
MedInc 0.4366932931343245
HouseAge 0.009435778033237972
AveRooms -0.10732204139090447
AveBedrms 0.645065693519812
Population -3.976389421211576e-06
AveOccup -0.003786542654971
Latitude -0.42131437752714385
```

Longitude -0.43451375467477743

Проанализируйте эти данные вместе с интерпретацией атрибутов датасетов и сделайте вывод о том, какие факторы как влияют на цену недвижимости в Калифорнии.

Как и в модели линейной регрессии, данный вектор не включает в себя свободный коэффициент. Он хранится в отдельном поле класса:

```
print("Intercept: \n", model.intercept_)
```

Intercept:

-36.94192020718441

Сделаем предсказания модели и выведем на экран первые несколько точек:

```
y_pred = model.predict(X)
print(y_pred[:5])
```

[4.13164983 3.97660644 3.67657094 3.2415985 2.41358744]

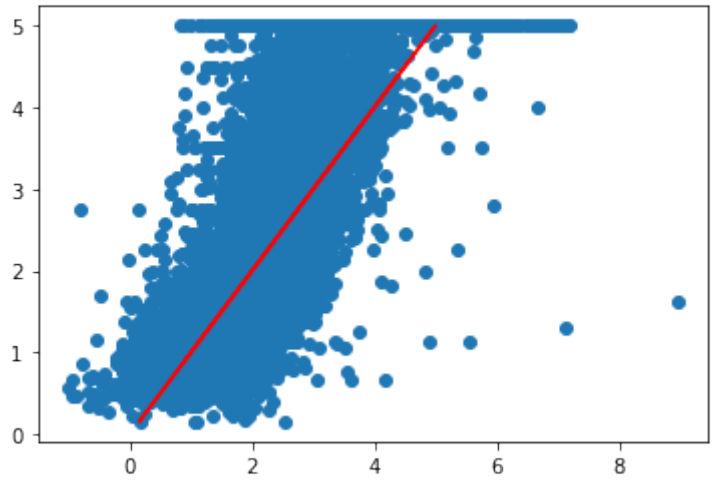
Для сравнения выведем реальные соответствующие значения целевой переменной:

```
print(y[:5])
```

0 4.526
1 3.585
2 3.521
3 3.413
4 3.422
Name: Price, dtype: float64

Конечно, так анализировать данные неудобно. Лучше построить график, демонстрирующий связь между реальными и предсказанными значениями целевой переменной:

```
plt.scatter(y_pred, y)
plt.plot(y, y, c='r')
```

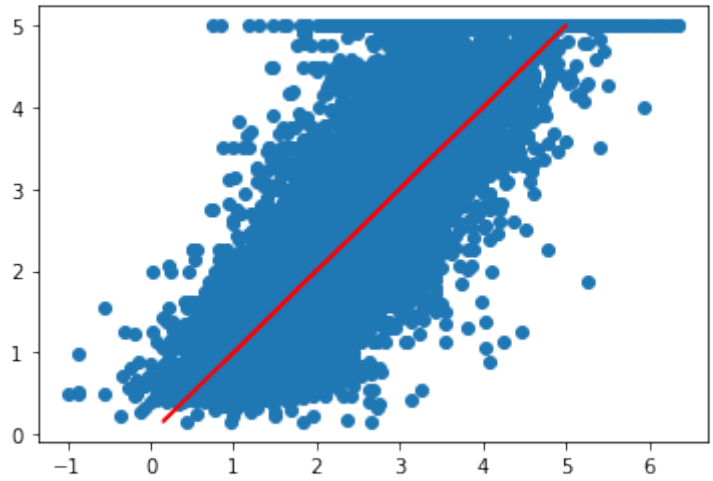


На этом графике чем ближе точки к центральной линии, тем более точные прогнозы делает модель. В данном случае разброс довольно велик. Чтобы оценить эффективность модели численно, опять обратимся к встроенной метрике, реализованной методом *score* - коэффициенту детерминации:

```
model.score(X, y)
```

0.606232685199805

Уровень 0.6 показывает, что модель могла бы быть более точной. Давайте попробуем построить другую модель - полиномиальную регрессию. Есть надежда, что введение полиномиальных признаков может существенно увеличить точность модели:

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(5).fit_transform(X)
```
В данном случае используем полиномиальные признаки пятой степени. Вы можете поэкспериментировать с другими степенями полинома. Построим предсказание для анализа:
```
polynomial = LinearRegression()
polynomial.fit(poly, y)
y_pred_poly = polynomial.predict(poly)
```
По этим данным можно построить график, подобный предыдущему:
```
plt.scatter(y_pred_poly, y)
plt.plot(y, y, c='r')
```


По этому графику можно сделать вывод, что модель стала несколько более точной. Но более конкретно это улучшение можно увидеть при помощи выбранной метрики:
```
polynomial.score(poly, y)
0.7406575543454206
```
Метрика показывает, что вторая модель примерно на 14 процентных пунктов лучше, чем первая.

Контрольные вопросы

1. Чем отличается применение разных моделей регрессии в бибилиотеке *sklearn* от моделей классификации?
2. Что показывает коэффициент детерминации для модели регрессии?
3. Какое значение имеют коэффициенты линейной регрессии?
4. Какие атрибуты имеет объект линейной регрессии?

Задания для самостоятельного выполнения

1. Какую еще информацию можно вывести для обученной модели? Попробуйте изменить аргументы при создании модели и посмотрите, как это влияет на качество предсказания.
2. Попробуйте применить к той же задаче другие модели регрессии. Для каждой из них выведите визуализацию регрессии и оценку точности. Рекомендуется исследовать следующие модели:
    i. Метод опорных векторов
        a. Без ядра

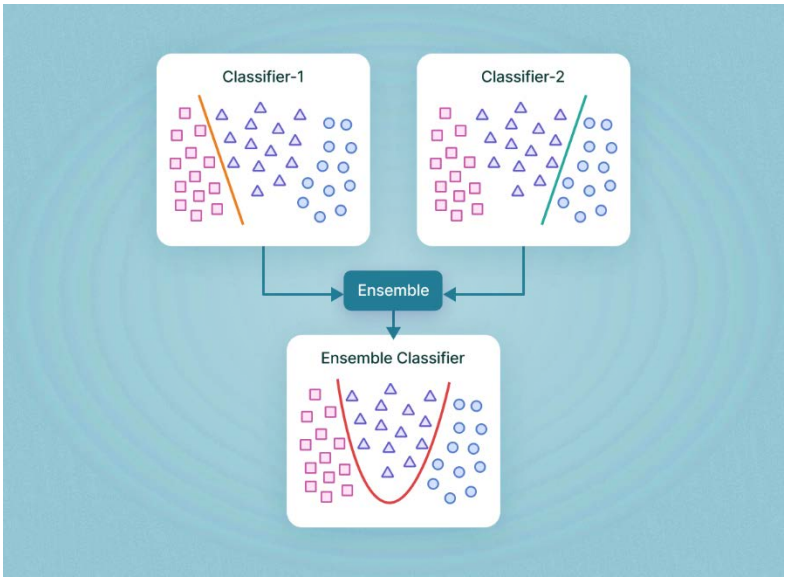
        b. С гауссовым ядром
        c. С полиномиальным ядром
    ii. Метод ближайших соседей
    iii. Многослойный перцептрон
    iv. Дерево решений
    v. (*) Другие методы:
        a. Гребневую регрессию
        b. Регрессию Лассо
        c. Регрессию ElasticNet
        d. Случайный лес
        e. Беггинг
        f. Другие модели по желанию
3. Напишите функцию, которая автоматически обучает все перечисленные модели и для каждой выдает оценку точности.
4. Повторите полностью анализ для другого набора данных - встроенного в *sklearn* датасета *diabetes*.

Лабораторная работа7

Тема: Ансамблевые методы

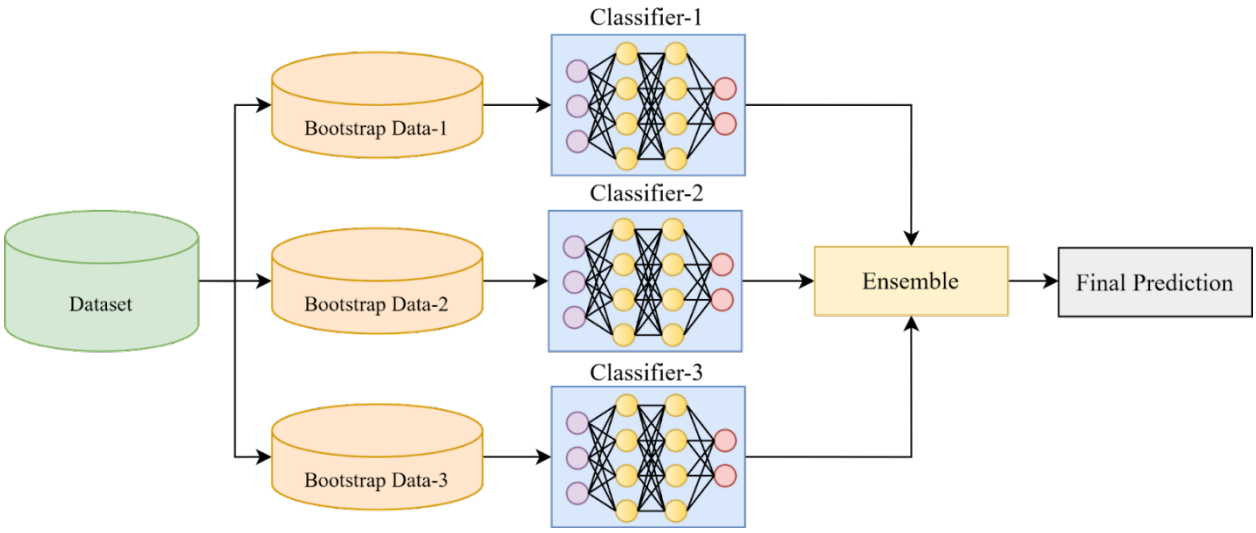
Цель работы: закрепление знаний, умений и навыков по работе с ансамблевыми методами

Методические указания



1. Ансамблевое обучение - техника обучения нескольких моделей на одной и той же задаче с целью повышения предсказательной эффективности.
2. Ансамбль - набор нескольких моделей машинного обучения и способ их комбинирования и обучения.
3. Ансамблирование - способ повысить эффективность за счет повышения вычислительной сложности.
4. Используется как в обучении с учителем, так и без него.
5. Ансамбль выступает как единая модель с входом и выходом. Техника обучения аналогичная.
6. Ансамбли также могут переобучаться и недообучаться.
7. Работает закон убывания отдачи в построении ансамбля.

Задание 1. Беггинг



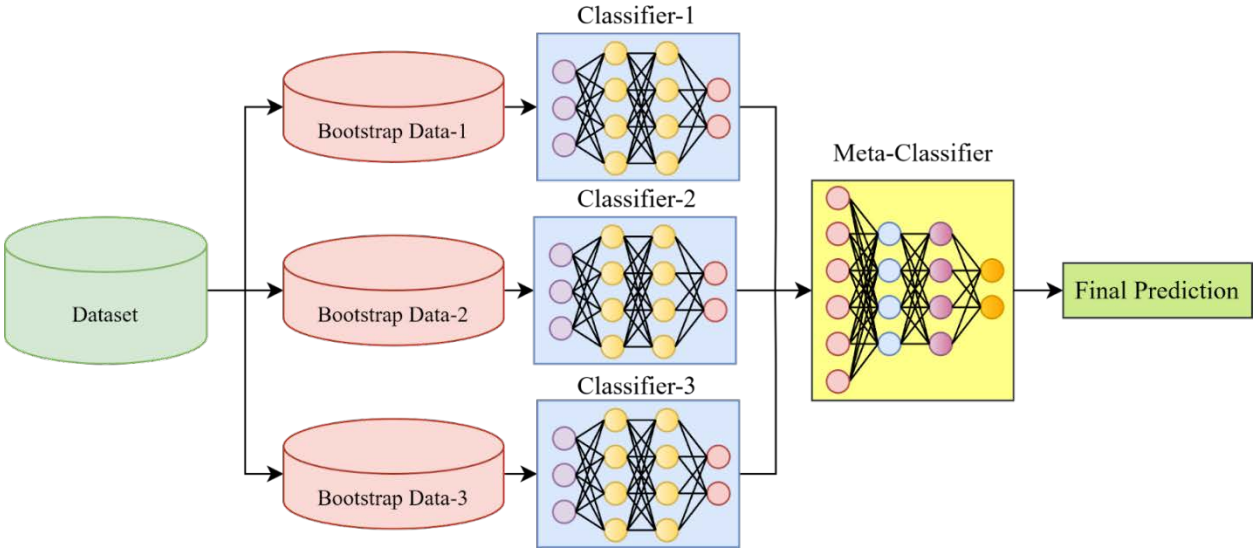

```
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(n_estimators=50, random_state=1)
clf3 = GaussianNB()

eclf = VotingClassifier(
    estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)],
voting='hard')
```

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()
bagging_clf = BaggingClassifier(base_estimator=tree,
n_estimators=1500,
random_state=42)
bagging_clf.fit(X_train, y_train)
```

Выводы:

1. Беггинг (Bagging, bootstrapaggregating) - метод обучения нескольких моделей на подвыборках обучающей выборки.
2. Обучающая выборка разделяется на подмножества с повторениями. На каждом подмножестве обучается своя модель.
3. Комбинирование предикторов осуществляется усреднением либо голосованием.
4. Беггинг без ресемплинга- это простой голосующий ансамбль.
5. Голосование может быть жестким или мягким.
6. Существенно уменьшает вариативность моделей. ТАким образом борется с пере-обучением (ошибки усредняются).
7. Итоговая модель менее чувствительна к аномалиям и ошибкам в данных.
8. Обучение предикторов можно распараллелить.
9. Минусы - потеря интерпретируемости, вычислительная сложность.

Задание 2. Стекинг

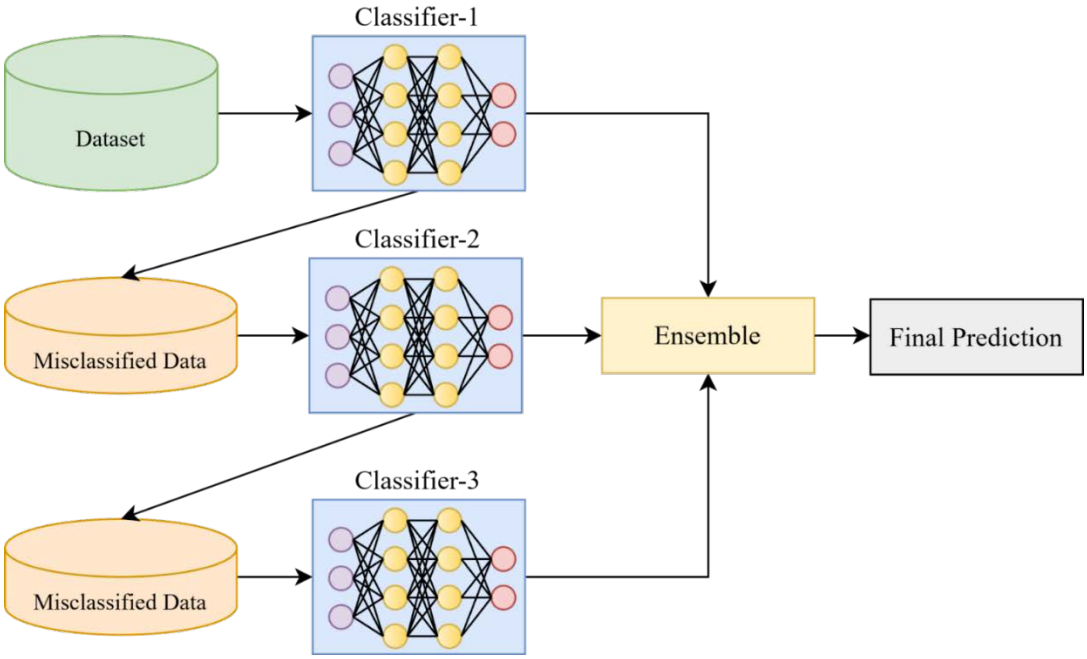



```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import StackingRegressor
final_estimator = GradientBoostingRegressor(
n_estimators=25, subsample=0.5,
min_samples_leaf=25, max_features=1,
random_state=42)
reg = StackingRegressor(
    estimators=estimators,
final_estimator=final_estimator)
```

Выводы:

1. Стакинг (стекинг, stacking, Stackedgeneralization) - аналогично беггингу, но с добавлением метапредиктора (модель второго порядка).
2. Метапредиктор использует выходы ансамблевых предикторов как входные данные.
3. При стекинге используется разбиение обучающего набора на K фолдов, как при кросс-валидации.
4. Из-за комбинации преимуществ разных моделей способен существенно повысить точность.
5. Обучается дольше из-за ограниченности параллелизации.
6. Склонен к переобучению из-за наличия модели второго порядка.

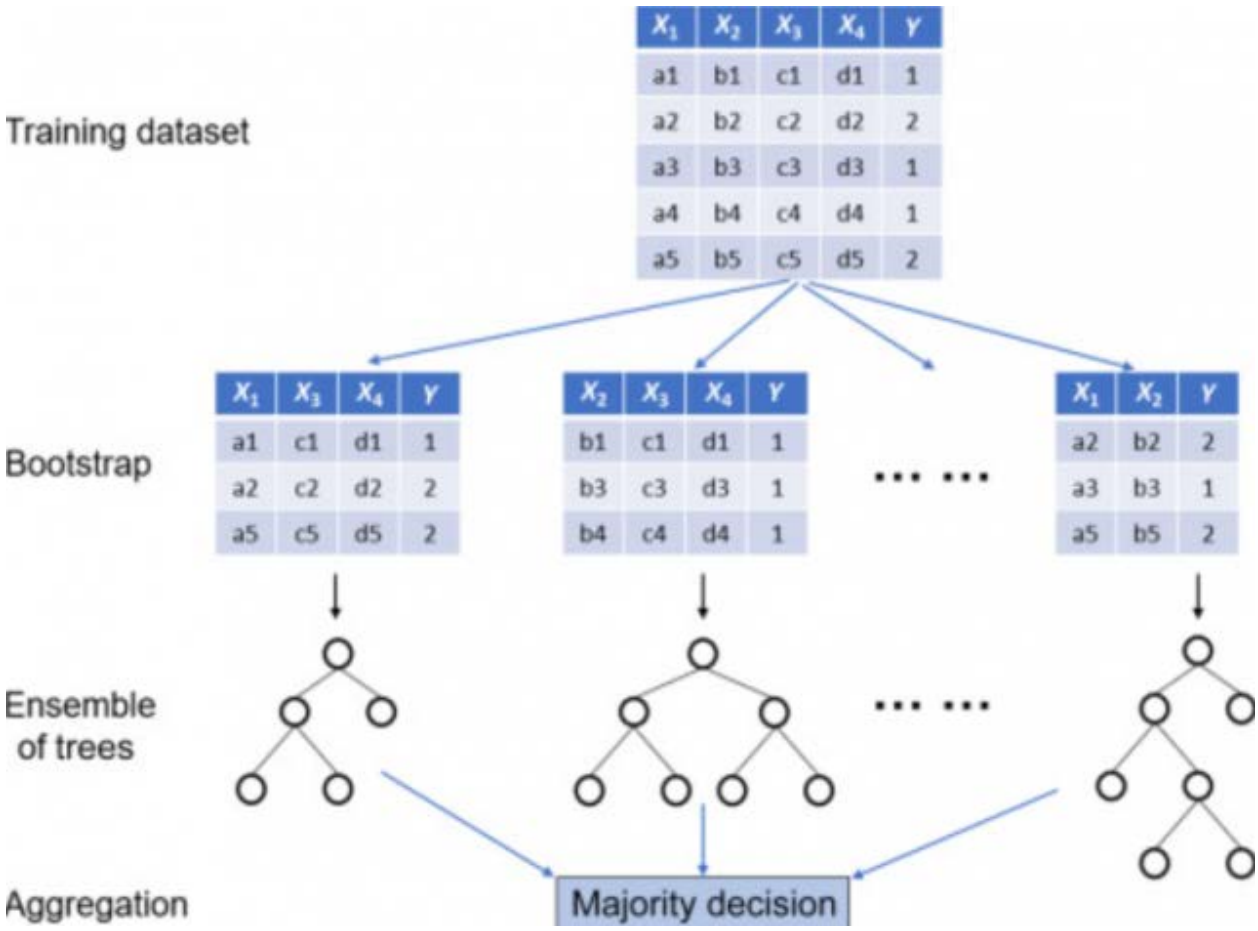
Задание 3. Бустинг



Выводы:

1. Идея бустинга - в последовательном анализе данных моделями из ансамбля.
2. Первая модель ансамбля обучается на всей выборке.
3. Следующая модель получает лишь те данные, на которых предыдущая модель ошиблась.
4. Последующие модели могут исправлять ошибки предыдущих.
5. Моделям присваиваются веса в зависимости от их эффективности.
6. Может понижать смещение моделей.
7. В качестве предикторов обычно берутся простые модели с высоким смещением и низкой вариацией.
8. Из-за последовательной обработки данных, вычислительно сложен.
9. Чувствителен к выбросам и аномалиям.

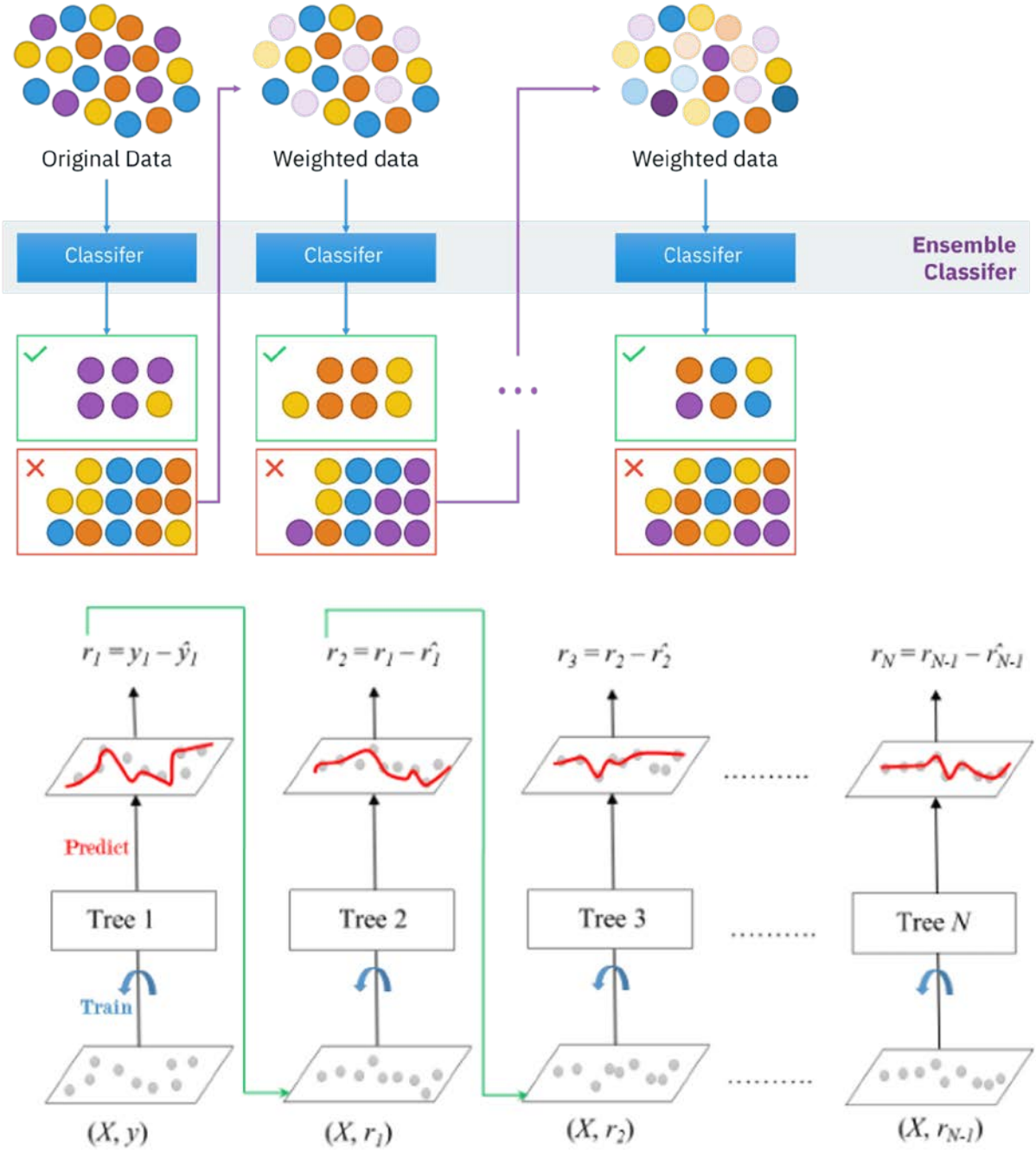
Задание 4. Случайный лес



Выводы:

1. Случайные лес – это беггинг над набором деревьев решений.
2. Деревья решения имеют высокую вариацию, которую может снизить беггинг.
3. В случайном лесе обычно используется семплирование как по строкам, так и по столбцам.
4. Используется как для классификации, так и для регрессии.
5. Менее подвержены переобучению.
6. Чем больше количество деревьев, тем больше регуляризационный эффект.
7. Можно настраивать максимальное количество признаков для индивидуальных деревьев.

Задание 5. Градиентный бустинг



Выводы:

1. Адаптивный бустинг (AdaBoost) присваивает веса точкам обучающей выборки в зависимости от того, правильно ли они были распознаны первыми классификаторами.
2. Последующие классификаторы фокусируются на “сложных” случаях пропорционально весам.
3. Адаптивный бустинг зачастую применяется для задач бинарной классификации.
4. Граиентный бустинг (GBM) передает в последующие модели величину отклонения предыдущих моделей.
5. Градиентный бустинг использует деревья решений в качестве индивидуальных предикторов.
6. XGBoost- это параллелизуемая высокоэффективная реализация градиентного бу-

стинга.

7. XGBoost может обрабатываеть большие объемы данных.
8. XGBoost склонен к переобучению, но использует регуляризацию.
9. XGBoost достаточно требователен к объему оперативной памяти.
10. Другие ивестные реалиации - LightGBM, CatBoost

Лабораторная работа 8

Тема: Стохастический поиск

Цель работы: закрепление знаний, умений и навыков по работе со стохастическим градиентным спуском

Методические указания

Стохастический градиентный спуск (StochasticGradientDescent - SGD) - это простой, но очень эффективный подход для обучения линейных классификаторов и регрессоров под выпуклые функции потерь, такие как (линейные) Метод опорных векторов и Логистическая регрессия. Несмотря на то, что SGD существует в сообществе машинного обучения уже давно, значительное внимание к нему в контексте крупномасштабного обучения было привлечено совсем недавно.

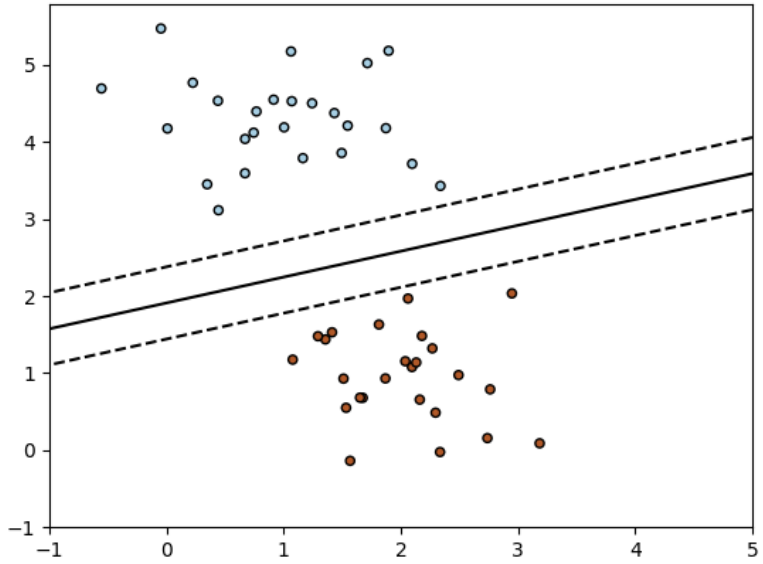
SGD успешно применяется для решения крупномасштабных и разреженных задач машинного обучения, часто встречающихся в классификации текстов и обработке естественного языка. Учитывая разреженность данных, классификаторы в этом модуле легко масштабируются на задачи с более чем 10^5 обучающими примерами и более чем 10^5 признаками.

Строго говоря, SGD - это всего лишь техника оптимизации и не соответствует определенному семейству моделей машинного обучения. Это лишь способ обучения модели. Часто экземпляр SGDClassifier или SGDRegressor будет иметь эквивалентную модель в scikit-learn API, потенциально использующий другую технику оптимизации. Например, использование SGDClassifier(loss='log_loss') приводит к логистической регрессии, то есть к модели, эквивалентной LogisticRegression, которая подгоняется с помощью SGD вместо того, чтобы подгоняться с помощью одного из других алгоритма в LogisticRegression. Аналогично, SGDRegressor(loss='squared_error', penalty='l2') и Ridge решают одну и ту же задачу оптимизации разными средствами.

Преимуществами стохастического градиентного спуска являются:
- Эффективность.
- Простота реализации (много возможностей для настройки кода).

К недостаткам стохастического градиентного спуска относятся:
- Для SGD требуется ряд гиперпараметров, таких как параметр регуляризации и количество итераций.
- SGD чувствителен к масштабированию признаков.

Задание 1. Стохастический градиентный спуск для задачи классификации
Класс **SGDClassifier** реализует обычную стохастическую процедуру обучения методом градиентного спуска, которая поддерживает различные функции потерь и штрафы для классификации. Ниже показана граница принятия решения классификатора **SGDClassifier**, обученного с hingeloss, что эквивалентно линейному SVM.



Как и другие классификаторы, SGD должен быть оснащен двумя массивами: массивом X формы (n_samples, n_features), содержащим обучающие выборки, и массивом y формы (n_samples,), содержащим целевые значения (метки классов) для обучающих выборок:

>>> **fromsklearn.linear_modelimport**SGDClassifier

>>>X = [[0., 0.], [1., 1.]]

>>>y = [0, 1]

>>>clf = SGDClassifier(loss="hinge", penalty="l2", max_iter=5)

>>>clf.fit(X, y)

SGDClassifier(max_iter=5)

После обучения модель может быть использована для предсказания новых значений:

>>>clf.predict([[2., 2.]])

array([1])

SGD подгоняет линейную модель к обучающим данным. Атрибут coef_ содержит параметры модели:

>>>clf.coef_

array([[9.9..., 9.9...]])

Атрибут intercept_ содержит перехват (он же смещение или отклонение):

>>>clf.intercept_

array([-9.9...])

Использует ли модель перехват, т.е. смещенную гиперплоскость, контролируется параметром fit_intercept.

Подписанное расстояние до гиперплоскости (вычисляемое как произведение точек между коэффициентами и входной выборкой плюс перехват) задается **SGDClassifier.decision_function**:

>>>clf.decision_function([[2., 2.]])

array([29.6...])

Конкретная функция потерь может быть задана с помощью параметра loss. SGDClassifier` поддерживает следующие функции потерь:

- loss="hinge": (soft-margin) линейный метод опорных векторов,
- loss="modified_huber": сглаженные hinge loss,
- loss="log_loss": логистическая регрессия,
- и все регрессионные потери ниже. В этом случае цель кодируется как -1 или 1, а задача рассматривается как задача регрессии. Предсказанный класс соответствует знаку предска-

занной цели.

Первые две функции потерь являются ленивыми, они обновляют параметры модели только в том случае, если пример нарушает ограничение на маржу, что делает обучение очень эффективным и может привести к разреженным моделям (т.е. с большим количеством нулевых коэффициентов), даже если используется штраф L2.

Использование loss="log_loss" или loss="modified_huber" включает метод predict_proba, который дает вектор вероятностных оценок P(y|x)‘ на выборку x:

>>>clf = SGDClassifier(loss="log_loss", max_iter=5).fit(X, y)

>>>clf.predict_proba([[1., 1.]])

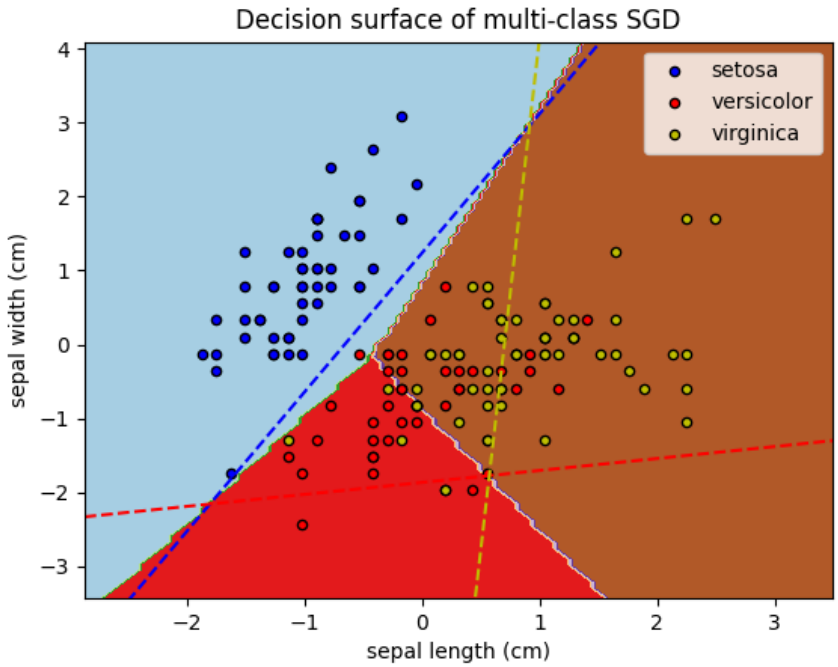
array([[0.00..., 0.99...]])

Конкретный штраф может быть задан с помощью параметра penalty. SGD поддерживает следующие штрафы:

- penalty="l2": L2 норма штрафа на coef_.
- penalty="l1": L1 норма штрафа на coef_.
- penalty="elasticnet": выпуклая комбинация штрафов L1 и L2; (1 - l1_ratio) * L2 + l1_ratio * L1.

По умолчанию задано penalty="l2". Штраф L1 приводит к разреженным решениям, обращая большинство коэффициентов в ноль. Эластичная сеть решает некоторые недостатки штрафа L1 при наличии сильно коррелированных признаков. Параметр l1_ratio управляет выпуклой комбинацией штрафов L1 и L2.

SGDClassifier поддерживает многоклассовую классификацию, объединяя несколько бинарных классификаторов по схеме “один против всех” (OVA). Для каждого из классов K обучается бинарный классификатор, который различает этот и все остальные классы K−1. Во время тестирования мы вычисляем показатель доверия (т.е. подписанные расстояния до гиперплоскости) для каждого классификатора и выбираем класс с наибольшим доверием. На рисунке ниже показан подход OVA на наборе данных Ирис. Пунктирные линии представляют три классификатора OVA; цвета фона показывают поверхность принятия решений, вызванную тремя классификаторами.



В случае многоклассовой классификации coef_ - это двумерный массив форм (n_classes, n_features), а intercept_ - одномерный массив форм (n_classes,). i-я строка coef_ содержит вектор веса OVA-классификатора для i-го класса; классы индексируются в порядке возрастания (см. ат-

рибут classes_).
Обратите внимание, что в принципе, поскольку они позволяют создать вероятностную модель, loss="log_loss" и loss="modified_huber" больше подходят для классификации “один против всех”.
SGDClassifier поддерживает взвешенные классы и взвешенные экземпляры через параметры обучения class_weight и sample_weight. Дополнительную информацию см. в примерах ниже и в строке документации **SGDClassifier.fit.**
SGDClassifier поддерживает усредненный SGD (ASGD). Усреднение можно включить, установив average=True. ASGD выполняет те же обновления, что и обычный SGD, но вместо использования последнего значения коэффициентов в качестве атрибута coef_ (т.е. значения последнего обновления), вместо coef_ устанавливается **среднее** значение коэффициентов по всем обновлениям. То же самое делается для атрибута intercept_. При использовании ASGD скорость обучения может быть больше и даже постоянной, что на некоторых наборах данных приводит к ускорению времени обучения.
Для классификации с логистическими потерями доступен другой вариант SGD со стратегией усреднения - алгоритм StochasticAverageGradient (SAG), доступный в качестве алгоритма в **LogisticRegression**.

Задание 2. Стохастический градиентный спуск для задачи регрессии
Класс **SGDRegressor** реализует простую стохастическую процедуру обучения градиентного спуска, которая поддерживает различные функции потерь и штрафы для обучения моделей линейной регрессии. **SGDRegressor** хорошо подходит для задач регрессии с большим количеством обучающих выборок (> 10.000), для других задач мы рекомендуем **Ridge, Lasso**, или **ElasticNet.**
Конкретная функция потерь может быть задана через параметр loss. SGDRegressor` поддерживает следующие функции потерь:

- loss="squared_error": Обычные наименьшие квадраты,
- loss="huber": hingeloss для робастной регрессии,
- loss="epsilon_insensitive": линейная регрессия опорных вектора.

Для робастной регрессии можно использовать функции hingeloss и epsilon_insensitive. Ширина области нечувствительности задается с помощью параметра epsilon. Этот параметр зависит от масштаба целевых переменных.
Параметр penalty определяет используемую регуляризацию (см. описание выше в разделе классификации).
SGDRegressor также поддерживает усредненный SGD (здесь опять же см. описание выше в разделе классификации).
Для регрессии с квадратичными потерями и штрафом l2 доступен другой вариант SGD со стратегией усреднения - алгоритм StochasticAverageGradient (SAG), доступный в качестве алгоритма в **Ridge.**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по организации и проведению самостоятельной работы
по дисциплине
«МАШИННОЕ ОБУЧЕНИЕ»
для студентов направления подготовки
02.03.01 Математика и компьютерные науки
направленность (профиль)
«Анализ данных и искусственный интеллект»

Общие положения

Самостоятельная работа - планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов (СРС) в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

К основным видам самостоятельной работы студентов относятся:

– формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);

– написание докладов;

– подготовка к семинарам, практическим и лабораторным работам, их оформление;

– составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);

– выполнение учебно-исследовательских работ, проектная деятельность;

– подготовка практических разработок и рекомендаций по решению проблемной ситуации;

– выполнение домашних заданий в виде решения отдельных задач, проведения типовых расчетов, расчетно-компьютерных и индивидуальных работ по отдельным разделам содержания дисциплин и т.д.;

– компьютерный текущий самоконтроль и контроль успеваемости на базе электронных обучающих и аттестующих тестов;

– выполнение курсовых работ (проектов) в рамках дисциплин;

– выполнение выпускной квалификационной работы и др.

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Самостоятельная работа по дисциплине **«Машинное обучение»** направлена на формирование следующих **компетенций**:

ПК-1. Способен демонстрировать базовые знания математических и естественных наук, основ программирования и информационных технологий.

1. Цель и задачи самостоятельной работы

Ведущая цель организации и осуществления СРС совпадает с целью обучения студента – формирование соответствующего набора компетенций будущего бакалавра по направлению подготовки «Математика и компьютерные науки».

При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

2. Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (акад.)		
			СРС	Контактная работа с преподавателем	Всего
8 семестр					
ПК-1	Самостоятельное изучение литературы	Собеседование	8	2	10
ПК-1	Подготовка к лабораторным занятиям	Защита лабораторной работы	10	2	12
ПК-1	Подготовка к экзамену	Вопросы к экзамену	52	2	54
		Итого за 8 семестр	**70**	**6**	**76**
		Итого	**70**	**6**	**76**

3. Порядок выполнения самостоятельной работы студентом

3.1. Методические рекомендации по работе с учебной литературой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют ***четыре основные установки в чтении научного текста***:

информационно-поисковый (задача – найти, выделить искомую информацию)

усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения излагаемые автором, так и всю логику его рассуждений)

аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)

творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

Основные виды систематизированной записи прочитанного:

Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;
2. Выделите главное, составьте план;
3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;
4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.
5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

3.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям

Для того чтобы практические и лабораторные занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на практических занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в стро-

гом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

3.3. Методические рекомендации по самопроверке знаний

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на практических занятиях и самостоятельно студенту рекомендуется, провести самопроверку усвоенных знаний, ответив на контрольные вопросы по изученной теме.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует помнить, что правильное решение задачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

3.4. Методические рекомендации по подготовке к экзаменам и зачетам

Изучение многих общепрофессиональных и специальных дисциплин завершается экзаменом. Подготовка к экзамену способствует закреплению, углублению и обобщению знаний, получаемых, в процессе обучения, а также применению их к решению практических задач. Готовясь к экзамену, студент ликвидирует имеющиеся пробелы в знаниях, углубляет, систематизирует и упорядочивает свои знания. На экзамене студент демонстрирует то, что он приобрел в процессе обучения по конкретной учебной дисциплине.

Экзаменационная сессия - это серия экзаменов, установленных учебным планом. Между экзаменами интервал 3-4 дня. Не следует думать, что 3-4 дня достаточно для успешной подготовки к экзаменам.

В эти 3-4 дня нужно систематизировать уже имеющиеся знания. На консультации перед экзаменом студентов познакомят с основными требованиями, ответят на возникшие у них вопросы. Поэтому посещение консультаций обязательно.

Требования к организации подготовки к экзаменам те же, что и при занятиях в течение семестра, но соблюдаться они должны более строго. Во-первых, очень важно соблюдение режима дня; сон не менее 8 часов в сутки, занятия заканчиваются не позднее, чем за 2-3 часа до сна. Оптимальное время занятий - утренние и дневные часы. В перерывах между занятиями рекомендуются прогулки на свежем воздухе, неутомительные занятия спортом. Во-вторых, наличие хороших собственных конспектов лекций. Даже в том случае, если была пропущена какая-либо лекция, необходимо во время ее восстановить (переписать ее на кафедре), обдумать, снять возникшие вопросы для того, чтобы запоминание материала было осознанным. В-третьих, при подготовке к экзаменам у студента должен быть хороший учебник или конспект литературы, прочитанной по указанию преподавателя в течение семестра. Здесь можно эффективно использовать листы опорных сигналов.

Вначале следует просмотреть весь материал по сдаваемой дисциплине, отметить для себя трудные вопросы. Обязательно в них разобраться. В заключение еще раз целесообразно повторить основные положения, используя при этом листы опорных сигналов.

Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

Вопросы к экзамену
по дисциплине **«Машинное обучение»**

1. Типы задач машинного обучения
2. Предмет и задачи машинного обучения и анализа данных.
3. Основные принципы, задачи и подходы, использование в различных областях науки и индустрии.
4. Основные этапы эволюции алгоритмов машинного обучения.
5. Метрические классификаторы
6. Общий вид метрического классификатора.
7. Алгоритм K ближайших соседей.
8. Алгоритмы отбора эталонов.
9. Алгоритмы кластеризации
10. Алгоритмы кластеризации с фиксированным количеством кластеров.
11. Алгоритмы кластеризации по плотности.
12. Иерархическая кластеризация.
13. Деревья решений
14. Правила и анализ качества (точность, полнота). Анализ с помощью ROC кривой. Алгоритм построения деревьев решений. Критерий информационного выигрыша и критерий Джини. Леса решающих деревьев.
15. Линейные классификаторы
16. Перцептрон и разделяющая гиперплоскость. Переход в пространство повышенной размерности. Метод опорных векторов
17. Нейронные сети и глубокое обучение
18. Логистическая регрессия.
19. Градиентный спуск.
20. Нейронные сети и алгоритм обратного распространения градиента.
21. Глубокое обучение, свертки и пулинг.
22. Регрессионный анализ
23. Линейная регрессия.
24. Полиномиальная регрессия.
25. Смещение и дисперсия.
26. Гребневая регрессия.
27. Ансамблевые методы
28. Голосование.
29. Бутстраппинг.
30. Бустинг, адаптивный бустинг, градиентный бустинг.
31. Стохастический поиск
32. Монте-Карло поиск.
33. Алгоритм симулированного отжига.
34. Генетический алгоритм

Контроль самостоятельной работы студентов

Контроль самостоятельной работы проводится преподавателем в аудитории.

Предусмотрены следующие виды контроля: собеседование, оценка выполнения лабораторной работы.

Подробные критерии оценивания компетенций приведены в Фонде оценочных средств для проведения текущей и промежуточной аттестации.

Список литературы для выполнения СРС

Перечень основной литературы:

1. Татарникова Т. М. Интеллектуальный анализ данных: учебное пособие / Т. М. Татарникова. - Москва, Вологда: Инфра-Инженерия, 2024. - 172 с. - ISBN 978-5-9729-1772-3. - Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. - URL: https://www.iprbookshop.ru/143351.html

2. Степашкина А. С. Численные методы и машинное обучение в метрологии: учебное пособие / А. С. Степашкина. - Москва, Вологда: Инфра-Инженерия, 2024. - 144 с. - ISBN 978-5-9729-1954-3. - Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. - URL: https://www.iprbookshop.ru/143592.html

Перечень дополнительной литературы:

1. Васильев Е. П. Интеллектуальный анализ данных в технологиях принятия решений: учебное пособие / Е. П. Васильев, В. И. Орешков. - Рязань: Рязанский государственный радиотехнический университет, 2023. - 180 с. - ISBN 978-5-7722-0344-6. - Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. - URL: https://www.iprbookshop.ru/134854.html

2. Замятин А. В. Интеллектуальный анализ данных: учебное пособие / А. В. Замятин. - Томск: Издательский Дом Томского государственного университета, 2020. - 194 с. - ISBN 978-5-94621-898-6. - Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. - URL: https://www.iprbookshop.ru/116889.html

3. Пальмов С. В. Интеллектуальный анализ данных: учебное пособие / С. В. Пальмов. - Самара: Поволжский государственный университет телекоммуникаций и информатики, 2017. - 127 с. - Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. - URL: https://www.iprbookshop.ru/75376.html

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

- https://www.kaggle.com/ - онлайн платформа для проектов в области науки о данных
- UCI Machine Learning Repository — репозиторий наборов данных для машинного обучения - http://archive.ics.uci.edu/ml/
- Открытый курс машинного обучения https://habr.com/company/ods/blog/322626/
- Ресурс, посвященный машинному обучению, распознаванию образов и интеллектуальному анализу данных. - http://machinelearning.ru