

Лабораторный практикум предназначен для студентов факультета радиофизики и компьютерных технологий специальности «Прикладная информатика» с целью отработки практических навыков и более глубокого освоения материала по курсу «Организация обработки данных в сложных системах». В лабораторном практикуме представлены 6 лабораторных работ, относящихся к различным разделам машинного обучения и теории интеллектуальных систем.

Лабораторная работа 1. Задача классификации

Установка Python.

Язык Python и написанные для него библиотеки является на данный момент одним из самых популярных средств, используемых в машинном обучении. Python является динамически типизированным интерпретируемым языком и для работы с ним необходимо установить соответствующий интерпретатор. Для того, чтобы проверить, установлен ли на вашем компьютере интерпретатор Python, необходимо в командной строке ввести:

```
C:\Users\User>python --version
```

Если Python уже установлен, то в ответ вы получите номер версии, например, Python 3.8.7. В процессе развития языка Python появлялось достаточно много его версий. Основное внимание необходимо уделить выбору между версиями 2.x и 3.x, так они не являются обратно совместимыми. Данный лабораторный практикум рассчитан на использование Python 3.8.7. Хотя на сегодняшний день распространяется более актуальная версия, некоторые библиотеки, используемые в последующих лабораторных работах, пока несовместимы с более поздними версиями. Версии необходимых библиотек будут указываться по мере их упоминания далее. Установщик Python можно скачать с сайта: python.org.

Для работы необходима также среда разработки. В качестве среды можно использовать текстовые редакторы, сохраняя код в файлах с расширением .py. Желательно использовать редакторы со встроенными функциями подсветки синтаксиса и прочими дополнительными возможностями, упрощающими задачу разработчика, такие как, PyCharm, Visual Studio Code и т.д. Также для разработки можно использовать специальные блокноты, в частности, в данном лабораторном практикуме предлагается использовать блокнот JupyterLab.

Настройка JupyterLab.

Блокнот JupyterLab – браузерный графический интерфейс с возможностями динамической визуализации. Помимо выполнения операторов языка Python, блокнот позволяет вставлять форматированный текст, уравнения, выводить графическую информацию, виджеты JavaScript и многое другое. Кроме этого, существует возможность сохранения файла с выведенными элементами, что является удобным инструментом для исследования работы алгоритмов.

Если на компьютере не установлен JupyterLab (проверить это можно командой `jupyter lab --version`), установите JupyterLab с помощью команды (ввести в командную строку):

```
pip install jupyterlab
```

Запустите JupyterLab с помощью команды:

```
jupyter lab
```

При запуске JupyterLab происходит запуск сервера на порте, установленном по умолчанию (обычно 8888 - <http://localhost:8888/lab>). Соответствующая страница открывается в браузере, как отдельное веб-приложение, что делает использование Jupyter практически независимым от платформы и открывает возможности для более удобного обмена в Интернете.

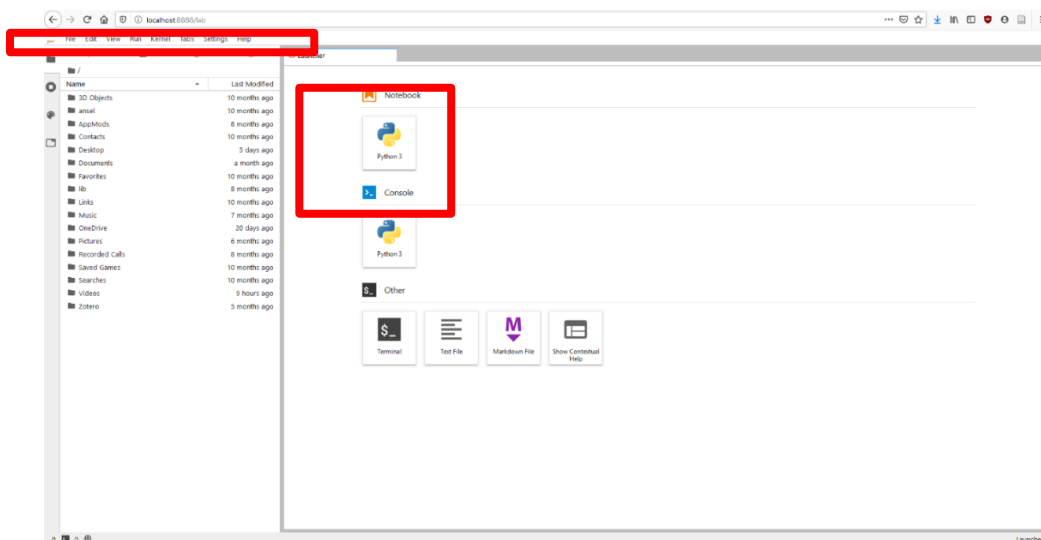



Рисунок 1.1 – Начальная страница JupyterLab

Не закрывайте командную строку, пока работаете с Блокнотом в браузере.

Создайте новый Блокнот (Notebook), выбрав соответствующий интерпретатор из предложенного списка (Python 3), см. рис. 1.1. Созданный Блокнот откроется в новой вкладке под названием Untitled.ipynb. В верхней части страницы расположена панель команд, а под названием вкладки расположена панель инструментов. На панели инструментов расположены элементы для работы с отдельными ячейками (cells) страницы, также к ним можно получить доступ из панели команд по команде Edit. Ячейки страницы являются ее составными элементами и задают общую структуру документа. Введите в первую ячейку код, выводящий название лабораторной работы, и запустите его (либо с панели инструментов () , либо с панели команд (Run->Run selected cells)):

```
print('Лабораторная работа 1. Задача классификации')
```

Импортирование библиотек.

Одно из преимуществ использования языка Python – наличие большого количества библиотек с открытым исходным кодом, которые можно добавлять в свой проект с уже готовыми реализованными алгоритмами, что значительно ускоряет и упрощает разработку приложений и анализ данных. Для использования сторонних библиотек в своей программе необходимо их проимпортировать с помощью ключевого слова `import`. Библиотеки можно импортировать как целиком, так и частично:

```
# знаком «решетка» обозначается однострочный комментарий
# введенный после него текст (код) выполняться не будет
import math                # импорт библиотеки math
import math as mt          # импорт и переименование
from math import log, pi   # импорт отдельных функций
from math import *         # импорт всех переменных
```

Обращение к проимпортированным функциям и переменным осуществляется через точку после названия библиотеки (`math.pi`) или в случае переименования через созданную новую переменную (`mt.pi`). При импортировании отдельных или всех переменных вызов осуществляется напрямую по названию переменной (`pi`).

В таблице 1 приведены функции, полезные при написании и отладке программы:

Таблица 1. Функции отладки программы

Функция	Описание
<code>type(name)</code>	Возвращает тип объекта
<code>dir(name)</code>	Возвращает имена переменных, доступные в локальной области, либо атрибуты указанного объекта в алфавитном порядке.
<code>help(name)</code>	Если передана строка, то производится попытка интерпретировать её как имя модуля, функции, класса, метода, или раздела документации, после чего справка выводится в консоль. Если передан объект любого другого типа, страница справки генерируется по его данным.

Чтение и анализ данных (при помощи Pandas).

Язык Python поддерживает различные типы и структуры данных. Для работы с множественными данными используются динамические массивы, представленные типом `list` и поддерживающие синтаксис с использованием квадратных скобок «`[]`». Примеры массивов разного уровня вложенности, различных типов данных, в том числе, смешанных массивов:

```
primes = [2, 3, 5]
planets = ['Mercury', 'Venus', 'Earth', 'Mars']
hands = [['J', 'Q', 'K'], ['2', '2', '2'], ['6', 'A', 'K']]
favourite_things = [32, 'raindrops on roses', help]
```

Примеры получения доступа к элементам и соответствующий вывод:

```
planets[0]      #->      'Mercury'
planets[0:3]    #->      'Mercury', 'Venus', 'Earth'
```

Для работы с объектами типа `list` также можно использовать встроенные функции, список которых можно получить с помощью команды `dir(list)`.

Для работы с датасетами в табличном виде наиболее распространена библиотека `pandas`. Она может быть установлена при помощи консольной команды:

```
pip install pandas
```

При использовании готового датасета данные чаще всего хранятся в файлах с разметкой, например, `.csv`. Чтение данных из файла `.csv` осуществляется функцией `read_csv` из библиотеки `pandas`:

```
import pandas as pd
data = pd.read_csv(file_path)
```

Переменная `file_path` является строкой, следовательно, значение передается в кавычках (одинарных или двойных). Для того, чтобы символ «\» не распознавался как начало escape-последовательности, используется последовательность «\\».

Функция `read_csv` возвращает фрейм – объект типа `pandas.DataFrame`, который инкапсулирует загруженные данные и позволяет работать с ними с помощью различных методов.

Пример значения `file_path`:

```
'D:\\pe-files-malwares\\dataset_malwares.csv'.
```

Для описания простейших статистических характеристик фрейма используется метод `describe` (вызов: `data.describe()`), выводящий к каждой колонке датасета 8 значений (таблица 2).

Таблица 2. Описание значений в выдаче функции `describe`

Характеристика	Описание
count	Количество заполненных значений
mean	Матожидание
std	Среднеквадратическое отклонение
min	Минимальное значение
25%	Значение – большее 25% и меньшее – 75% данных
50%	Значение – большее 50% и меньшее – 50% данных
75%	Значение – большее 75% и меньшее – 25% данных
max	Максимальное значение

Для обращения к отдельным колонкам датасета можно использовать точечную нотацию и индексатор («`ColumnX`» - название одной из колонок датасета):

```
y = data.ColumnX
y = data[['ColumnX', 'ColumnY', 'ColumnZ']]
```

У объекта `DataFrame` есть большое количество атрибутов, свойств и методов, которые можно использовать для более эффективной работы с данными. Подробнее ознакомиться с ними можно в документации Pandas (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html?highlight=da>

taframe#pandas.DataFrame). Например, свойство `columns` возвращает выведет названия всех присутствующих в загруженном фрейме колонок:

```
data.columns    #вывод названий всех присутствующих колонок
в датасете data
data1=data.drop(columns['ColumnX',          'ColumnY',
'ColumnZ'])    #скопировать data в переменную data1 без
перечисленных по названиям колонок
```

Для работы с данными в формате времени и даты используются специализированные типы данных. Для работы с различными алгоритмами может быть удобно также дату сконвертировать в UNIX timestamp (целое число, количество секунд с 1 января 1970).

Одной из частых проблем работы с датасетами является наличие пропусков в данных. Для заполнения только пропущенных значений может использоваться метод фрейма `fillna`. В простейшем случае, пропущенные данные могут быть заменены на некоторые константные значения, либо на среднее арифметическое (метод `mean`):

```
data=data.fillna(-1)
data=data.fillna(data.mean())
```

Для более корректного решения данной проблемы существует различные приемы, называемые методами условных оценок (`impute`). Данные методы автоматически заполняют пропуски на основе статистических распределений в данных (например, в `scikit-learn` для этого используется модуль `sklearn.impute` <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.impute>).

Графическое отображение. Построение графиков и корреляционных карт.

Достаточно часто интерес представляет наглядное представление информации и наиболее популярным графическим отображением данных являются графики, гистограммы, диаграммы и т.д. В данном разделе приводятся примеры построения основных видов графиков и диаграмм с помощью модуля `matplotlib.pyplot`. Для установки библиотеки используется команда

```
pip install matplotlib
```

Примеры отображений с использованием matplotlib.pyplot:

1. Обычный график. Вывод нескольких зависимостей количества пользователей по годам:

```
from matplotlib import pyplot as plt
years = [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019]
users1 = [30, 156, 212, 190, 450, 1149, 1300, 1298, 1783]
users2 = [87, 260, 429, 1190, 500, 149, 131, 129, 178]
plt.plot(years, users1, color='red', marker='o', linestyle='solid')
plt.plot(years, users2, 'g-', label='data2')
plt.title('Количество пользователей по годам')
plt.ylabel('Количество пользователей')
plt.legend(loc=9) #отобразить легенду вверху посередине
plt.show()
```



Рисунок 1.2 – Линейная диаграмма

2. Столбчатая диаграмма. Вывод количества вакансий по различным технологиям на сайте dev.by (bar – для горизонтальной диаграммы, barh – для вертикальной):

```
technology = ['Front-end/JavaScript', 'Java', 'PHP', 'Python',
'.NET/C#', 'iOS', 'Android', 'node.js', 'C/C++', 'Ruby/Rails', 'Golang',
'DataScience', 'Salesforce']
vacancies = [533, 257, 222, 206, 174, 166, 156, 132, 126, 43, 38,
34, 21]
plt.barh(technology, vacancies)
plt.title('Количество вакансий на сайте dev.by')
plt.ylabel('Технологии')
plt.show()
```



Рисунок 1.3 – Столбчатая диаграмма

3. Точечные диаграммы или диаграммы рассеивания.

```
data1 = [7, 58, 99, 70, 139, 190]
data2 = [30, 156, 212, 190, 450, 1149]
plt.scatter (data1, data2)
plt.axis('equal') #сопоставимые оси
plt.show()
```

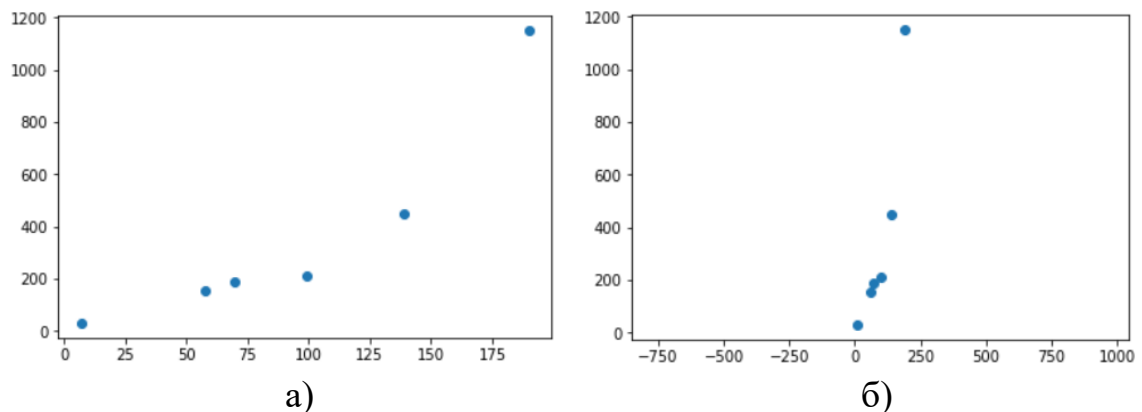


Рисунок 1.4 – Точечная диаграмма: а) несопоставимые по шкале оси; б) сопоставимые по шкалам оси

С помощью данных функций можно отобразить и сопоставить более сложные зависимости. Например, при анализе временных рядов часто полезным является построение гистограммы с отображенной поверх плотности вероятности.

Полезной функцией при анализе датасетов является построение корреляционных карт с целью выявить сильно коррелированные зависимости

между данными. Для работы с такими визуализациями используется библиотека seaborn (pip install seaborn):

```
import seaborn as sns
sns.heatmap(data.corr())
```

Для более углубленного знакомства с темой визуализации данных рекомендуется рассмотреть библиотеки seaborn, Bokeh и ggplot.

Случайные леса (randomforest).

Алгоритм случайных лесов основан на составлении ансамбля деревьев принятия решений. Пример простейшего дерева принятия решений для определения загаданного фрукта представлен на рисунке 1.5.



Рисунок 1.5 – Дерево принятия решений

То есть на каждом шаге алгоритма принимается одно из возможных решений, разделяющее все пространство возможных вариантов по одному из признаков. Например, если при определении фрукта (рисунок 1.5) одним из свойств объектов является «размер», то при выборе одного из вариантов («большой», «средний», «маленький»), все пространство возможных исходов разделяется на 3 части относительно признака «размер». Таким образом, постепенно отсекая ненужные варианты относительно различных признаков, решение сводится к правильному результату.

Деревья принятия решений, как правило, интуитивно понятны, легко интерпретируемы, могут работать с численными (масса фрукта в граммах) и с категориальными признаками («большой/средний/маленький» размер). Основной проблемой является нахождение «оптимального» дерева принятия решений для набора обучающих данных. Для этого используются

специальные алгоритмы, основанные на анализе выборки данных: ID3, C4.5, CART и другие.

Деревья принятия решений имеют тенденцию к переобучению, то есть к приспособлению к обучающим данным. Решить данную проблему помогают алгоритмы, основанные на использовании целой группы или ансамбле деревьев, называемые случайными лесами. При этом деревья в ансамбле не являются идентичными, а для принятия итогового решения используется голосование.

Для построения различных деревьев принятия решения в рамках одного леса используются 2 подхода: случайное разделение обучающей выборки и случайная выборка из набора признаков.

Для запуска алгоритмов в рамках лабораторной работы предлагается использовать классы `RandomForestClassifier` и `DecisionTreeClassifier` модуля `sklearn.tree` библиотеки `scikit-learn` (`pip install scikit-learn`):

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier().fit(X_train, y_train) #X_train-
входные значения, y_train- результат -> на этих данных будет построена
модель случайного леса
y_actual = tree.predict(X_test) #получить результат на других
данных (например, на тестовой выборке)
```

Подробнее о деревьях принятия решений и случайных лесах можно почитать тут:

https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B9.

Примеры вызова функции `RandomForestClassifier`, а также визуальное отображение того, как изменяется пространство принятия решения при использовании отдельных деревьев и при использовании случайных лесов можно посмотреть тут:

<https://github.com/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.08-Random-Forests.ipynb>

Метод опорных векторов (support vector machine, SVM)

Метод опорных векторов основан на поиске гиперплоскости, которая наилучшим образом разделяет данные в обучающей выборке. Поиск такой гиперплоскости является задачей оптимизации и может вызывать необходимость перехода в другое признаковое пространство, в том числе, большей размерности.

При этом, может получиться, что существует несколько гиперплоскостей, разделяющих отдельные классы в обучающей выборке. Вследствие этого, метод опорных вектор подбирает разделяющую гиперплоскость, максимизируя отступ (margin), простирающийся до ближайшей точки (рисунок 1.6). При этом точки, попадающие на границу отступа, являются ключевыми элементами аппроксимации и задают опорные вектора, в честь которых и назван метод.

Возможности метода опорных векторов расширяются с использованием ядер (kernels). Если данные не могут быть разделены линейно, как на рисунке 1.6б, имеет смысл использовать какую-либо другую функцию, например, радиальную.

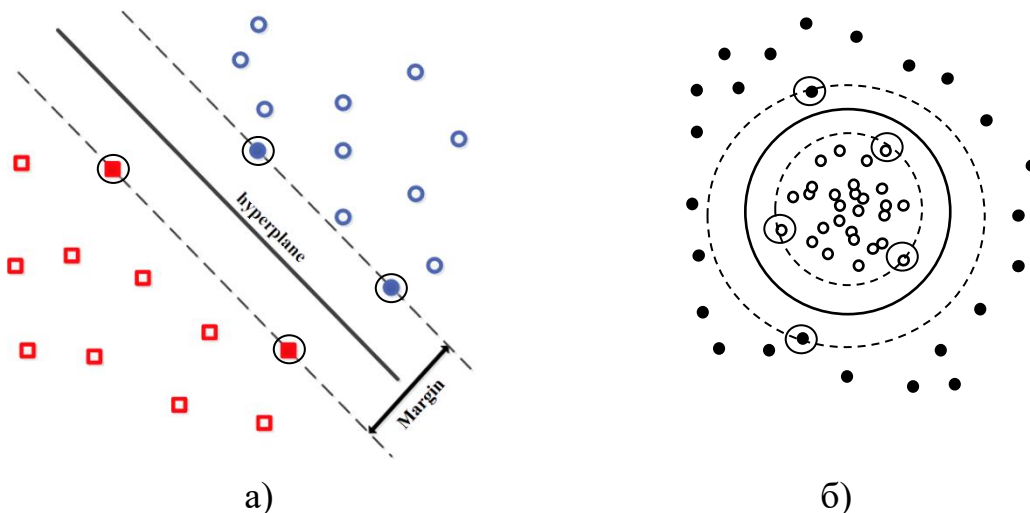


Рисунок 1.6 – Визуальная иллюстрация работы метода опорных векторов: а) использование линейного ядра; б) использование радиальной базисной функции

Одна из применяемых с этой целью стратегий состоит в вычислении базисных функций, центрированных по каждой из точек набора данных, с тем чтобы далее алгоритм SVM проанализировал полученные результаты. Эта разновидность преобразования базисных функций, известная под названием преобразования ядра (kernel transformation), основана на отношении подобия между каждой парой точек. На практике для снижения вычислительной сложности чаще всего применяется процедура под названием kernel trick (https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D0%B5%D1%80%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4).

Для разделения перекрывающихся данных в реализациях метода SVM обычно вводится небольшой поправочный параметр для «размытия» отступа.

Данный параметр допускает, что некоторые точки могут «заходить» на отступ, если это приводит к лучшей аппроксимации.

Реализация линейного метода опорных векторов в scikit-learn: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC>

Логистическая регрессия

Регрессионный анализ — набор статистических методов исследования влияния одной или нескольких независимых переменных X_i на зависимую переменную Y . В зависимости от аппроксимационной функции проводимый регрессионный анализ может различным образом описывать данные. Для задачи классификации наиболее популярной является логистическая регрессия.

В модели логистической регрессии используется логистическая функция:

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}},$$

где θ — вектор параметров, x — входной вектор признаков.
Вид функции изображен на рисунке 1.7.

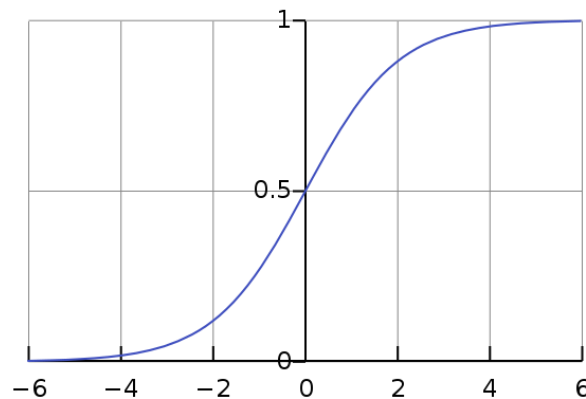


Рисунок 1.7. – Логистическая функция

Параметры θ могут быть подобраны на основе минимизации функции стоимости, например, по методу градиентного спуска:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2,$$

где m — размер обучающей выборки, x — значения признаков i -ого элемента обучающей выборки, y_i — выходное значение i -ого элемента обучающей выборки.

Подробнее о реализации логистической регрессии в scikit-learn:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Нейронные сети прямого распространения

Простейшая модель нейронной сети – это однослойный перцептрон, который представляет собой приближенную математическую модель одиночного нейрона, состоящего из n входящих сигналов или значений. Он вычисляет взвешенную сумму входящих сигналов и активизируется при достижении некоторого порогового значения.

Чуть более сложной моделью являются нейронная сеть прямого распространения, состоящая из нескольких слоев нейронов. Обязательным является наличие одного входного, одного выходного и вариативное количество скрытых слоев (рисунок 1.8).

Значения, поступающие с предыдущего слоя, умножаются на весовые коэффициенты связи (между соответствующими нейронами), и затем сумма таких взвешенных значений с предыдущего слоя поступает на вход активационной функции нейрона следующего слоя.

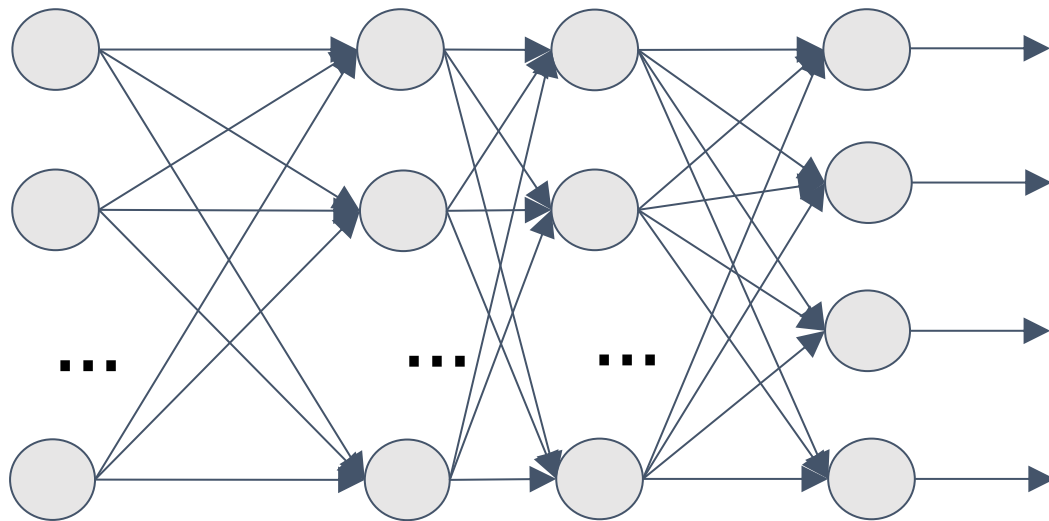


Рисунок 1.8. Структура многослойного перцептрона прямого распространения

В качестве активационных функций выступают линейная, пороговая, сигмоидальные, функции семейства ReLU, Swish, SoftPlus и др. (https://en.wikipedia.org/wiki/Activation_function).

Веса связей нейронных сетей подбираются в процессе обучения. Наиболее известным алгоритмом обучения является алгоритм градиентного

спуска, реализуемый на нейронных сетях с помощью метода обратного распространения ошибки.

Примеры использования библиотеки scikit-learn для построения нейронных сетей:

https://scikit-learn.org/stable/modules/neural_networks_supervised.html .

Метод k ближайших соседей

Прогнозная модель классификации на основе ближайших соседей является одной из простейших. В ней не делается никаких математических допущений и не требуется какого-то тяжелого функционального аппарата. Единственное, что требуется, это:

- некоторое представление о расстоянии;
- предположение, что точки, расположенные близко друг к другу, подобны.

Больше методов классификации обращаются ко всей выборке в целом с тем, чтобы в результате обучения извлечь из данных схемы или закономерности. В отличие от них метод ближайших соседей вполне осознанно пренебрегает значительной частью информации, поскольку предсказание для любой новой точки зависит всего лишь от нескольких ближайших к ней точек.

В данном методе на первом этапе необходимо найти k ближайших элементов обучающей выборки и посмотреть на их результат. Чаще всего отдельные элементы выборки представлены в векторном виде, и чтобы посчитать расстояние между ними необходимо введение некоторой метрики. В качестве вводимой метрики может выступать евклидово расстояние, расстояние Махаланобиса и т.д. При этом число k может быть задано произвольно.

На втором этапе алгоритма принимается итоговое решение на основании голосования по результатам ближайших соседей. При этом, если количество голосов равнозначное, итоговое решение может быть принято по одной из следующих стратегий:

- выбрать победителя случайным образом;
- взвесить голоса по удаленности и выбрать максимального взвешенного победителя;
- уменьшать k, пока не будет найден единственный победитель.

Библиотека scikit-learn располагает большим количеством моделей на основе метода ближайших соседей (<https://scikit-learn.org/stable/modules/neighbors.html>).

Наивный байесовский классификатор

Задачу классификации можно рассматривать с точки зрения теории вероятности и интерпретировать ее постановку, как определение наиболее вероятного исхода. При этом, при наличии какого-то числа возможных исходов, которые могут привести или не привести к наступлению более сложного события, необходимо учитывать вероятности каждого из них.

Например, если рассмотреть событие «на кубике выброшено четное число очков», то к наступлению этого события могут привести или не привести наступления каждого из 6 возможных исходов.

То есть B_i – вероятность выбросить i очков на кубике, и для всех i она будет равна $1/6$, а $P(A | B_i)$ – вероятность того, что число очков чётное при условии того, что выпало i очков – соответственно $P(A | B_1) = 0$, $P(A | B_2) = 1$, $P(A | B_3) = 0$, $P(A | B_4) = 1$, $P(A | B_5) = 0$, $P(A | B_6) = 1$

Когда рассматривается теорема Байеса, подразумевается, что какое-то сложное событие уже наступило. При помощи теоремы Байеса можно определить, с какой вероятностью к наступлению этого сложного события привёл каждый из элементарных исходов.

Классическая задача на теорему Байеса – задача про монету со смещенным центром тяжести. Допустим, имеются 2 монеты – одна обычная (с вероятностью выбросить орёл и решку $1/2$) и одна со смещенным центром тяжести, у которой вероятность выбросить орла $1/3$, а решку – $2/3$. Из двух монет наудачу выбирается одна и подбрасывается. Какая вероятность выпадения орла?

Для решения задачи в такой формулировке нужно использовать формулу полной вероятности – сложить вероятность выбросить орла при условии выбора обычной монеты, умноженную на вероятность выбора обычной монеты, и вероятность выбросить орла при условии выбора кривой монеты, умноженную на вероятность выбора кривой монеты.

Теперь если задачу сформулировать так: в результате вышеописанного эксперимента выпал орёл. Какова вероятность, что бросалась монета со смещенным центром тяжести? Для этого как раз и можно использовать теорему Байеса.

Основная идея в следующем – при помощи формулы полной вероятности можно определить так называемую априорную вероятность – вероятность наступления события в будущем, ещё до того, как произошёл случайный эксперимент. При помощи теоремы Байеса можно определить апостериорную вероятность – то есть вероятность того, что событие произошло в результате того или иного исхода, по уже наступившему событию в прошлом, уже после того, как эксперимент произошёл.

Если событие A наступило при условии B , то вероятность того, что именно наступление B привело к наступлению A , определяется по теореме Байеса:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

Наивные байесовские классификаторы основаны на байесовских методах классификации, в основе которых лежит теорема. В байесовской классификации нас интересует поиск вероятности метки (категории) при определенных заданных признаках, являющихся результатами наблюдений/экспериментов. Теорема Байеса позволяет выразить это в терминах величин, которые мы можем вычислить напрямую.

Один из способов выбора между двумя метками — вычислить отношение апостериорных вероятностей для каждой из них, то есть ответить на вопрос, какова вероятность такого значения признака при известном результате.

Все, что нужно, — модель, с помощью которой можно было бы вычислить вероятности признаков для каждой из меток. Подобная модель называется порождающей моделью (generative model), поскольку определяет гипотетический случайный процесс генерации данных. Задание порождающей модели для каждой из меток/категорий — основа обучения подобного байесовского классификатора.

Обобщенная версия подобного шага обучения — непростая задача, но возможно ее упрощение на основе принятия допущений о виде модели.

Именно на этом этапе возникает слово «наивный» в названии «наивный байесовский классификатор»: сделав очень «наивное» допущение относительно порождающей модели для каждой из меток/категорий, можно будет отыскать грубое приближение порождающей модели для каждого класса, после чего перейти к байесовской классификации. Различные виды наивных байесовских классификаторов основываются на различных «наивных» допущениях относительно данных.

Вероятно, самый простой для понимания наивный байесовский классификатор — Гауссов. В этом классификаторе допущение состоит в том, что данные всех категорий взяты из простого нормального распределения. Допустим, имеются следующие данные, изображенные на рисунке 1.7. Отобразить данные можно с помощью следующих вызовов функций:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```



```

from sklearn.datasets import make_blobs
X, y = make_blobs(100, 2, centers=2, random_state=2,
cluster_std=1.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu');

```

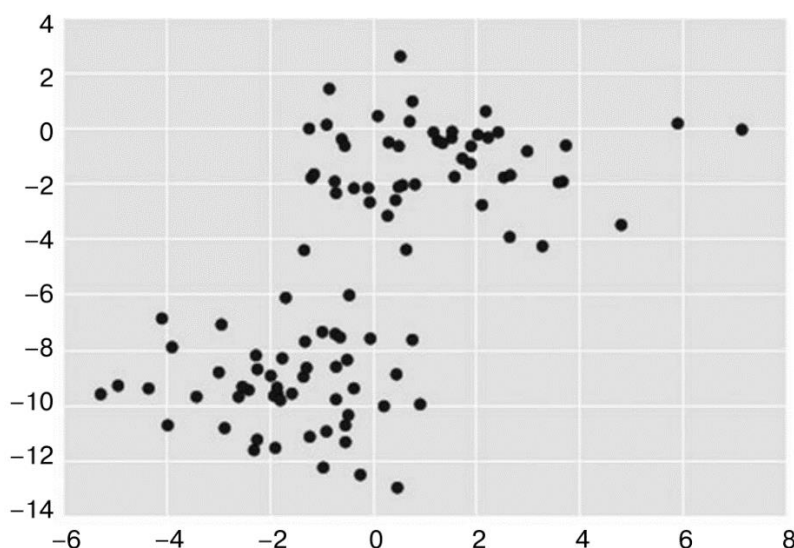


Рис. 1.7. Данные для наивной байесовской классификации

Один из самых быстрых способов создания простой модели — допущение о том, что данные подчиняются нормальному распределению без ковариации между измерениями. Для обучения этой модели достаточно найти среднее значение и стандартное отклонение точек внутри каждой из категорий — это все, что требуется для описания подобного распределения. Результат этого наивного Гауссова допущения показан на рис. 1.8.

Эллипсы на этом рисунке представляют Гауссову порождающую модель для каждой из меток с ростом вероятности по мере приближении к центру эллипса.

С помощью этой порождающей модели для каждого класса можно легко вычислить вероятность для каждой точки данных, а следовательно, быстро рассчитать соотношение для апостериорной вероятности и определить, какая из меток с большей вероятностью соответствует конкретной точке.

Эта процедура реализована в оценителе `sklearn.naive_bayes.GaussianNB`:

```

from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X, y);

```

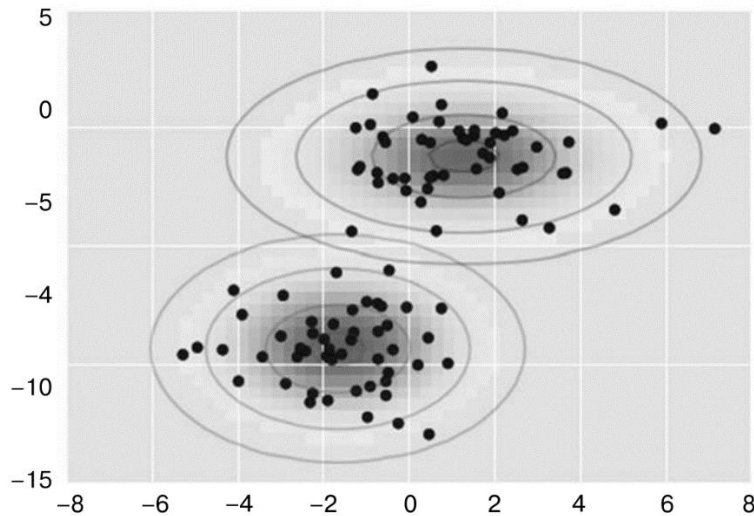


Рис. 1.8. Визуализация Гауссовой наивной байесовской модели

Сгенерируем какие-нибудь новые данные и выполним предсказание метки:

```
rng = np.random.RandomState(0)
Xnew = [-6, -14] + [14, 18] * rng.rand(2000, 2)
ynew = model.predict(Xnew)
```

Теперь есть возможность построить график этих новых данных и понять, где пролегает граница принятия решений (decision boundary) (рис. 1.9):

```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu')
lim = plt.axis()
plt.scatter(Xnew[:, 0], Xnew[:, 1], c=ynew, s=20, cmap='RdBu',
            alpha=0.1)
plt.axis(lim);
```

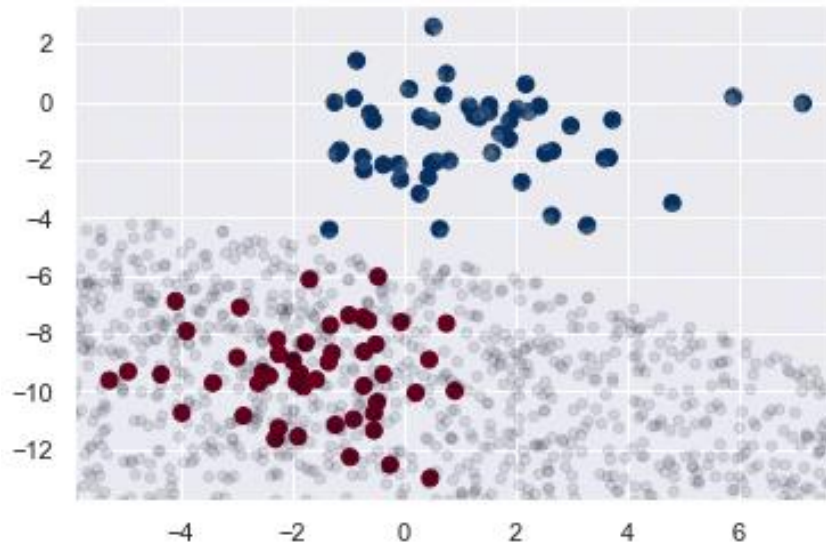


Рис. 1.9. Визуализация Гауссовой наивной байесовской классификации

Видно, что граница слегка изогнута, в целом граница при Гауссовом наивном байесовском классификаторе соответствует кривой второго порядка.

Положительная сторона этого байесовского формального представления заключается в возможности естественной вероятностной классификации, рассчитать которую можно с помощью метода `predict_proba`:

```
yprob = model.predict_proba(Xnew)
yprob[-8:].round(2)
Out[6]: array([[ 0.89,  0.11],
 [ 1. ,  0. ],
 [ 1. ,  0. ],
 [ 1. ,  0. ],
 [ 1. ,  0. ],
 [ 1. ,  0. ],
 [ 0. ,  1. ],
 [ 0.15,  0.85]])
```

Столбцы отражают апостериорные вероятности первой и второй меток соответственно. Подобные байесовские методы могут оказаться весьма удобным подходом при необходимости получения оценок погрешностей в классификации.

Качество получаемой в итоге классификации не может превышать качества исходных допущений модели, поэтому Гауссов наивный байесовский классификатор зачастую не демонстрирует слишком хороших результатов. Тем не менее во многих случаях — особенно при значительном количестве

признаков — исходные допущения не настолько плохи, чтобы нивелировать удобство Гауссова наивного байесовского классификатора.

Гауссово допущение — далеко не единственное простое допущение, которое можно использовать для описания порождающего распределения для всех меток. Еще один полезный пример — полиномиальный наивный байесовский классификатор с допущением о том, что признаки сгенерированы на основе простого полиномиального распределения. Полиномиальное распределение описывает вероятность наблюдения количеств вхождений в несколько категорий, таким образом, полиномиальный наивный байесовский классификатор лучше всего подходит для признаков, отражающих количество или частоту вхождения.

Основная идея остается точно такой же, но вместо моделирования распределения данных с оптимальной Гауссовой функцией мы моделируем распределение данных с оптимальным полиномиальным распределением.

В силу столь строгих допущений относительно данных наивные байесовские классификаторы обычно работают хуже, чем более сложные модели. Тем не менее у них есть несколько достоинств:

- они выполняют как обучение, так и предсказание исключительно быстро;
- обеспечивают простое вероятностное предсказание;
- их результаты часто очень легки для интерпретации;
- у них очень мало (если вообще есть) настраиваемых параметров.

Метрики оценки качества алгоритмов и моделей.

Для корректного определения результатов работы алгоритма необходимо использовать данные, на которых алгоритм не обучался. Для этого используется разделение выборки на обучающую и тестовую, а иногда еще и на валидационную. Пример разделения обучающей выборки на 2 части:

```
from sklearn.model_selection import train_test_split
dataset_train, dataset_test = train_test_split(data_all,
test_size=0.2, shuffle=False)
```

Часто в задачах классификации недостаточно оценить процент только правильных результатов. Особенно показательным является пример в задачах при неравномерном распределении выборок, относящихся к положительному и отрицательному классу. Если в выборке 95% процентов данных относятся к одному классу, то автоматическое принятие решения и отнесение всех

результатов к этому классу будет давать достаточно хороший результат (95% точности), однако, абсолютно не приблизит к решению задачи.

Поэтому при бинарной классификации наиболее популярными подходами является определение более специфических метрик оценки результатов работы алгоритма. В данной лабораторной работе необходимо использовать такие оценки, как Precision, Recall, Accuracy, F1 и Confusion Matrix:

<https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics>

Пример вывода метрики для переменных `y_test` (значения из выборки) и `y_actual` (значения, полученные на тестовой выборке с помощью построенной модели):

```
accuracy_score(y_test, y_actual)
```

Задача 1. Сердечно-сосудистые заболевания

На сегодняшний день болезни сердца являются одной из основных причин смерти людей по данным различных стран. Ваша задача заключается в обработке собранного набора данных, включающего факторы риска возникновения сердечно-сосудистых заболеваний и последующей разработке алгоритма бинарной классификации, определяющего есть ли у человека заболевание.

Источник данных:

<https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease>

Задача 2. Качество воды

Доступ к безопасной и чистой воде необходим для сохранения здоровья человека. Ваша задача заключается в обработке собранного набора данных, включающего показатели качества воды и последующей разработке алгоритма бинарной классификации, определяющего пригодна ли вода для потребления человеком.

Источник данных:

<https://www.kaggle.com/datasets/adityakadiwal/water-potability>

Задача 3. Стартапы

Стартапы играют важную роль в экономическом росте, приносят новые идеи, стимулируют инновации, создают рабочие места. При этом такие проекты сталкиваются с высокой степенью неопределенности, имеют высокий процент неудач, и лишь единицы становятся успешными и прибыльными. Ваша задача заключается в обработке собранного набора данных,

включающего отраслевые тенденции, информацию об инвестициях и информацию об отдельных компаниях и последующей разработке алгоритма бинарной классификации, определяющего будет ли успешным очередной стартап-проект.

Источник данных:

<https://www.kaggle.com/datasets/manishkc06/startup-success-prediction>

Задача 4. Задержка рейса

Время – ценнейший человеческий ресурс. Различные нестыковки в расписании и перенесенные рейсы вызывают как потерю времени, так и финансовые затраты. Ваша задача заключается в обработке собранного набора данных, включающего информацию об авиарейсах, аэропортах и погодных условиях и последующей разработке алгоритма бинарной классификации, определяющего будет ли задерживаться рейс.

Источник данных:

<https://www.kaggle.com/datasets/threnjen/2019-airline-delays-and-cancellations>

Задача 5. Вредоносные сайты

Распространение вредоносных веб-сайтов вызывает серьезную обеспокоенность, так как сложно анализировать один за другим и индексировать каждый URL в черном списке. Ваша задача заключается в разработке алгоритма бинарной классификации для выявления таких сайтов.

Источник данных:

<https://www.kaggle.com/xwolf12/malicious-and-benign-websites>

Задание к лабораторной 1.

1. Получите задание в соответствии с вашим вариантом (номер задачи зависит от номера группы, а исследуемые методы от варианта):

№ варианта	Используемые методы для решения
1	<ul style="list-style-type: none">• Случайные леса• Нейронные сети прямого распространения
2	<ul style="list-style-type: none">• Метод опорных векторов• Нейронные сети прямого распространения
3	<ul style="list-style-type: none">• Логистическая регрессия• Нейронные сети прямого распространения
4	<ul style="list-style-type: none">• Метод k ближайших соседей• Нейронные сети прямого распространения
5	<ul style="list-style-type: none">• Наивный байесовский классификатор

	<ul style="list-style-type: none"> • Нейронные сети прямого распространения
6	<ul style="list-style-type: none"> • Деревья принятия решений • Нейронные сети прямого распространения

2. Проанализируйте датасет. Опишите его характеристики: имеющиеся колонки, размер. Преобразуйте датасет, приведя все данные к виду, пригодному для последующей обработки (колонки, содержащие дату и время, строковые некатегориальные значения и т.д.). Отобразите корреляционную матрицу. Предложите варианты по расширению датасета (то есть каким образом можно было бы собрать больше данных в исследуемой предметной области).
3. Решите задачу бинарной классификации на предложенном датасете методами, соответствующими вашему варианту. Исследуйте параметры предложенных методов, найдите наиболее оптимальные значения для вашей задачи. Обоснуйте полученные результаты.
4. Оцените полученные результаты с помощью описанных в тексте лабораторной метрик. Проведите анализ элементов датасета, которые были классифицированы ошибочно.
5. Попробуйте улучшить работу алгоритмов с помощью изменения различных параметров, дополнительной обработки датасета и т.д.
6. Представьте ваше исследование в виде Блокнота JupyterLab, содержащего все пункты задания.