**ФГБОУ ВО   
Уфимский университет науки и технологий**

**Кафедра ВМиК**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Оценка качества методов классификации

**ОТЧЕТ**

**по лабораторной работе**

**по** Основам машинного обучения

(*наименование дисциплины*)

|  |
| --- |
| Лабораторная работа 2 |
| (обозначение документа) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа |  |  | Фамилия, И., О. | Подпись | Дата | Оценка |
| МО-325Б |  |
|  |  |
| Студент | | | Шарыгин М.С.,  Агафонов Р.В.,  Лепоринский Г.А. |  |  |  |
| Преподаватель | | | Миронов К.В. |  |  |  |
| Принял | | |  |  |  |  |

**Уфа 2025 г****.**

Содержание

[1 Цель работы 3](#_Toc193501225)

[2 Практическая часть 4](#_Toc193501226)

[2.1 Подготовка датасета 4](#_Toc193501227)

[2.2 Подготовка таблиц и массивов 4](#_Toc193501228)

[2.3 Дополнительные функции 5](#_Toc193501229)

[2.4 Метод k-ближайших соседей 6](#_Toc193501230)

[2.5 Метод радиус-ближайших соседей 7](#_Toc193501231)

[2.6 Метод центройда 7](#_Toc193501232)

[2.7 Метод дерева решений 7](#_Toc193501233)

[2.8 Метод случайного леса 8](#_Toc193501234)

[2.9 Метод наивного Байеса 8](#_Toc193501235)

[2.10 Метод машины опорных векторов 9](#_Toc193501236)

[2.11 Сравнение результатов 9](#_Toc193501237)

[3 Вывод 11](#_Toc193501238)

[4 Приложение 12](#_Toc193501239)

[4.1 Код программы 12](#_Toc193501240)

# Цель работы

Оценить качество методов классификации с помощью «10-Validation» и выявить их основные гиперпараметры.

# Практическая часть

## Подготовка датасета

Для начала необходимо импортировать и нормализовать датасет, а также заполнить таблицу собственных векторов и значений так, как показано на рисунке 2.1[[1]](#footnote-1).

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.1 – Подготовка датасета

## Подготовка таблиц и массивов

Затем необходимо функцией «PrettyTable()» создать таблицы для хранения результатов выполнения каждого метода, в которые с помощью «field\_names» добавим колонки; также для каждого метода и лучших результатов нужно создать массивы, которые будут использованы для сохранения лучших результатов (рисунок 2.2).

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.2 – Создание таблиц и массивов

## Дополнительные функции

Для начала реализуем функцию «search\_for\_best\_results», которая будет принимать в себя следующие параметры:

* «table» – таблица, в которую будут записаны полученные на данном шаге результаты;
* «result» – массив, хранящий лучший результат до текущего шага;
* «dimension» – размерность пространства на текущем шаге;
* «accuracy» – полученная точность при заданных параметрах;
* «period» – время выполнения классификации;
* «parameter\_1» – первый гиперпараметр;
* «parameter\_2» – второй гиперпараметр.

Она [функция] будет сравнивать результаты текущего шага с лучшими и если текущие окажутся лучше, то они будут сохранены на их месте [месте лучших], как показано на рисунке 2.3. Текущие результаты также будут сохранены в соответствующей таблицы своего метода.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.3 – Функция «search\_for\_best\_results»

Теперь реализуем функцию «table\_best\_formation», добавляющую лучшие результаты в соответствующую таблицу. Функция приведена на рисунке 2.4.

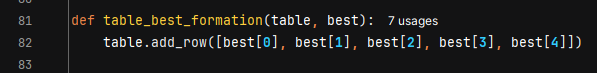


Рисунок 2.4 – Функция «table\_best\_formation»

## Метод k-ближайших соседей

Сначала запустим цикл «for», который будет задавать размерность пространства от одного до всех параметров. После этого применим метод «PCA» для сокращения размерности пространства и функцию «train\_test\_split» для разделения выборки на обучающую и тренировочную – данные действия выполняются однажды для всех методов сразу (рисунок 2.5).

Изображение выглядит как снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.5 – Сокращение размерности и разделение выборки

Теперь реализуем «метод k-ближайших соседей», который был разобран в предыдущей лабораторной работе. Единственным отличием станет использование метода «10-Validation» для оценки качества вместо «Leave-One-Out» (рисунок 2.6); для «метода радиус-ближайших соседей» (рисунок 2.7) и «метода центройда» (рисунок 2.8) это также станет единственным отличием

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.6 – Метод k-ближайших соседей

## Метод радиус-ближайших соседей

Изображение выглядит как текст, программное обеспечение, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.7 – Метод радиус-ближайших соседей

## Метод центройда

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.8 – Метод центройда

## Метод дерева решений

Реализуем «метод дерева решений», в котором можно выделить следующие основные гиперпараметры и который приведен на рисунок 2.9:

* «criterion» – определяет функцию качества разбиения («gini», «entropy» или «log\_loss»), которая используется для выбора наилучшего признака и порога при построении дерева;
* «splitter» – определяет  стратегию выбора разбиения («best» – лучшая или «random» – случайное) на каждом узле дерева.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.9 – Метод дерева решений

## Метод случайного леса

В «методе случайного леса» (рисунок 2.10) можно выделить:

* «n\_estimators» – определяет количество деревьев в лесу;
* «criterion» – определяет функцию качества разбиения («gini», «entropy» или «log\_loss»), которая используется для выбора наилучшего признака и порога при построении дерева.

Изображение выглядит как текст, программное обеспечение, Шрифт, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.10 – Метод случайного леса

## Метод наивного Байеса

Теперь используем «метод наивного Байеса», а именно «GaussianNB», который изображен на рисунке 2.11 и в котором можно выделить гиперпараметр «var\_smoothing», отвечающий за добавление небольших величин к дисперсии признаков, что помогает избежать нулевых дисперсий, которые могут возникнуть, если в данных есть признаки с постоянным значением.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.11 – Метод наивного Байеса

## Метод машины опорных векторов

Наконец применим «метод машины опорных векторов» (рисунок 2.12), в котором выделим:

* «kernel» – указывает тип ядра, который будет использоваться в алгоритме;
* «degree» – степень функции полиномиального ядра (т.е. он [гиперпараметр] игнорируется всеми другими ядрами).

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.12 – Метод машины опорных векторов

## Сравнение результатов

После выполнения всех классификаций добавим информацию о лучших результатах в сводную таблицу (каждая строка в этой таблице – это наилучший результат по точности и времени выполнения при разных размерностях и гиперпараметрах), как показано на рисунках 2.13 и 2.14.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.13 – Добавлений строк с лучшими результатами в сводную таблицу

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.14 – Таблица лучших результатов

Большинство методов показали одинаковые точности (0.66 – 0.67) и время выполнения (0.01 – 0.03 секунд), исключение составили «метод центройда» и «метод дерева решений», которые показали чуть худшее время, и «метод случайного леса», чьё время выполнения – 2.52 секунды. При этом «Метод наивного Байеса» показал лучшее время при меньшей размерности.

Таким образом, можно отметить, что из-за особенности выборки данных в данном датасете большой разницы в результатах применения между рассматриваемыми методами нет. Причины таких результатов заключаются в следующем:

* неравномерная выборка: объектов класса «Other» значительно больше остальных (316 против 77 и 47);
* объекты выборки плохо разделимы: при одинаковых характеристиках имеются примеры объектов разных классов;
* неравномерные расстояния между объектами разных классов.

# Вывод

В ходе лабораторной работы мы оценили качество методов классификации с помощью «10-Validation» и выявили их основные гиперпараметры.

# Приложение

## Код программы

from ucimlrepo import fetch\_ucirepo  
from sklearn import preprocessing  
from sklearn.decomposition import PCA  
from sklearn.model\_selection import cross\_val\_score**,** train\_test\_split  
from sklearn.neighbors import KNeighborsClassifier**,** RadiusNeighborsClassifier**,** NearestCentroid  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.svm import SVC  
from prettytable import PrettyTable  
import numpy as np  
import time  
  
  
  
# Импорт датасета  
wholesale\_customers = fetch\_ucirepo(id=**292**)  
x = wholesale\_customers.data.features  
y = wholesale\_customers.data.targets  
  
  
  
# Нормализация датасета  
x\_train = np.array(x)  
x\_scaled = preprocessing.StandardScaler().fit\_transform(x\_train)  
y = np.array(y).ravel()  
  
  
  
# Заполнение таблицы собственных векторов и значений  
pca = PCA()  
X = pca.fit\_transform(x\_scaled)  
eigenvectors = pca.fit\_transform(X)  
eigenvalues = pca.explained\_variance\_  
  
  
  
# Создание таблиц для представления результатов  
table\_knc = PrettyTable()  
table\_rnc = PrettyTable()  
table\_nc = PrettyTable()  
table\_dtc = PrettyTable()  
table\_rfc = PrettyTable()  
table\_gnb = PrettyTable()  
table\_svc = PrettyTable()  
table\_best = PrettyTable()  
  
# Добавление колонок в таблицы  
table\_knc.field\_names = ["Размерность пространства"**,** "Точность"**,** "Время выполнения"**,** "k"**,** "Вес" ]  
table\_rnc.field\_names = ["Размерность пространства"**,** "Точность"**,** "Время выполнения"**,** "Радиус"**,** "Вес" ]  
table\_nc.field\_names = ["Размерность пространства"**,** "Точность"**,** "Время выполнения"**,** "Метрика"**,** "-" ]  
table\_dtc.field\_names = ["Размерность пространства"**,** "Точность"**,** "Время выполнения"**,** "Критерий"**,** "Разделитель"]  
table\_rfc.field\_names = ["Размерность пространства"**,** "Точность"**,** "Время выполнения"**,** "Кол-во деревьев"**,** "Критерий" ]  
table\_gnb.field\_names = ["Размерность пространства"**,** "Точность"**,** "Время выполнения"**,** "Сглаживание"**,** "-" ]  
table\_svc.field\_names = ["Размерность пространства"**,** "Точность"**,** "Время выполнения"**,** "Ядро"**,** "Степень" ]  
table\_best.field\_names = ["Размерность пространства"**,** "Точность"**,** "Время выполнения"**,** "Параметр 1"**,** "Параметр 2" ]  
  
# Создание массивов для поиска лучших результатов  
best\_knc = [**0, 0, 0,** ""**,** ""]  
best\_rnc = [**0, 0, 0,** ""**,** ""]  
best\_nc = [**0, 0, 0,** ""**,** ""]  
best\_dtc = [**0, 0, 0,** ""**,** ""]  
best\_rfc = [**0, 0, 0,** ""**,** ""]  
best\_gnb = [**0, 0, 0,** ""**,** ""]  
best\_svc = [**0, 0, 0,** ""**,** ""]  
  
  
  
def search\_for\_best\_results (table**,** result**,** dimension**,** accuracy**,** period**,** parameter\_1**,** parameter\_2):  
 if (result[**1**] < accuracy) or (result[**1**] == accuracy and result[**2**] > period):  
 result[**0**] = dimension  
 result[**1**] = accuracy  
 result[**2**] = period  
 result[**3**] = parameter\_1  
 result[**4**] = parameter\_2  
  
 table.add\_row([dimension**,** accuracy**,** period**,** parameter\_1**,** parameter\_2])  
  
  
  
def table\_best\_formation(table**,** best):  
 table.add\_row([best[**0**]**,** best[**1**]**,** best[**2**]**,** best[**3**]**,** best[**4**]])  
  
  
  
# Цикл по всем размерностям  
for component in range(**1,** len(eigenvalues) + **1**):  
 pca = PCA(n\_components=component)  
 X = pca.fit\_transform(x\_scaled)  
 X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.3,** random\_state=**42**)  
  
  
  
 # Метод k-ближайших соседей  
 for weight in ["uniform"**,** "distance"]:  
 for k in range(**1, 30, 2**):  
 knc = KNeighborsClassifier(n\_neighbors=k**,** weights=weight)  
  
 start\_time = time.time()  
 scores = cross\_val\_score(knc**,** X\_train**,** y\_train**,** cv=**10**)  
 end\_time = time.time()  
  
 search\_for\_best\_results(table\_knc**,** best\_knc**,** component**,** scores.mean()**,** end\_time - start\_time**,** k**,** weight)  
  
  
  
 # Метод радиус-ближайших соседей  
 for weight in ["uniform"**,** "distance"]:  
 for radius in range(**15, 25**):  
 rnc = RadiusNeighborsClassifier(radius=radius**,** weights=weight)  
  
 start\_time = time.time()  
 scores = cross\_val\_score(rnc**,** X\_train**,** y\_train**,** cv=**10**)  
 end\_time = time.time()  
  
 search\_for\_best\_results(table\_rnc**,** best\_rnc**,** component**,** scores.mean()**,** end\_time - start\_time**,** radius**,** weight)  
  
  
  
 # Метод центройда  
 for metric in ["euclidean"**,** "manhattan"]:  
 nc = NearestCentroid(metric=metric)  
  
 start\_time = time.time()  
 scores = cross\_val\_score(nc**,** X\_train**,** y\_train**,** cv=**10**)  
 end\_time = time.time()  
  
 search\_for\_best\_results(table\_nc**,** best\_nc**,** component**,** scores.mean()**,** end\_time - start\_time**,** metric**,** "-")  
  
  
  
 # Метод дерева решений  
 for criterion in ["gini"**,** "entropy"**,** "log\_loss"]:  
 for splitter in ["best"**,** "random"]:  
 dtc = DecisionTreeClassifier(criterion=criterion**,** splitter=splitter)  
  
 start\_time = time.time()  
 scores = cross\_val\_score(dtc**,** X\_train**,** y\_train**,** cv=**10**)  
 end\_time = time.time()  
  
 search\_for\_best\_results(table\_dtc**,** best\_dtc**,** component**,** scores.mean()**,** end\_time - start\_time**,** criterion**,** splitter)  
  
  
  
 # Метод случайного леса  
 for n in range(**20, 200, 20**):  
 for criterion in ["gini"**,** "entropy"**,** "log\_loss"]:  
 rfc = RandomForestClassifier(n\_estimators=n**,** criterion=criterion)  
  
 start\_time = time.time()  
 scores = cross\_val\_score(rfc**,** X\_train**,** y\_train**,** cv=**10**)  
 end\_time = time.time()  
  
 search\_for\_best\_results(table\_rfc**,** best\_rfc**,** component**,** scores.mean()**,** end\_time - start\_time**,** n**,** criterion)  
  
  
  
 # Метод наивного Байеса  
 for var\_smoothing in range(-**10, 11**):  
 gnb = GaussianNB(var\_smoothing=(pow(**10,** var\_smoothing)))  
  
 start\_time = time.time()  
 scores = cross\_val\_score(gnb**,** X\_train**,** y\_train**,** cv=**10**)  
 end\_time = time.time()  
  
 search\_for\_best\_results(table\_gnb**,** best\_gnb**,** component**,**scores.mean()**,** end\_time - start\_time**,** var\_smoothing**,** "-")  
  
  
  
 # Метод машины опорных векторов  
 for kernel in ["linear"**,** "poly"**,** "rbf"**,** "sigmoid"]:  
 for n in range(**5**):  
 svc = SVC(kernel=kernel**,** degree=n)  
  
 start\_time = time.time()  
 scores = cross\_val\_score(svc**,** X\_train**,** y\_train**,** cv=**10**)  
 end\_time = time.time()  
  
 search\_for\_best\_results(table\_svc**,** best\_svc**,** component**,** scores.mean()**,** end\_time - start\_time**,** kernel**,** n)  
  
  
  
table\_best\_formation(table\_best**,** best\_knc)  
table\_best\_formation(table\_best**,** best\_rnc)  
table\_best\_formation(table\_best**,** best\_nc)  
table\_best\_formation(table\_best**,** best\_dtc)  
table\_best\_formation(table\_best**,** best\_rfc)  
table\_best\_formation(table\_best**,** best\_gnb)  
table\_best\_formation(table\_best**,** best\_svc)  
  
  
  
print("Метод k-ближайших соседей")  
print(table\_knc)  
  
print("\nМетод радиус-ближайших соседей")  
print(table\_rnc)  
  
print("\nМетод центройда")  
print(table\_nc)  
  
print("\nМетод дерева решений")  
print(table\_dtc)  
  
print("\nМетод случайного леса")  
print(table\_rfc)  
  
print("\nМетод наивного Байеса")  
print(table\_gnb)  
  
print("\nМетод машины опорных векторов")  
print(table\_svc)  
  
print("\nЛучшие результаты")  
print(table\_best)

1. Импорт, нормализация и заполнение были подробно разобраны в лабораторной работе 1. [↑](#footnote-ref-1)