**Лабораторная работа № 4**

**МЕТРИЧЕСКИЕ МЕТОДЫ КЛАССИФИКАЦИИ**

**Цель работы:** изучение принципов построения информационных систем с использованием метрических методов классификации.

**Задача:** изучение инструментария Python для реализации алгоритмов метрической классификации; изучение методов оптимизации параметров метрической классификации; освоение модификаций kNN-метода.

. Pandas – это библиотека Python, предоставляющая широкие возможности для анализа данных. Данные, с которыми работают специалисты Data Science, часто хранятся в табличном формате (.csv, .tsv, .xlsx, …). С помощью библиотеки Pandas данные удобно загружать, обрабатывать и анализировать с помощью SQL-подобных запросов. Pandas предоставляет широкие возможности визуального анализа табличных данных в связке с библиотеками Matplotlib и Seaborn.

Основными структурами данных в Pandas являются классы Series и DataFrame. Первый из них представляет собой одномерный индексированный массив данных некоторого фиксированного типа. Второй – это двухмерная структура данных, представляющая собой таблицу, каждый столбец которой содержит данные одного типа. Можно представлять еѐ как словарь объектов типа Series. Структура DataFrame отлично подходит для представления реальных данных: строки соответствуют признаковым описаниям отдельных объектов, а столбцы соответствуют признакам.

На рис. 1 представлен код в Python Notebook для загрузки исходного набора данных.

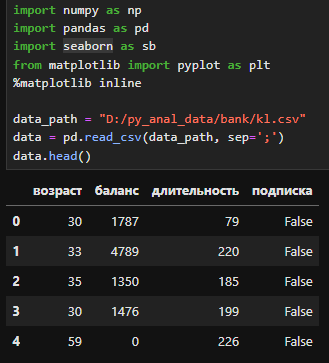


Рисунок 1 – Использование pandas для загрузки данных.

После загрузки данных можно визуализировать полученный набор данных. Для визуализации будем использовать библиотеку seaborn (рис. 2). В результате будет выведен график, отображающий распределение объектов попарно по различным признакам (рис. 3).



Рисунок 2 – Добавление шапки DataFrame.

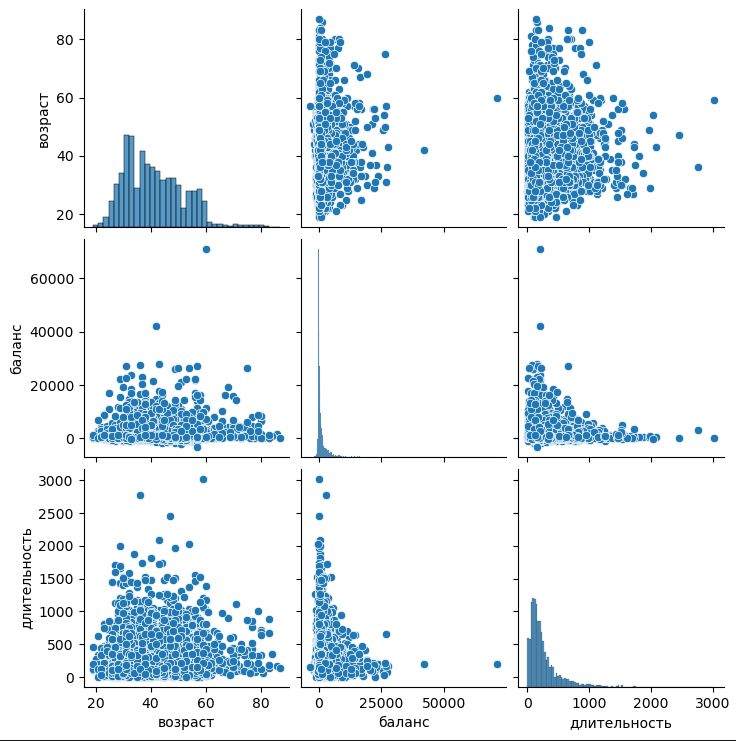


Рисунок 3 – Попарное признаковое распределение клиентов.

На графике попарного распределения видно преимущество символьного обозначения столбцов – график легче интерпретировать. Отдельные классы не отмечаются различными цветами, но видно, что на отдельных подграфиках множества точек разделены.

Для придания отдельным классам своих цветов необходимо указать, по какому признаку разделяются точки (рис. 4). Результат представлен на рис. 5.



Рисунок 4 – Построение графика с указанием признака отдельных классов

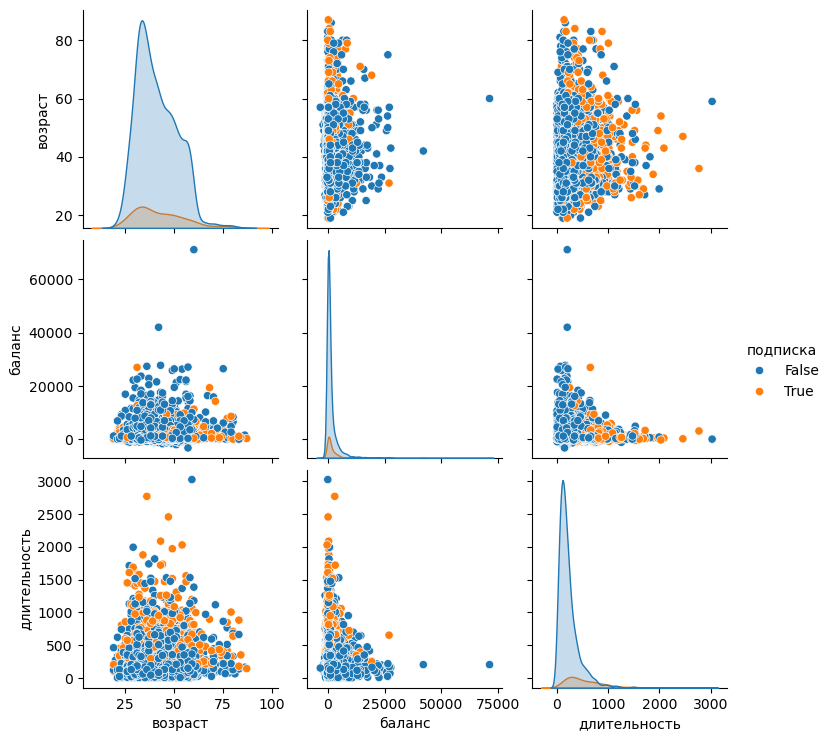


Рисунок 5 – Попарное признаковое распределение ирисов с разделением на классы.

Перейдем к построению модели. Модель метрической классификации должна обеспечивать следующий алгоритм работы: пользователь вводит новое признаковое описание объекта (объект отсутствует в обучающей выборке), а алгоритм классификации относит новый объект к одному из классов ирисов.

Существует несколько вариаций метода К-ближайших соседей. Каждая модель предполагает наличие различных параметров для оптимизации. Воспользуемся библиотекой scikit для построения классификатора (рис. 6)

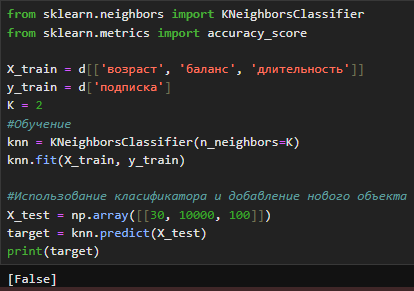


Рисунок 8 – Основные этапы решения задачи классификации методом К-ближайших соседей с использованием библиотеки scikit

Модель построена и выдает ответ для новых (отсутствующих в исходной выборке) объектов. Но, анализируя код на рис. 8, следует отметить следующие недостатки такого подхода:

– в качестве количества ближайших соседей выбрано значение K=2, выбор данного значения не обосновывается, но в данном методе именно данный параметр должен оптимизироваться;

– отсутствует какое-либо графическое представление модели, нет визуализации процесса принятия решения.

Модифицируем программный код для устранения указанных недостатков.

Займемся обоснованием выбора оптимального значения количества ближайших соседей. Для этого будем использовать простейшую оценку качества hold-out (рис. 7).

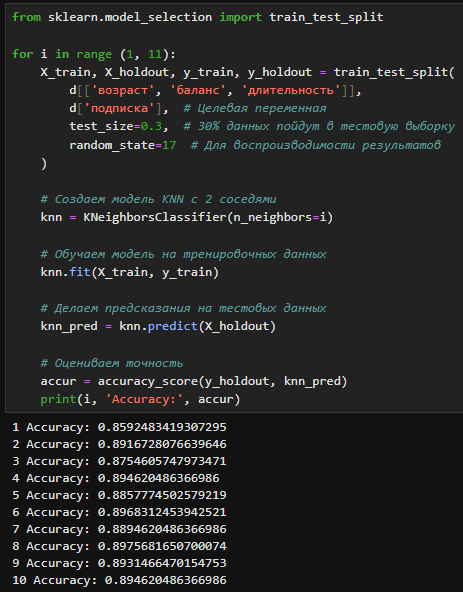


Рисунок 7 – Оценка точности классификатора с использованием методики hold-out

Я изменю значение количества соседей на 8, так как это будет более точная оценка качества.

Еще одна оценка качества – cross validation (CV) error. Cross-Validation (CV, кросс-валидация) — это метод оценки качества модели, который помогает:

1. Уменьшить зависимость от разбиения данных (в отличие от простого hold-out).
2. Более надежно оценить обобщающую способность модели.
3. Выбрать оптимальные гиперпараметры (например, k в KNN).

На рис. 10 показана реализация алгоритма получения оценки точности классификации CV и процедура выбора оптимального значения количества соседей в алгоритме kNN на основе данной оценки. Вывод для данного кода представлен на рис. 11.

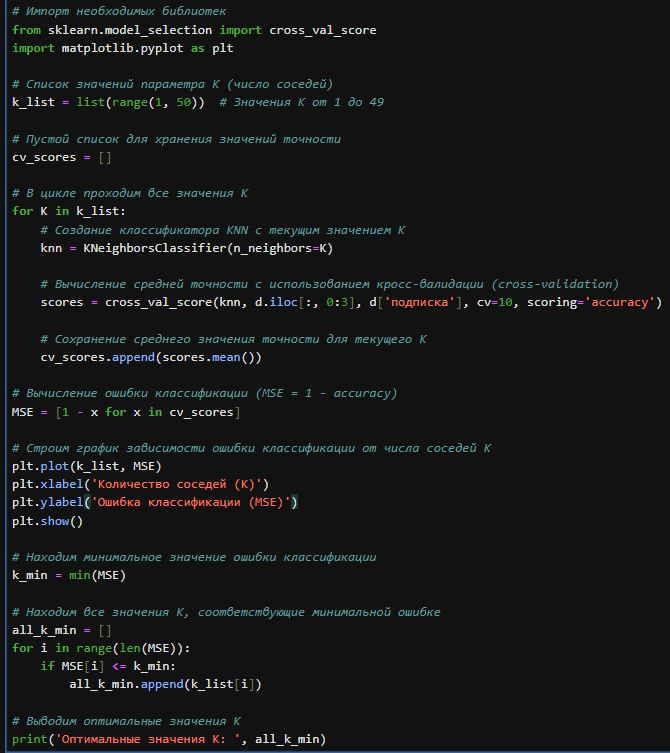


Рисунок 10 – Реализация процедуры выбора оптимального параметра на основе cross validation error.

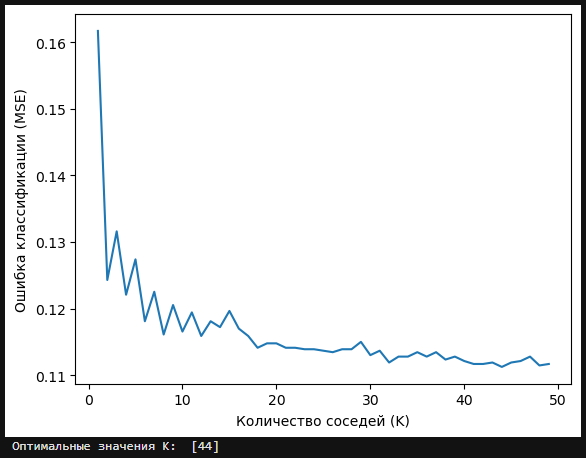


Рисунок 11 – Визуализация выбора оптимального параметра на основе cross validation error.

Теперь проверим этапы решения задачи классификации методом К-ближайших соседей, в качестве количества ближайших соседей выбрано значение K=40, так как это оценка более точная, а также CV усредняет ошибку по всем возможным разбиениям, поэтому результат надежнее.

На рисунке 12 решения задачи классификации методом К-ближайших соседей со значение К = 40 (количества ближайших соседей).

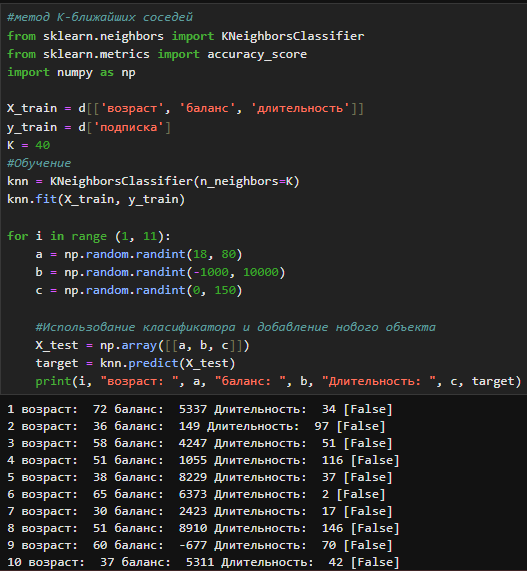


Рисунок 12 – решения задачи классификации методом К-ближайших соседей со значение К = 40

**Приложение 1**

import numpy as np

import pandas as pd

import seaborn as sb

from matplotlib import pyplot as plt

%matplotlib inline

data\_path = "D:/py\_anal\_data/bank/kl.csv"

d = pd.read\_csv(data\_path, sep=';')

data.head()

sb.pairplot(d, vars=['возраст', 'баланс', 'длительность'])

sb.pairplot(d, vars=['возраст', 'баланс', 'длительность'], hue = 'подписка')

plt.show()

#метод К-ближайших соседей

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

import numpy as np

X\_train = d[['возраст', 'баланс', 'длительность']]

y\_train = d['подписка']

K = 40

#Обучение

knn = KNeighborsClassifier(n\_neighbors=K)

knn.fit(X\_train, y\_train)

for i in range (1, 11):

a = np.random.randint(18, 80)

b = np.random.randint(-1000, 10000)

c = np.random.randint(0, 150)

#Использование класификатора и добавление нового объекта

X\_test = np.array([[a, b, c]])

target = knn.predict(X\_test)

print(i, "возраст: ", a, "баланс: ", b, "Длительность: ", c, target)

from sklearn.model\_selection import train\_test\_split

for i in range (1, 11):

X\_train, X\_holdout, y\_train, y\_holdout = train\_test\_split(

d[['возраст', 'баланс', 'длительность']],

d['подписка'], # Целевая переменная

test\_size=0.3, # 30% данных пойдут в тестовую выборку

random\_state=17 # Для воспроизводимости результатов

)

# Создаем модель KNN с 2 соседями

knn = KNeighborsClassifier(n\_neighbors=i)

# Обучаем модель на тренировочных данных

knn.fit(X\_train, y\_train)

# Делаем предсказания на тестовых данных

knn\_pred = knn.predict(X\_holdout)

# Оцениваем точность

accur = accuracy\_score(y\_holdout, knn\_pred)

print(i, 'Accuracy:', accur)

# Импорт необходимых библиотек

from sklearn.model\_selection import cross\_val\_score

import matplotlib.pyplot as plt

# Список значений параметра K (число соседей)

k\_list = list(range(1, 50)) # Значения K от 1 до 49

# Пустой список для хранения значений точности

cv\_scores = []

# В цикле проходим все значения K

for K in k\_list:

# Создание классификатора KNN с текущим значением K

knn = KNeighborsClassifier(n\_neighbors=K)

# Вычисление средней точности с использованием кросс-валидации (cross-validation)

scores = cross\_val\_score(knn, d.iloc[:, 0:3], d['подписка'], cv=10, scoring='accuracy')

# Сохранение среднего значения точности для текущего K

cv\_scores.append(scores.mean())

# Вычисление ошибки классификации (MSE = 1 - accuracy)

MSE = [1 - x for x in cv\_scores]

# Строим график зависимости ошибки классификации от числа соседей K

plt.plot(k\_list, MSE)

plt.xlabel('Количество соседей (K)')

plt.ylabel('Ошибка классификации (MSE)')

plt.show()

# Находим минимальное значение ошибки классификации

k\_min = min(MSE)

# Находим все значения K, соответствующие минимальной ошибке

all\_k\_min = []

for i in range(len(MSE)):

if MSE[i] <= k\_min:

all\_k\_min.append(k\_list[i])

# Выводим оптимальные значения K

print('Оптимальные значения K: ', all\_k\_min)