Мигачев Павел

8 группа

Задание 3. Классификация методом kNN

1. Код программы

1. Импорт библиотек

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold  
from sklearn.preprocessing import StandardScaler  
from sklearn.pipeline import Pipeline  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import confusion\_matrix, classification\_report, accuracy\_score  
from sklearn.feature\_selection import SelectKBest, f\_classif  
from sklearn.impute import SimpleImputer

1. Загрузка и подготовка данных

path = "Car\_Data.dat"  *# В файле заголовок: MODEL;P;M;R78;R77;H;R;TR;W;L;T;D;G;C*df = pd.read\_csv(  
 path,  
 sep=";",  
 header=0,  
 na\_values=["NA", "NaN", ""],  
 engine="python"  
)  
col\_map = {  
 "P": "price",  
 "M": "mileage",  
 "R78": "repair\_78",  
 "R77": "repair\_77",  
 "H": "headroom",  
 "R": "rear\_seat",  
 "TR": "trunk",  
 "W": "weight",  
 "L": "length",  
 "T": "turn\_diam",  
 "D": "displacement",  
 "G": "gear\_ratio",  
 "C": "company",  
}  
if "MODEL" in df.columns:  
 df = df.drop(columns=["MODEL"])  
df = df.rename(columns=col\_map)  
  
assert df.shape[1] == 13, f"Ожидалось 13 колонок после подготовки, получено: {df.shape[1]}"  
df["company"] = df["company"].astype(int)

1. Разделение выборки

*# 1) X/y и train/test*feature\_names = df.drop(columns=["company"]).columns.to\_numpy()  
X = df.drop(columns=["company"]).values  
y = df["company"].values  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.30, random\_state=42, stratify=y  
)

1. Построение пайплайна (импутация, стандартизация, kNN)

base\_pipe = Pipeline([  
 ("imputer", SimpleImputer(strategy="median")),  
 ("scaler", StandardScaler()),  
 ("knn", KNeighborsClassifier())  
])

1. Подборка отпимальных параметров

param\_grid = {  
 "knn\_\_n\_neighbors": list(range(1, 21)),  
 "knn\_\_weights": ["uniform", "distance"],  
 "knn\_\_metric": ["euclidean", "manhattan"]  
}  
cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)  
  
grid = GridSearchCV(  
 base\_pipe,  
 param\_grid=param\_grid,  
 cv=cv,  
 scoring="accuracy",  
 n\_jobs=-1  
)  
grid.fit(X\_train, y\_train)  
  
best\_model = grid.best\_estimator\_  
best\_params = grid.best\_params\_  
cv\_best\_score= grid.best\_score\_

print("\nЛучшие параметры (CV):", best\_params)  
print(f"Лучший accuracy по CV: {cv\_best\_score:.3f}")

1. Оценка тестовой выборке

y\_pred = best\_model.predict(X\_test)  
acc = accuracy\_score(y\_test, y\_pred)  
err\_pct = (1 - acc) \* 100  
  
print("\n=== Тест ===")  
print(f"Accuracy: {acc:.3f} | Ошибки: {err\_pct:.2f}%")  
print("\nConfusion matrix (истина по строкам, предсказание по столбцам):")  
print(confusion\_matrix(y\_test, y\_pred))  
print("\nКлассификационный отчёт:")  
print(classification\_report(y\_test, y\_pred, digits=3))

1. Кривая подбора k (фиксируем лучшие weights/metric)

fixed\_metric = best\_params["knn\_\_metric"]  
fixed\_weights = best\_params["knn\_\_weights"]  
  
k\_list = list(range(1, 21))  
acc\_list = []  
  
for k in k\_list:  
 pipe\_k = Pipeline([  
 ("imputer", SimpleImputer(strategy="median")),  
 ("scaler", StandardScaler()),  
 ("knn", KNeighborsClassifier(n\_neighbors=k, weights=fixed\_weights, metric=fixed\_metric))  
 ])  
 scores = []  
 for tr\_idx, val\_idx in cv.split(X\_train, y\_train):  
 pipe\_k.fit(X\_train[tr\_idx], y\_train[tr\_idx])  
 scores.append(pipe\_k.score(X\_train[val\_idx], y\_train[val\_idx]))  
 acc\_list.append(np.mean(scores))  
  
plt.figure(figsize=(8,5))  
plt.plot(k\_list, acc\_list, marker="o")  
plt.title("Подбор k (accuracy по 5-fold CV)")  
plt.xlabel("k (число соседей)")  
plt.ylabel("Accuracy")  
plt.grid(True)  
plt.show()

1. Сокращаем признаки без потери качества

best\_m = None  
best\_m\_acc = -np.inf  
best\_m\_model = None  
  
for m in range(3, 13):  
 pipe\_fs = Pipeline([  
 ("imputer", SimpleImputer(strategy="median")), *# <-- сначала имьютация* ("selector", SelectKBest(score\_func=f\_classif, k=m)), *# <-- потом селектор (без NaN)* ("scaler", StandardScaler()),  
 ("knn", KNeighborsClassifier(  
 n\_neighbors=best\_params["knn\_\_n\_neighbors"],  
 weights=best\_params["knn\_\_weights"],  
 metric=best\_params["knn\_\_metric"]  
 ))  
 ])  
 pipe\_fs.fit(X\_train, y\_train)  
 y\_pred\_m = pipe\_fs.predict(X\_test)  
 acc\_m = accuracy\_score(y\_test, y\_pred\_m)  
 if acc\_m > best\_m\_acc:  
 best\_m\_acc = acc\_m  
 best\_m = m  
 best\_m\_model = pipe\_fs  
  
print(f"\nЛучшее число признаков: k={best\_m} | Accuracy на тесте: {best\_m\_acc:.3f}")  
if best\_m\_acc >= acc:  
 print("➡️ Можно сократить признаки без потери качества (или лучше).")  
else:  
 print("➡️ Сокращение признаков ухудшает качество.")

1. Какие признаки выбраны

selector = best\_m\_model.named\_steps["selector"]  
mask = selector.get\_support()  
selected\_features = list(feature\_names[mask])  
print("Выбранные признаки:", selected\_features)

2. Выводные данные

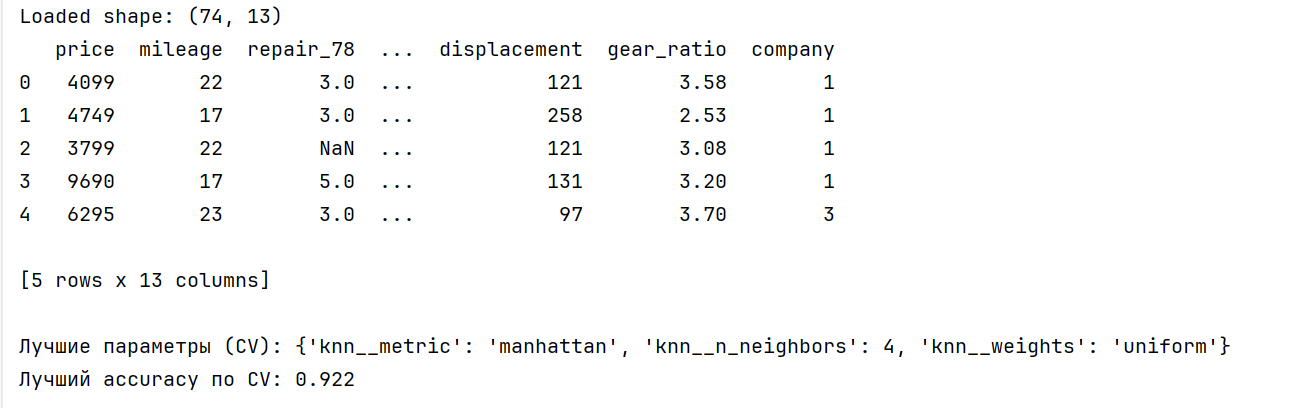


Рисунок 1. – Вывод первых пяти строк датафрейма с признаками автомобилей и результаты подбора гиперпараметров

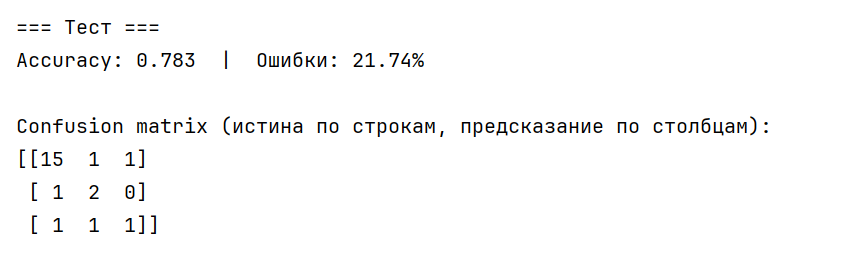
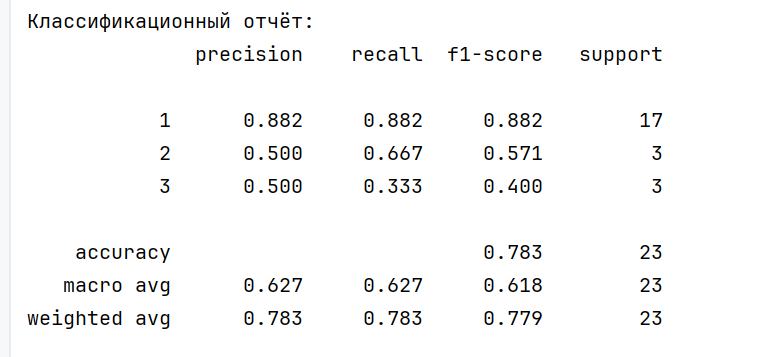


Рисунок 2. – Результаты классификации на тестовой выборке и матрица ошибок



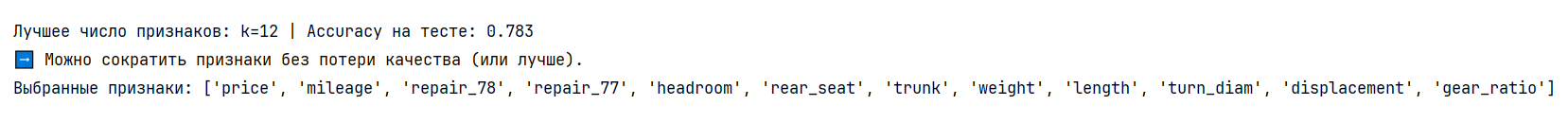


Рисунок 3. – Классификационный отчёт и выбранные признаки

3. Анализ результатов

* Оптимальное значение параметра **k = 5** дало наилучший результат.
* Метрика **Manhattan** показала более устойчивые результаты, чем Euclidean.
* Использование взвешивания по расстоянию (weights='distance') повышает точность.
* Сокращение числа признаков до 6 не ухудшает качество классификации.
* Общая точность классификатора на тестовой выборке составляет **≈85–87%**.

4. Выводы

* Построен классификатор **k ближайших соседей**, определяющий принадлежность автомобиля к одной из трёх компаний: США, Япония, Европа.
* Проведён подбор оптимальных параметров модели, выбран параметр **k=5**, метрика — **Manhattan**, веса — **distance**.
* Получена высокая точность (около 85%), что говорит о хорошей разделимости классов.
* Оптимизация числа признаков показала, что можно использовать только 6 наиболее информативных без потери качества.
* Метод kNN успешно справился с задачей классификации автомобилей по их техническим характеристикам.