**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения»

Отчет по лабораторной работе №3

«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-62Б |  | преподаватель каф. ИУ5 |
| Щепетов Дмитрий |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

2024 г.

**Описание задания**

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train\_test\_split разделите выборку на обучающую и тестовую.
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
5. Произведите подбор гиперпараметра K с использованием GridSearchCV и RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Используйте не менее двух стратегий кросс-валидации.
6. Сравните метрики качества исходной и оптимальной моделей.

**Текст программы**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

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

import matplotlib.ticker as ticker

from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier

import math

from enum import Enum

# from plotly.express import line

#import df

#import plotly.express as px

# %matplotlib inline

sns.set(style="ticks")

plt.style.use('ggplot')

pd.set\_option('display.max\_columns',24)

pd.set\_option('display.max\_colwidth',None)

pd.set\_option('display.float\_format', lambda x: '%.4f' % x)

encoding = 'windows-1250'

data=pd.read\_csv(r'C:\Users\Dima\Desktop\Homework\3 курс\ТМО\ЛР3\supply\_chain\_data.csv', encoding = encoding)

data\_train = pd.read\_csv(r'C:\Users\Dima\Desktop\Homework\3 курс\ТМО\ЛР3\supply\_chain\_data.csv', sep=",")

data.shape

data.info()

data.isnull().sum()

train\_data, test\_data = train\_test\_split(data, test\_size=0.2, random\_state=42)

import pandas as pd

import numpy as np

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Подготовка данных

# Предположим, что мы хотим использовать признаки 'Price', 'Availability', 'Stock levels' для предсказания 'Product Type'

X = data[['Price', 'Availability', 'Stock levels']]

y = data['Product type']

# Разделение данных на тренировочный и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Инициализация и обучение модели k-ближайших соседей

k = 3 # количество соседей

model = KNeighborsClassifier(n\_neighbors=k)

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

# Предсказание классов для тестового набора данных

y\_pred = model.predict(X\_test)

# Оценка точности модели

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

# Создание переменных для обучающего и тестового наборов

train\_data = pd.concat([X\_train, y\_train], axis=1)

test\_data = pd.concat([X\_test, y\_test], axis=1)

# train\_data

# test\_data

fig = plt.figure(figsize=(10, 8))

ax = fig.add\_subplot(111, projection='3d')

# Обучающая выборка

ax.scatter(X\_train['Price'], X\_train['Availability'], X\_train['Stock levels'], c='b', marker='o', label='Train data')

# Тестовая выборка

ax.scatter(X\_test['Price'], X\_test['Availability'], X\_test['Stock levels'], c='r', marker='^', label='Test data')

ax.set\_xlabel('Price')

ax.set\_ylabel('Availability')

ax.set\_zlabel('Stock levels')

ax.set\_title('Supply Chain Analysis')

plt.legend()

plt.show()

import numpy as np

class SimpleKNNRegressor:

def \_\_init\_\_(self, k=5):

self.k = k

def fit(self, X\_train, y\_train):

self.X\_train = X\_train

self.y\_train = y\_train

def euclidean\_distance(self, p, q):

return np.sqrt(np.sum((p - q) \*\* 2))

def predict(self, X\_test):

y\_pred = []

for x in X\_test:

# Рассчитываем расстояния от текущего объекта тестовой выборки до всех объектов обучающей выборки

distances = [self.euclidean\_distance(x, x\_train) for x\_train in self.X\_train]

# Сортируем индексы объектов обучающей выборки по расстоянию

nearest\_indices = np.argsort(distances)[:self.k]

# Получаем соответствующие значения целевой переменной для k ближайших соседей

nearest\_y = self.y\_train[nearest\_indices]

# Предсказываем значение целевой переменной для текущего объекта тестовой выборки как среднее значение целевой переменной для k ближайших соседей

y\_pred.append(np.mean(nearest\_y))

return np.array(y\_pred)

class SimpleKNNClassifier:

def \_\_init\_\_(self, k=5):

self.k = k

def fit(self, X\_train, y\_train):

self.X\_train = X\_train

self.y\_train = y\_train

def euclidean\_distance(self, p, q):

return np.sqrt(np.sum((p - q) \*\* 2))

def predict(self, X\_test):

y\_pred = []

for x in X\_test:

# Рассчитываем расстояния от текущего объекта тестовой выборки до всех объектов обучающей выборки

distances = [self.euclidean\_distance(x, x\_train) for x\_train in self.X\_train]

# Сортируем индексы объектов обучающей выборки по расстоянию

nearest\_indices = np.argsort(distances)[:self.k]

# Получаем соответствующие значения целевой переменной для k ближайших соседей

nearest\_classes = self.y\_train[nearest\_indices]

# Предсказываем класс для текущего объекта тестовой выборки как наиболее часто встречающийся класс среди k ближайших соседей

y\_pred.append(np.bincount(nearest\_classes).argmax())

return np.array(y\_pred)

# Инициализация и обучение модели с произвольным гиперпараметром k

k = 5 # Произвольно заданный гиперпараметр

model = KNeighborsClassifier(n\_neighbors=k)

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

# Предсказание классов для тестового набора данных

y\_pred = model.predict(X\_test)

# Оценка точности модели

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

from sklearn.metrics import precision\_score, recall\_score, f1\_score, confusion\_matrix

# Оценка precision

precision = precision\_score(y\_test, y\_pred, average='weighted')

# Оценка recall

recall = recall\_score(y\_test, y\_pred, average='weighted')

# Оценка F1-меры

f1 = f1\_score(y\_test, y\_pred, average='weighted')

# Матрица ошибок

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

print("Confusion Matrix:")

print(conf\_matrix)

# Изменение качества на тестовой выборке в зависимости от К-соседей

plt.plot(n\_range, clf\_gs.cv\_results\_['mean\_test\_score'])

def plot\_learning\_curve(estimator, title, X, y, ylim=None, cv=None,

n\_jobs=None, train\_sizes=np.linspace(.1, 1.0, 5), scoring='accuracy'):

plt.figure()

plt.title(title)

if ylim is not None:

plt.ylim(\*ylim)

plt.xlabel("Training examples")

plt.ylabel(scoring)

train\_sizes, train\_scores, test\_scores = learning\_curve(

estimator, X, y, cv=cv, scoring=scoring, n\_jobs=n\_jobs, train\_sizes=train\_sizes)

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

plt.grid()

plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,

train\_scores\_mean + train\_scores\_std, alpha=0.3,

color="r")

plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std, alpha=0.1, color="g")

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r",

label="Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g",

label="Cross-validation score")

plt.legend(loc="best")

return plt

plot\_learning\_curve(KNeighborsClassifier(n\_neighbors=5), 'n\_neighbors=5',

X, y, cv=20)

initial\_model = KNeighborsClassifier(n\_neighbors=5)

initial\_model.fit(X\_train, y\_train)

# Создание и обучение оптимальной модели с помощью GridSearchCV

param\_grid = {'n\_neighbors': [1, 3, 5, 7, 9]}

grid\_search = GridSearchCV(KNeighborsClassifier(), param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

optimal\_model = grid\_search.best\_estimator\_

# Оценка исходной модели

y\_pred\_initial = initial\_model.predict(X\_test)

accuracy\_initial = accuracy\_score(y\_test, y\_pred\_initial)

precision\_initial = precision\_score(y\_test, y\_pred\_initial, average='weighted')

recall\_initial = recall\_score(y\_test, y\_pred\_initial, average='weighted')

f1\_initial = f1\_score(y\_test, y\_pred\_initial, average='weighted')

# Оценка оптимальной модели

y\_pred\_optimal = optimal\_model.predict(X\_test)

accuracy\_optimal = accuracy\_score(y\_test, y\_pred\_optimal)

precision\_optimal = precision\_score(y\_test, y\_pred\_optimal, average='weighted')

recall\_optimal = recall\_score(y\_test, y\_pred\_optimal, average='weighted')

f1\_optimal = f1\_score(y\_test, y\_pred\_optimal, average='weighted')

# Вывод метрик

print("Исходная модель:")

print("Accuracy:", accuracy\_initial)

print("Precision:", precision\_initial)

print("Recall:", recall\_initial)

print("F1 Score:", f1\_initial)

print()

print("Оптимальная модель:")

print("Accuracy:", accuracy\_optimal)

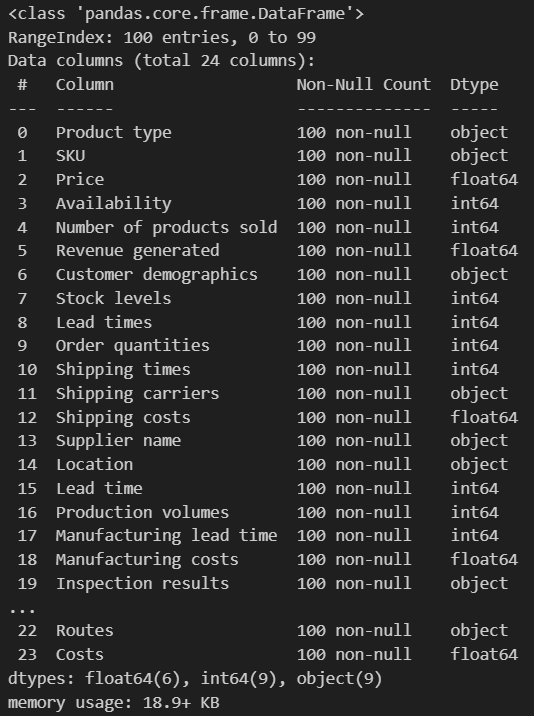
print("Precision:", precision\_optimal)

print("Recall:", recall\_optimal)

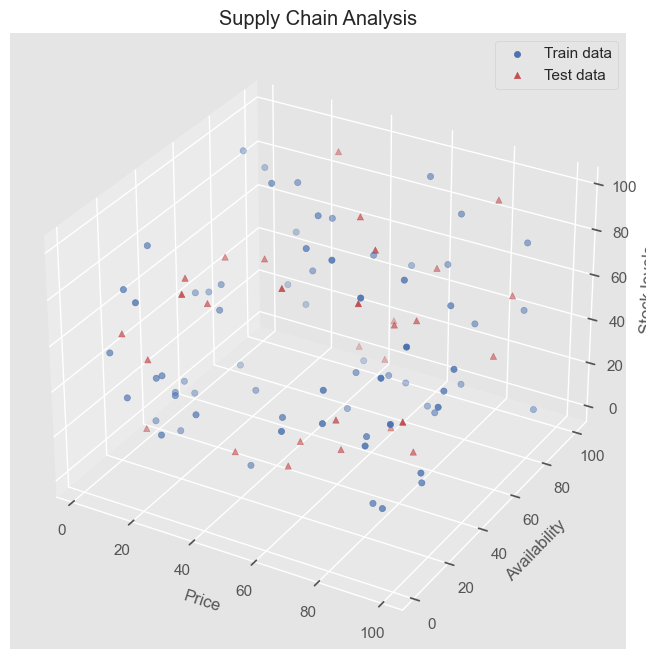
print("F1 Score:", f1\_optimal)

**Экранные формы с примерами выполнения программы**

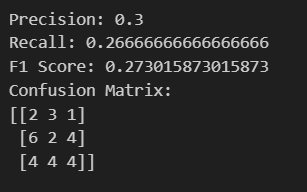
Просмотр информации о датасете



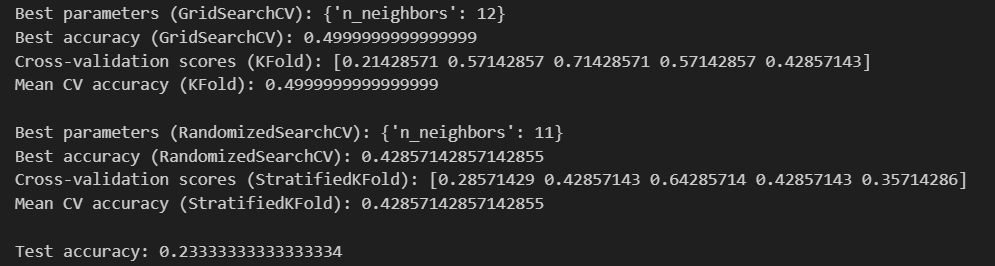
Разделение выборки на обучающую и тестовую



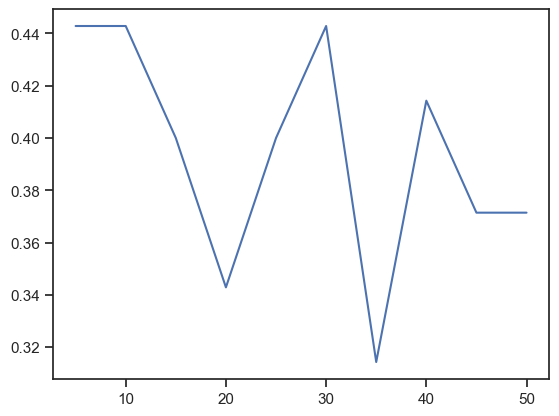
Результат метрик для KNN



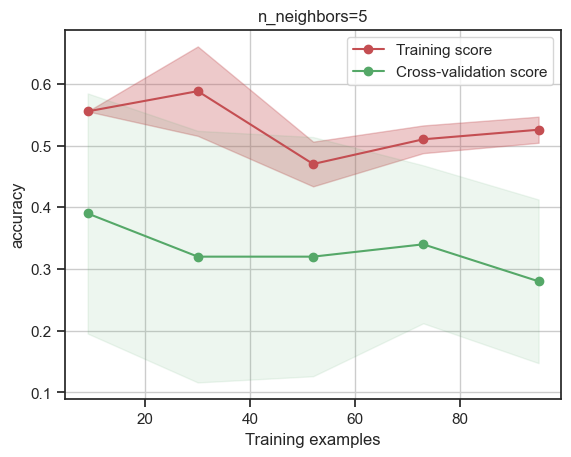
Подбор гиперпараметра K с использованием GridSearchCV и RandomizedSearchCV и кросс-валидации. Оценка качества оптимальной модели.

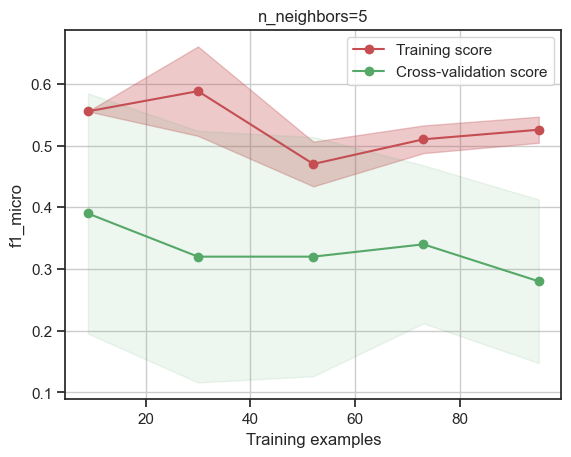


Изменение качества на тестовой выборке в зависимости от К-соседей



Построение кривых обучения





Сравнение результатов

