**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 №6

«Ансамбли моделей машинного обучения. Часть 2»

|  |  |  |
| --- | --- | --- |
| 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 регресии.
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 следующие ансамблевые модели:
   * одну из моделей группы стекинга.
   * модель многослойного персептрона. По желанию, вместо библиотеки scikit-learn возможно использование библиотек TensorFlow, PyTorch или других аналогичных библиотек.
   * двумя методами на выбор из семейства МГУА (один из линейных методов COMBI / MULTI + один из нелинейных методов MIA / RIA) с использованием библиотеки gmdh.
5. Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных 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

import os

# from plotly.express import line

#import data

#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\ЛР6\melb\_data.csv', encoding = encoding)

data.shape

data.info()

data = data.drop(columns='Suburb')

data = data.drop(columns='Address')

data = data.drop(columns='Type')

data = data.drop(columns='Method')

data = data.drop(columns='SellerG')

data = data.drop(columns='Date')

data = data.drop(columns='CouncilArea')

data = data.drop(columns='Regionname')

# Удаление строк с пропущенными значениями в столбцах "Method" и "Car"

# data.dropna(subset=["Method", "Car"], inplace=True)

data.dropna(subset=["Car"], inplace=True)

# Заполнение пропущенных значений в столбцах "BuildingArea", "YearBuilt" и "CouncilArea"

# Для числовых столбцов можно использовать медиану

median\_BuildingArea = data["BuildingArea"].median()

median\_YearBuilt = data["YearBuilt"].median()

# Для категориального столбца можно использовать наиболее часто встречающееся значение

# mode\_CouncilArea = data["CouncilArea"].mode()[0]

data["BuildingArea"].fillna(median\_BuildingArea, inplace=True)

data["YearBuilt"].fillna(median\_YearBuilt, inplace=True)

# data["CouncilArea"].fillna(mode\_CouncilArea, inplace=True)

# Заполнение пропущенных значений в столбцах "BuildingArea", "YearBuilt" и "CouncilArea"

# Для числовых столбцов можно использовать медиану

median\_BuildingArea = data["BuildingArea"].median()

median\_YearBuilt = data["YearBuilt"].median()

# Для категориального столбца можно использовать наиболее часто встречающееся значение

# mode\_CouncilArea = data["CouncilArea"].mode()[0]

data["BuildingArea"].fillna(median\_BuildingArea, inplace=True)

data["YearBuilt"].fillna(median\_YearBuilt, inplace=True)

# data["CouncilArea"].fillna(mode\_CouncilArea, inplace=True)

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

from sklearn.preprocessing import LabelEncoder

X = data.drop(columns=["Price"]) # Признаки (все столбцы кроме "Price")

y = data["Price"] # Целевая переменная (столбец "Price")

# Разделение выборки на обучающую и тестовую

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Вывод размеров обучающей и тестовой выборок

print("Размер обучающей выборки:", X\_train.shape[0])

print("Размер тестовой выборки:", X\_test.shape[0])

def val\_mae(model):

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

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

result = mean\_absolute\_error(y\_test, y\_pred)

print(model)

print('MAE={}'.format(result))

# Точность на отдельных моделях

for model in [

LinearRegression(),

DecisionTreeRegressor(),

RandomForestRegressor(n\_estimators=50)

]:

val\_mae(model)

print('==========================')

print()

X1, y1 = data.iloc[:, :-1], data.iloc[:, -1]

y1 = pd.Series(LabelEncoder().fit\_transform(y1))

X1\_train, X1\_test, y1\_train, y1\_test = train\_test\_split(X1, y1, random\_state=0)

print(data)

X2, y2 = load\_diabetes(return\_X\_y=True, as\_frame=True)

X2\_train, X2\_test, y2\_train, y2\_test = train\_test\_split(X2, y2, random\_state=0)

print(X2, y2, sep='\n')

reg\_estimators = [RandomForestRegressor(random\_state=0),

GradientBoostingRegressor(random\_state=0)]

stacking\_reg = Stacking(estimators=reg\_estimators, final\_estimator=RidgeCV())

stacking\_reg\_pred\_res = stacking\_reg.fit\_predict(X2\_train, y2\_train, X2\_test)

stacking\_mape = mean\_absolute\_percentage\_error(stacking\_reg\_pred\_res, y2\_test)

print(f'stacking\_regressor\_mape {stacking\_mape}')

print(stacking\_reg\_pred\_res, '', sep='\n')

sk\_reg\_estimators = [('rfr', RandomForestRegressor(random\_state=0)),

('gbr', GradientBoostingRegressor(random\_state=0))]

sk\_stacking\_reg = StackingRegressor(estimators=sk\_reg\_estimators)

sk\_stacking\_reg.fit(X2\_train, y2\_train)

sk\_stacking\_reg\_pred\_res = sk\_stacking\_reg.predict(X2\_test)

sk\_stacking\_mape = mean\_absolute\_percentage\_error(sk\_stacking\_reg\_pred\_res, y2\_test)

print(f'sk\_stacking\_regressor\_mape {sk\_stacking\_mape}')

print(sk\_stacking\_reg\_pred\_res)

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

sns.set(font\_scale=1)

# pd.options.display.mpl\_style = 'default'

labels = []

values = []

for col in null\_columns:

labels.append(col)

values.append(data[col].isnull().sum())

ind = np.arange(len(labels))

width=0.6

fig, ax = plt.subplots(figsize=(6,5))

rects = ax.barh(ind, np.array(values), color='purple')

ax.set\_yticks(ind+((width)/2.))

ax.set\_yticklabels(labels, rotation='horizontal')

ax.set\_xlabel("Количество пропусков")

ax.set\_ylabel("Столбцы")

ax.set\_title("Пропуски")

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

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPRegressor

# Подготовка данных

# Заполнение пропущенных значений, кодирование категориальных признаков и т.д.

# Выделение признаков и целевой переменной

X = data.drop(columns=['Price']) # Признаки

y = data['Price'] # Целевая переменная

# Разделение данных на обучающий и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Масштабирование признаков

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Обучение многослойного персептрона

mlp = MLPRegressor(hidden\_layer\_sizes=(100, 50), activation='relu', solver='adam', max\_iter=500)

mlp.fit(X\_train\_scaled, y\_train)

# Оценка модели

train\_score = mlp.score(X\_train\_scaled, y\_train)

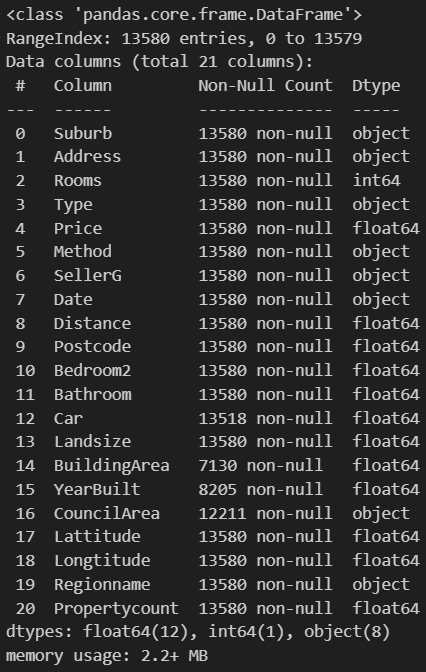
test\_score = mlp.score(X\_test\_scaled, y\_test)

print(f'Train R^2 Score: {train\_score}')

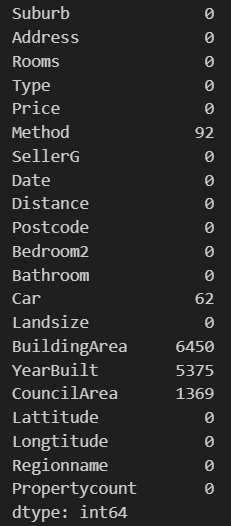
print(f'Test R^2 Score: {test\_score}')

**Экранные формы с примерами выполнения программы**

Просмотр информации о датасете



Проверка нулевых значений



Разделение данных на тестовую и тренировочную выборки



Точность на отдельных моделях

