

2023-2024 学年第一学期计算机 23C1-C2 班人工智能模块
《机器学习》期末大作业

班级： 计算机 23C2

学号： 2023031459

姓名： 赵新超

2024 年 11 月

利用支持向量机预测学生成绩的研究

一、 利用机器学习算法解决问题的过程

1. 实验背景介绍

1.1 问题研究

学生的学业成绩受到多种因素的影响，如学习习惯、家庭环境和身体情况等。随着机器学习的广泛应用，预测学生成绩并分析影响因素变得更加可行和准确。特别是支持向量机回归（SVR）在处理数据中的复杂关系时表现不错。

学生的成绩是衡量教育效果的直接指标，如何通过已有数据预测成绩并识别关键因素，对教育管理和教学改进具有重要意义。本文旨在利用 SVR 算法，建立一个预测学生成绩的模型。通过 SVR 模型预测学生的考试成绩（Exam_Score），并对模型表现进行分析。

1.2 数据集来源及特征描述

该数据集来源于 Kaggle 平台，全面概述了影响学生考试成绩的各种因素，包括有关学习习惯、出勤率、父母参与和影响学习的其他方面的信息。数据集包含 6607 条样本和 17 个特征，涵盖了学生的学习习惯、家庭因素及身体状况等信息。目标特征为学生的考试成绩（Exam_Score）。数据中包含 7 个数值型特征和 10 个分类特征。

2. 数据预处理

2.1 缺失值处理

缺失值会导致数据分析结果的不准确，因此需要在数据预处理阶段进行处理。一般来说，是使用 `isnull()` 方法识别缺失值的存在，并统计每个特征的缺失情况。然后根据缺失值的数量和特征的重要性，再决定删除还是补全。本数据集中的 `Teacher_Quality`、`Parental_Education_Level` 和 `Distance_from_Home` 也有缺失值，但因为缺失值较少，所以选择删除含缺失值的样本。最后在验证处理的结果，确保缺失值已被成功移除。

2.2 异常值处理

异常值会对模型训练产生不利的影响，因此需要对数据集进行检查和处理。可以使用 `describe()` 方法查看各特征的基本统计信息，这样就可能会发现异常值。而对于发现的异常值，可以通过过滤方法进行删除。最后还可以再次验证数据，确认数据集中没有异常值。

2.3 类别变量的处理

机器学习模型通常只能处理数值型数据，所以需要将分类特征转换为数值形式。分类特征是指那些具有有限取值的特征，比如性别、教育水平或分类标签等。对数据进行编码常用的有独热编码，使用 `get_dummies()` 方法将分类特征转换为多个新的二进制特征。处理分类特征后，数据集将包含更多的数值特征，

2.4 数据标准化

数据标准化是指将不同范围的数值特征转换为统一的标准，因为由于不同特征的数值范围可能相差很大，可能导致某些特征对模型的训练产生过大的影响，而其他特征则可能被忽略。通过标准化，模型能够更有效地学习到数据的特征，这对于使用基于距离的算法尤其重要，如 SVR。

3. 支持向量机回归（SVR）算法介绍

4. 模型选择与训练

4.1 模型选择

4.2 模型训练与评估

5. 支持向量机回归（SVR）模型的优化

5.1 核函数选择

5.2 超参数选择

5.3 网格搜索

5.4 优化结果

6. 参考文献

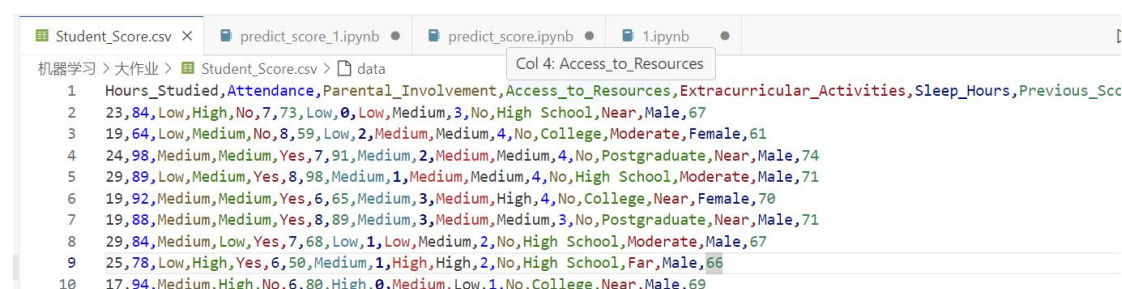
- [1]金秀玲,卓艳如. 基于遗传算法和支持向量机模型的高考成绩预测 [J]. 河南工程学院学报(自然科学版), 2020, 32 (02): 62-65. DOI:10.16203/j.cnki.41-1397/n.2020.02.014.
- [2]陆丛林. 支持向量机在高考成绩预测分析中的应用[D]. 苏州大学, 2015.

二、 选择该算法的原因

三、 实现过程及成果展示

1. 数据展示

数据集来源于 Kaggle 平台，它涉及了多种影响学生考试成绩的因素。部分数据展示如下。



	Hours_Studied	Attendance	Parental_Involvement	Access_to_Resources	Extracurricular_Activities	Sleep_Hours	Previous_Scores	Motivation_Level	Tutoring_Sessions	Family_Income	Teacher_Quality	Physical_Activity	Learning_Disabilities	Parental_Education_Level	Distance_from_Home	Gender	Exam_Score
1	23	84	Low	High	No	7	73	Low	0	Low	Medium	3	No	High School	Near	Male	67
2	19	64	Low	Medium	No	8	59	Low	2	Medium	Medium	4	No	College	Moderate	Female	61
3	24	98	Medium	Medium	Yes	7	91	Medium	2	Medium	Medium	4	No	Postgraduate	Near	Male	74
4	29	89	Low	Medium	Yes	8	98	Medium	1	Medium	Medium	4	No	High School	Moderate	Male	71
5	19	92	Medium	Medium	Yes	6	65	Medium	3	Medium	High	4	No	College	Near	Female	70
6	19	88	Medium	Medium	Yes	8	89	Medium	3	Medium	Medium	3	No	Postgraduate	Near	Male	71
7	29	84	Medium	Low	Yes	7	68	Low	1	Low	Medium	2	No	High School	Moderate	Male	67
8	25	78	Low	High	Yes	6	50	Medium	1	High	High	2	No	High School	Far	Male	66
9	17	94	Medium	High	No	6	80	High	0	Medium	Low	1	No	College	Near	Male	69

数据共有 17 个特征，分别是 Hours_Studied（每周学习时间）、Attendance（上课出席率）、Parental_Involvement（家长对学生学习的参与程度）、Access_to_Resources（教育资源的可用性）、Extracurricular_Activities（课外活动）、Sleep_Hours（睡眠小时数）、Previous_Scores（以前考试的分数）、Motivation_Level（学生的积极性水平）、Tutoring_Sessions（参加辅导课程次数）、Family_Income（家庭收入）、Teacher_Quality（教师质量）、Physical_Activity（身体活动小时数）、Learning_Disabilities（是否存在学习障碍）、Parental_Education_Level（父母的教育水平）、Distance_from_Home（家到学校的距离）、Gender（性别）和 Exam_Score（考试成绩）。

```
import pandas as pd
df = pd.read_csv('Student_Score.csv')
df.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6607 entries, 0 to 6606
Data columns (total 17 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Hours_Studied                        6607 non-null  int64
1   Attendance                          6607 non-null  int64
2   Parental_Involvement                6607 non-null  object
3   Access_to_Resources                 6607 non-null  object
4   Extracurricular_Activities          6607 non-null  object
5   Sleep_Hours                        6607 non-null  int64
6   Previous_Scores                     6607 non-null  int64
7   Motivation_Level                    6607 non-null  object
8   Tutoring_Sessions                   6607 non-null  int64
9   Family_Income                       6607 non-null  object
10  Teacher_Quality                     6529 non-null  object
11  Physical_Activity                   6607 non-null  int64
12  Learning_Disabilities               6607 non-null  object
13  Parental_Education_Level            6517 non-null  object
14  Distance_from_Home                  6540 non-null  object
15  Gender                              6607 non-null  object
16  Exam_Score                          6607 non-null  int64
dtypes: int64(7), object(10)
memory usage: 877.6+ KB
```

通过上图的结果可以看到，数据集包括 6607 条样本和 17 个特征，其中 Exam_Score 是要预测的目标特征。数值型特征共有 7 个，数据类型是 int64；而有 10 个数据类型为 object 的特征，在后续处理过程中需要这些特征进行独热编码。有三列存在缺失值，Teacher_Quality 缺失 78 个值，Parental_Education_Level 缺失 90 个值，Distance_from_Home 缺失 67 个值。因为缺失值的可能会影响模型的性能，所以需要在数据预处理阶段进行处理，删除含缺失值的样本。

```
df.describe()
```

✓ 0.0s

	Hours_Studied	Attendance	Sleep_Hours	Previous_Scores	Tutoring_Sessions	Physical_Activity	Exam_Score
count	6607.000000	6607.000000	6607.000000	6607.000000	6607.000000	6607.000000	6607.000000
mean	19.975329	79.977448	7.02906	75.070531	1.493719	2.967610	67.235659
std	5.990594	11.547475	1.46812	14.399784	1.230570	1.031231	3.890456
min	1.000000	60.000000	4.00000	50.000000	0.000000	0.000000	55.000000
25%	16.000000	70.000000	6.00000	63.000000	1.000000	2.000000	65.000000
50%	20.000000	80.000000	7.00000	75.000000	1.000000	3.000000	67.000000
75%	24.000000	90.000000	8.00000	88.000000	2.000000	4.000000	69.000000
max	44.000000	100.000000	10.00000	100.000000	8.000000	6.000000	101.000000

根据上图 df.describe() 的统计结果，可以了解到特征的统计信息。这其中考试成绩最高达到了 101，可能是异常值，在数据预处理的时候可以进行处理。

2. 数据预处理

通过 `dropna()` 方法删除所有包含缺失值的行，代码和结果如下。

```
df_clean = df.dropna()
✓ 0.0s

df_clean.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
Int64Index: 6378 entries, 0 to 6606
Data columns (total 17 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Hours_Studied                        6378 non-null   int64
1   Attendance                          6378 non-null   int64
2   Parental_Involvement                6378 non-null   object
3   Access_to_Resources                 6378 non-null   object
4   Extracurricular_Activities          6378 non-null   object
5   Sleep_Hours                        6378 non-null   int64
6   Previous_Scores                    6378 non-null   int64
7   Motivation_Level                   6378 non-null   object
8   Tutoring_Sessions                  6378 non-null   int64
9   Family_Income                      6378 non-null   object
10  Teacher_Quality                    6378 non-null   object
11  Physical_Activity                  6378 non-null   int64
12  Learning_Disabilities              6378 non-null   object
13  Parental_Education_Level           6378 non-null   object
14  Distance_from_Home                 6378 non-null   object
15  Gender                             6378 non-null   object
16  Exam_Score                         6378 non-null   int64
dtypes: int64(7), object(10)
memory usage: 896.9+ KB
```

可以看到，现在已经没有缺失值了。

接下来将考试成绩大于 100 的异常值删除。

```
df_clean = df_clean[df_clean['Exam_Score'] <= 100]
df_clean['Exam_Score'].describe()
✓ 0.0s

count    6377.000000
mean      67.246825
std        3.891637
min       55.000000
25%       65.000000
50%       67.000000
75%       69.000000
max      100.000000
Name: Exam_Score, dtype: float64
```

从结果可以看到数据减少了一条，说明 101 的值只有一条。

然后就可以开始使用 `get_dummies()` 方法，也就是独热编码将分类特征转换

为数值型特征

```
transform_features = [
    'Parental_Involvement', 'Access_to_Resources',
    'Extracurricular_Activities', 'Motivation_Level',
    'Family_Income', 'Teacher_Quality',
    'Learning_Disabilities', 'Parental_Education_Level',
    'Distance_from_Home', 'Gender'
]
df_final = pd.get_dummies(df_clean, columns=transform_features, drop_first=True)
df_final.info()
```

✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
Int64Index: 6377 entries, 0 to 6606
Data columns (total 24 columns):
Column Non-Null Count Dtype

0 Hours_Studied 6377 non-null int64
1 Attendance 6377 non-null int64
2 Sleep_Hours 6377 non-null int64
3 Previous_Scores 6377 non-null int64
4 Tutoring_Sessions 6377 non-null int64
5 Physical_Activity 6377 non-null int64
6 Exam_Score 6377 non-null int64
7 Parental_Involvement_Low 6377 non-null uint8
8 Parental_Involvement_Medium 6377 non-null uint8
9 Access_to_Resources_Low 6377 non-null uint8
10 Access_to_Resources_Medium 6377 non-null uint8
11 Extracurricular_Activities_Yes 6377 non-null uint8
12 Motivation_Level_Low 6377 non-null uint8
13 Motivation_Level_Medium 6377 non-null uint8
14 Family_Income_Low 6377 non-null uint8
15 Family_Income_Medium 6377 non-null uint8
16 Teacher_Quality_Low 6377 non-null uint8
17 Teacher_Quality_Medium 6377 non-null uint8
18 Learning_Disabilities_Yes 6377 non-null uint8
19 Parental_Education_Level_High_School 6377 non-null uint8
...
22 Distance_from_Home_Near 6377 non-null uint8
23 Gender_Male 6377 non-null uint8
dtypes: int64(7), uint8(17)
memory usage: 504.4 KB

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output settings...

可以看到特征值变成了 24 个，包括了 7 个原始的数值型特征，还有 17 个独热编码后生成的分类特征。

```
df_final.head()
```

✓ 0.0s Python

	resources_Low	...	Family_Income_Low	Family_Income_Medium	Teacher_Quality_Low	Teacher_Quality_Medium	Learning_Disabilities_Yes	Parental
	0	...	1	0	0	1	0	
	0	...	0	1	0	1	0	
	0	...	0	1	0	1	0	
	0	...	0	1	0	1	0	
	0	...	0	1	0	0	0	

使用 StandardScaler 对数值特征进行标准化处理，将特征转换为均值为 0，标准差为 1 的标准正态分布，然后查看标准化后的特征统计信息。结果如下。


```

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_final_scaler = df_final.copy(deep=True)
scaler_features = ['Hours_Studied', 'Attendance', 'Sleep_Hours',
                  'Previous_Scores', 'Tutoring_Sessions',
                  'Physical_Activity']
# 进行标准化
df_final_scaler[scaler_features] = scaler.fit_transform(df_final_scaler[scaler_features])
df_final_scaler[scaler_features].describe()

```

✓ 0.0s

	Hours_Studied	Attendance	Sleep_Hours	Previous_Scores	Tutoring_Sessions	Physical_Activity
count	6.377000e+03	6.377000e+03	6.377000e+03	6.377000e+03	6.377000e+03	6.377000e+03
mean	-3.838339e-16	1.565838e-16	3.443659e-17	1.556436e-17	4.631007e-17	-1.497156e-15
std	1.000078e+00	1.000078e+00	1.000078e+00	1.000078e+00	1.000078e+00	1.000078e+00
min	-3.170693e+00	-1.733384e+00	-2.067559e+00	-1.740675e+00	-1.212085e+00	-2.889144e+00
25%	-6.643495e-01	-8.674728e-01	-7.051386e-01	-8.378121e-01	-4.011884e-01	-9.453691e-01
50%	4.008892e-03	-1.561546e-03	-2.392835e-02	-4.399908e-03	-4.011884e-01	2.651849e-02
75%	6.723672e-01	8.643497e-01	6.572819e-01	8.984633e-01	4.097081e-01	9.984061e-01
max	4.014159e+00	1.730261e+00	2.019702e+00	1.731876e+00	5.275088e+00	2.942181e+00

3. 模型选择

Exam_Score 表示的是学生的考试成绩，这是一个连续数值，因此需要预测的是一个具体数值，而不是分类标签。而回归模型的目标是预测一个连续的变量，和这个问题相符合。

首先我选择了几个回归模型，分别是线性回归、决策树回归、随机森林回归、支持向量机回归（SVR）、K 最近邻回归和多层感知器回归（MLP）。

先使用没有标准化的数据，然后用这几种回归模型，依次进行训练和预测，最后计算 R^2 得分，比较模型的性能。代码如下。

```

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score
# 划分数据集
X = df_final.drop('Exam_Score', axis=1)
y = df_final['Exam_Score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
models = {
    "线性回归": LinearRegression(),
    "决策树回归": DecisionTreeRegressor(random_state=42),
    "随机森林回归": RandomForestRegressor(random_state=42),
    "支持向量机回归(SVR)": SVR(),
    "K最近邻回归": KNeighborsRegressor(),
    "多层感知器回归(MLP)": MLPRegressor(random_state=42, max_iter=500)
}

```



```

# 评估模型
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    print(f"{model_name} - R^2 Score: {r2:.4f}")

```

运行结果如下。

```

线性回归 - R^2 Score: 0.7555
决策树回归 - R^2 Score: -0.0373
随机森林回归 - R^2 Score: 0.6438
支持向量机回归 (SVR) - R^2 Score: 0.6415
K最近邻回归 - R^2 Score: 0.5488
多层感知器回归 (MLP) - R^2 Score: 0.7393

```

接下来使用经过标准化的数据，代码和运行结果如下。

```

# 划分数据集
X_scaler = df_final_scaler.drop('Exam_Score', axis=1)
y = df_final_scaler['Exam_Score']
X_train_scaler, X_test_scaler, y_train_scaler, y_test_scaler = train_test_split(
    X_scaler, y, test_size=0.3, random_state=42)
models = {
    "线性回归": LinearRegression(),
    "决策树回归": DecisionTreeRegressor(random_state=42),
    "随机森林回归": RandomForestRegressor(random_state=42),
    "支持向量机回归(SVR)": SVR(),
    "K最近邻回归": KNeighborsRegressor(),
    "多层感知器回归(MLP)": MLPRegressor(random_state=42, max_iter=500)
}
# 评估模型
for model_name, model in models.items():
    model.fit(X_train_scaler, y_train_scaler)
    y_pred_1 = model.predict(X_test_scaler)
    r2_1 = r2_score(y_test_scaler, y_pred_1)
    print(f"{model_name} - R^2 Score: {r2_1:.4f}")

```

✓ 11.4s

```

线性回归 - R^2 Score: 0.7555
决策树回归 - R^2 Score: -0.0353
随机森林回归 - R^2 Score: 0.6443
支持向量机回归 (SVR) - R^2 Score: 0.7452
K最近邻回归 - R^2 Score: 0.5403
多层感知器回归 (MLP) - R^2 Score: 0.7216

```

从结果上看线性回归的 R^2 得分始终保持最高，而且在是否标准化的情况下得分都稳定在 0.7555，表明线性回归模型在这个数据集上效果较好。支持向量机回归 (SVR) 在标准化数据后 R^2 得分提高，SVR 提高到了 0.7452，已经非常接近线性回归的得分。

4. 算法模型调整

4.1 对多项式核函数（poly）进行调整

通过网格搜索来寻找最优参数组合，并绘制不同参数组合下的 R^2 得分的折线图。

首先，定义参数范围，指定参数 C 和 γ 的值， C 值的设置包括 0.1、1 和 10， γ 值的设置包括 0.001、0.01 和 0.1。并设置使用的核函数为 'poly'。接下来，使用 GridSearchCV，传入 SVR 模型、参数范围、交叉验证次数、评分标准。最后，获取每个参数组合并绘制折线图。绘制的折线图将展示不同 C 和 γ 值下的 R^2 得分。代码如下。

```
import matplotlib.pyplot as plt
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
plt.rcParams['font.sans-serif'] = ['SimHei']
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [0.001, 0.01, 0.1],
    'kernel': ['poly']
}

svr = SVR()
grid_search = GridSearchCV(svr, param_grid, cv=5, scoring='r2', verbose=3)
grid_search.fit(X_train_scaler, y_train_scaler)

print('模型 GridSearch 交叉验证最高分为: {:.3f}'.format(grid_search.score(X_test_scaler, y_test_scaler)))
print('GridSearch 交叉验证最佳参数设置: {}'.format(grid_search.best_params_))

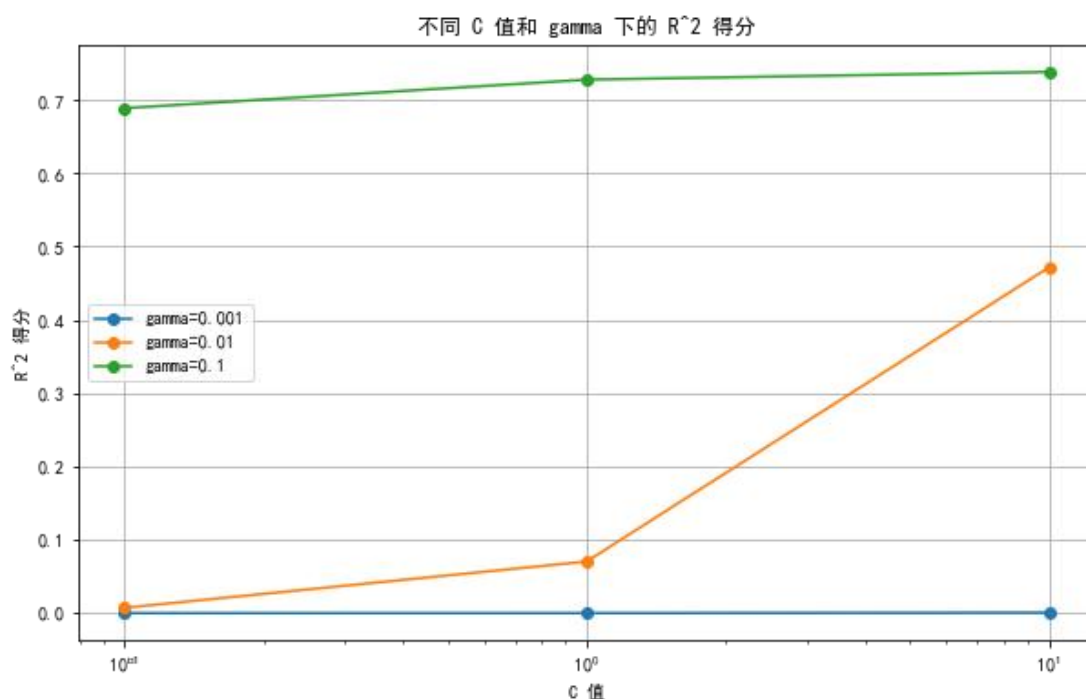
C_values = param_grid['C']
test_r2_scores = {}
for gamma in param_grid['gamma']:
    for C in C_values:
        svr_model = SVR(C=C, gamma=gamma, kernel='poly')
        svr_model.fit(X_train_scaler, y_train_scaler)
        y_pred_test = svr_model.predict(X_test_scaler)
        test_r2_score = r2_score(y_test_scaler, y_pred_test)
        test_r2_scores[gamma].append(test_r2_score)
```

```
# 折线图
plt.figure(figsize=(10, 6))
for gamma in param_grid['gamma']:
    plt.plot(C_values, test_r2_scores[gamma], marker='o', label=f'gamma={gamma}')
plt.xlabel('C 值')
plt.ylabel('R^2 得分')
plt.title('不同 C 值和 gamma 下的 R^2 得分')
plt.legend()
plt.xscale('log')
plt.grid()
plt.show()
```

✓ 4m 11.2s

运行结果如下。在使用多项式核函数进行支持向量回归的模型优化时，通过网格搜索和交叉验证的结果，模型的最佳得分为 0.739，最佳的参数配置 C 值为 10， γ 值为 0.1。通过图表可以看到在 γ 值为 0.001 时，模型的得分为 0，且此时 C 值的变化对得分影响不大；在 γ 值为 0.01 时，随着 C 值的增大，

模型的得分也在上升；在 γ 值为 0.1 时，C 值对模型的得分影响不是很大。



模型 GridSearch 交叉验证最高分为: 0.739

GridSearch 交叉验证最佳参数设置: {'C': 10, 'gamma': 0.1, 'kernel': 'poly'}

当我将 γ 值调成 1，C 值为 0.1 时，可以看到得分并没有提高，但所需要的时间变得更长了。当 γ 值为 1，C 值为 1 时，这时训练一个子集的时间达到了 128 分钟。这个时间太长了，且得分并没有明显的提高。

```
[CV 1/5] END .....C=0.1, gamma=1, kernel=poly;; score=0.509 total time= 9.7min
[CV 2/5] END .....C=0.1, gamma=1, kernel=poly;; score=0.731 total time= 9.4min
[CV 3/5] END .....C=0.1, gamma=1, kernel=poly;; score=0.636 total time= 8.2min
[CV 4/5] END .....C=0.1, gamma=1, kernel=poly;; score=0.567 total time= 9.2min
[CV 5/5] END .....C=0.1, gamma=1, kernel=poly;; score=0.733 total time= 9.0min
[CV 1/5] END .....C=1, gamma=1, kernel=poly;; score=0.482 total time=128.7min
```

4.2 对线性核函数和 RBF 核函数进行调整

定义参数范围，指定参数 C 和 γ 的值，C 值的设置包括 0.1、1、10、20 和 50， γ 值的设置包括 0.005、0.01、0.1 和 1。并设置使用的核函数为线性核函数和 RBF 核函数。接下来，使用 GridSearchCV，设置 SVR 模型、参数范围、交叉验证的次数和评分标准。最后使用最佳模型在测试集上进行预测，并计算了测试集的 R^2 得分得到交叉验证的最高得分以及最佳参数设置。代码如下。

```

import matplotlib.pyplot as plt
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

plt.rcParams['font.sans-serif'] = ['SimHei']

param_grid = {
    'C': [0.1, 1, 10, 20, 50],
    'gamma': [0.005, 0.01, 0.1, 1],
    'kernel': ['linear', 'rbf']
}

svr = SVR()
grid_search = GridSearchCV(svr, param_grid, cv=5, scoring='r2', verbose=3)
grid_search.fit(X_train_scaler, y_train_scaler)

print('模型 GridSearch 交叉验证最高分为: {:.3f}'.format(grid_search.score(X_test_scaler, y_test_scaler)))
print('GridSearch 交叉验证最佳参数设置: {}'.format(grid_search.best_params_))

```

从结果看，使用网格搜索和交叉验证后得到模型的最佳得分为 0.758，最佳的参数配置 C 值为 50，gamma 值为 0.005，核函数为线性核函数。

```

[CV 2/5] END .....C=50, gamma=1, kernel=rbf, score=0.422 total time= 1.12s
模型 GridSearch 交叉验证最高分为: 0.758
GridSearch 交叉验证最佳参数设置: {'C': 50, 'gamma': 0.005, 'kernel': 'linear'}

```

接下来，计算了每个参数组合在测试集上的 R^2 得分，并绘制图表，两个核函数分开展现，代码如下。

```

results = grid_search.cv_results_
C_values = param_grid['C']
gamma_values = param_grid['gamma']
test_scores_matrix = []

for i in range(len(C_values)):
    for j in range(len(gamma_values)):
        for kernel in param_grid['kernel']:
            svr_temp = SVR(C=C_values[i], gamma=gamma_values[j], kernel=kernel)
            svr_temp.fit(X_train_scaler, y_train_scaler)
            y_test_pred = svr_temp.predict(X_test_scaler)
            test_score = r2_score(y_test_scaler, y_test_pred)
            test_scores_matrix.append((C_values[i], gamma_values[j], kernel, test_score))

test_scores_dict = {kernel: {gamma: [] for gamma in gamma_values} for kernel in param_grid['kernel']}

for C, gamma, kernel, score in test_scores_matrix:
    test_scores_dict[kernel][gamma].append(score)

plt.figure(figsize=(12, 10))
# 线性核函数
plt.subplot(2, 1, 1)
for gamma in gamma_values:
    plt.plot(C_values, test_scores_dict['linear'][gamma], marker='o', label=f'gamma={gamma}')
plt.xlabel('C 值')
plt.ylabel('R^2 得分')
plt.title('线性核函数下不同 C 值和 gamma 的 R^2 得分')
plt.legend()
plt.xscale('log')
plt.grid()

```

```

# RBF
plt.subplot(2, 1, 2)
for gamma in gamma_values:
    plt.plot(C_values, test_scores_dict['rbf'][gamma], marker='o', label=f'gamma={gamma}')
plt.xlabel('C 值')
plt.ylabel('R^2 得分')
plt.title('RBF 核函数下不同 C 值和 gamma 的 R^2 得分')
plt.legend()
plt.xscale('log')
plt.grid()

plt.tight_layout()
plt.show()

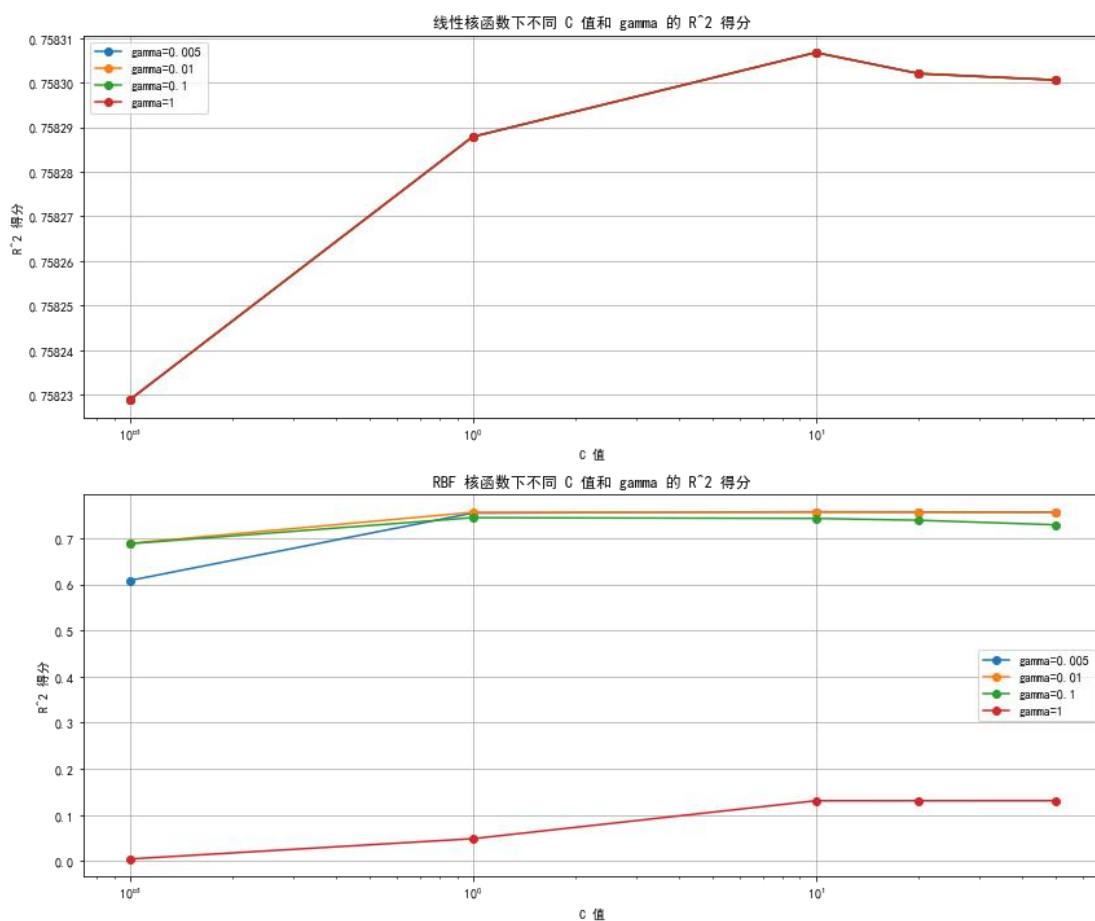
```

✓ 6m 12.9s

从结果图中可以看到：

在线性核函数下，模型的得分和 gamma 值无关，而随着 C 值的增大，模型的得分有微小的提高。

在 RBF 核函数下，当 gamma 值增加到 1 时，得分下降到了 0-0.1 左右；gamma 值小的时候，C 值的变化对模型预测的得分有一定的影响，但随着 C 值的增大，变化就不明显了。



5. 可视化界面

5.1 数据预处理

在开始之前按照之前处理的方式对数据进行预处理。

```
import tkinter as tk
from tkinter import ttk, messagebox
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# 数据预处理
df = pd.read_csv('Student_Score.csv')
df_clean = df.dropna()
df_clean = df_clean[df_clean['Exam_Score'] <= 100]
transform_features = [
    'Parental_Involvement', 'Access_to_Resources',
    'Extracurricular_Activities', 'Motivation_Level',
    'Family_Income', 'Teacher_Quality',
    'Learning_Disabilities', 'Parental_Education_Level',
    'Distance_from_Home', 'Gender'
]

# 独热编码
df_final = pd.get_dummies(df_clean, columns=transform_features, drop_first=True)

# 标准化
scaler = StandardScaler()
scaler_features = ['Hours_Studied', 'Attendance', 'Sleep_Hours',
                  'Previous_Scores', 'Tutoring_Sessions',
                  'Physical_Activity']
df_final[scaler_features] = scaler.fit_transform(df_final[scaler_features])
```

5.2 登录界面

通过输入用户名和密码进行登录学生成绩预测系统。代码和界面如下图所示。

```
def login():
    username = username_entry.get()
    password = password_entry.get()

    if username == "zxc" and password == "123":
        messagebox.showinfo("登录成功", "欢迎来到预测系统!")
        login_frame.pack_forget()
        main_menu()
    else:
        messagebox.showerror("登录失败", "用户名或密码错误。")

login_frame = tk.Frame(window)
login_frame.pack(pady=20)
login_frame.place(relx=0.5, y=200, anchor="n")
title_label = tk.Label(login_frame, text="学生成绩预测系统", font=("Arial", 16, "bold"))
title_label.grid(row=0, columnspan=2, pady=10)

tk.Label(login_frame, text="用户名:", font=("Arial", 12)).grid(row=1, column=0, padx=10, pady=10, sticky="e")
username_entry = tk.Entry(login_frame, font=("Arial", 12))
username_entry.grid(row=1, column=1, padx=10, pady=10)

tk.Label(login_frame, text="密码:", font=("Arial", 12)).grid(row=2, column=0, padx=10, pady=10, sticky="e")
password_entry = tk.Entry(login_frame, show='*', font=("Arial", 12))
password_entry.grid(row=2, column=1, padx=10, pady=10)

login_button = tk.Button(login_frame, text="登录", command=login, font=("Arial", 12), bg="lightblue")
login_button.grid(row=3, columnspan=2, pady=10)
login_button.config(width=15)
```



5.3 主菜单界面

主菜单界面包括两个操作，分别为训练模型和预测成绩。代码和界面如下图所示。

```
def main_menu():
    for widget in window.winfo_children():
        widget.destroy()
    tk.Label(window, text="请选择操作:", bg="#f0f0f0", font=font_large).pack(pady=20)

    train_button = tk.Button(window, text="训练模型", command=model_selection_frame, font=font_medium, bg="#2196F3", fg="white")
    train_button.pack(pady=10)

    predict_button = tk.Button(window, text="预测成绩", command=go_to_prediction, font=font_medium, bg="#2196F3", fg="white")
    predict_button.pack(pady=10)
```



预测成绩的前提是训练模型，如果没有训练模型就预测成绩则会弹出警告。我在全局定义了一个 model 变量，如果 model = None，则说明没有训练模型。

```
def go_to_prediction():
    if model is None:
        messagebox.showwarning("警告", "请先训练模型!")
    return
```




5.4 训练模型界面

训练模型界面可以选择支持向量机回归（SVR）模型的核函数及参数，核函数选择有 linear、poly、rbf。可以在输入框中输入 C 值和 gamma 值的参数。最后就是两个按钮，分别是确认选择和返回。代码和界面如下图所示。

```
# 模型参数选择界面
def model_selection_frame():
    global kernel_var, C_var, gamma_var
    kernel_var = tk.StringVar(value='linear')
    C_var = tk.StringVar(value=1.0)
    gamma_var = tk.StringVar(value='scale')

    for widget in window.winfo_children():
        widget.destroy()

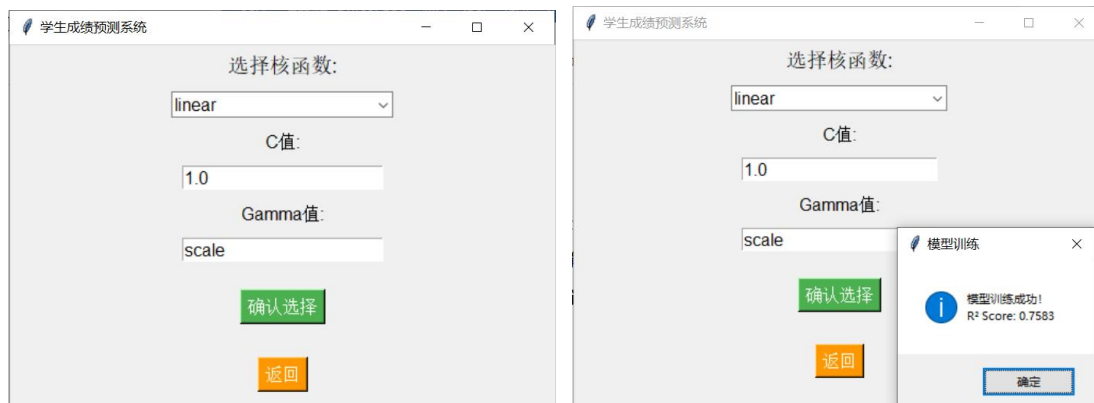
    tk.Label(window, text="选择核函数:", bg="#f0f0f0", font=font_large).pack(pady=5)
    kernel_options = ttk.Combobox(window, textvariable=kernel_var, values=['linear', 'poly', 'rbf'], font=font_medium)
    kernel_options.pack(pady=5)

    tk.Label(window, text="C值:", bg="#f0f0f0", font=font_medium).pack(pady=5)
    tk.Entry(window, textvariable=C_var, font=font_medium).pack(pady=5)

    tk.Label(window, text="Gamma值:", bg="#f0f0f0", font=font_medium).pack(pady=5)
    gamma_entry = tk.Entry(window, textvariable=gamma_var, font=font_medium)
    gamma_entry.pack(pady=5)

    next_button = tk.Button(window, text="确认选择", command=train_model, font=font_medium, bg="#4CAF50", fg="white")
    next_button.pack(pady=20)

    back_button = tk.Button(window, text="返回", command=main_menu, font=font_medium, bg="#FF9800", fg="white")
    back_button.pack(pady=10)
```



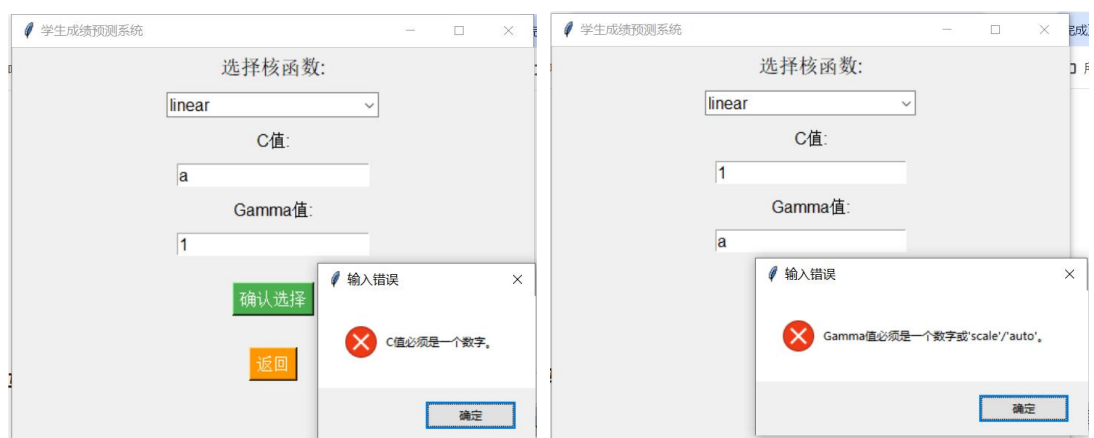
在点击确认选择后就会调用 `train_model` 函数进行模型训练，当然函数里有验证输入的值是否合法，如果 `gamma` 值或 `C` 值输入错误，则会弹出消息框提示错误。代码和界面如下图所示。

```
def train_model():
    global model
    gamma_value = gamma_var.get()
    if gamma_value in ['scale', 'auto']:
        gamma_value = gamma_value
    else:
        try:
            gamma_value = float(gamma_value)
        except ValueError:
            messagebox.showerror("输入错误", "Gamma值必须是一个数字或'scale'/'auto'。")
            return
    c_value = C_var.get()
    try:
        c_value = float(c_value)
    except ValueError:
        messagebox.showerror("输入错误", "C值必须是一个数字。")
        return

    model = SVR(kernel=kernel_var.get(), C=c_value, gamma=gamma_value)
    X = df_final.drop(columns=['Exam_Score'])
    y = df_final['Exam_Score']
    X_train_scaler, X_test_scaler, y_train_scaler, y_test_scaler = train_test_split(X, y, test_size=0.3, random_state=42)
    model.fit(X_train_scaler, y_train_scaler)

    y_pred = model.predict(X_test_scaler)
    r2 = r2_score(y_test_scaler, y_pred)

    messagebox.showinfo("模型训练", f"模型训练成功! \nR² Score: {r2:.4f}")
    main_menu()
```



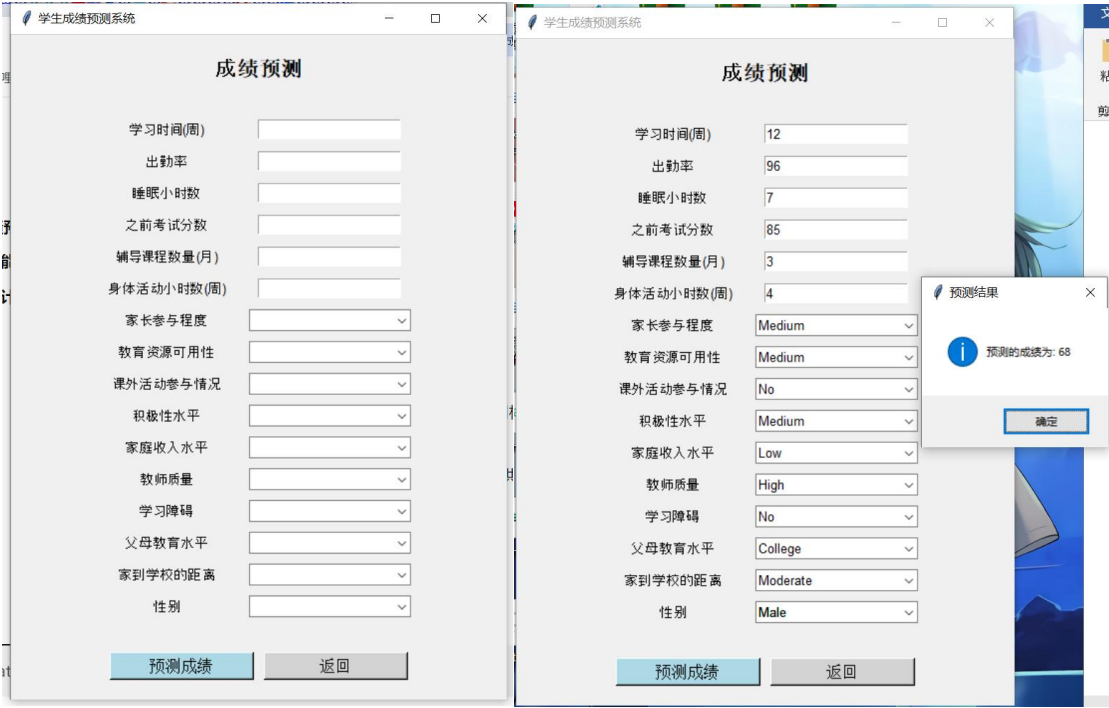
5.5 成绩预测界面

输入相关的特征数据就可以进行成绩预测。输入框可以用来输入数值，如学习时间、出勤率、睡眠小时数等，而类别特征则使用下拉框进行选择，如家长参与程度、教育资源可用性等。最后就是还有两个按钮，分别是预测成绩和返回。此界面的部分代码和界面如下图所示。

```
categorical_features = {
    'Parental_Involvement': ['Low', 'Medium', 'High'],
    'Access_to_Resources': ['Low', 'Medium', 'High'],
    'Extracurricular_Activities': ['Yes', 'No'],
    'Motivation_Level': ['Low', 'Medium', 'High'],
    'Family_Income': ['Low', 'Medium', 'High'],
    'Teacher_Quality': ['Low', 'Medium', 'High'],
    'Learning_Disabilities': ['Yes', 'No'],
    'Parental_Education_Level': ['High School', 'College', 'Postgraduate'],
    'Distance_from_Home': ['Near', 'Moderate', 'Far'],
    'Gender': ['Male', 'Female']
}

categorical_mapping = {
    'Parental_Involvement': '家长参与程度',
    'Access_to_Resources': '教育资源可用性',
    'Extracurricular_Activities': '课外活动参与情况',
    'Motivation_Level': '积极性水平',
    'Family_Income': '家庭收入水平',
    'Teacher_Quality': '教师质量',
    'Learning_Disabilities': '学习障碍',
    'Parental_Education_Level': '父母教育水平',
    'Distance_from_Home': '家到学校的距离',
    'Gender': '性别'
}

for feature, display_name in categorical_mapping.items():
    tk.Label(input_frame, text=display_name, font=("Arial", 10)).grid(row=row_index, column=0, padx=10, pady=5)
    combo = ttk.Combobox(input_frame, values=categorical_features[feature], font=("Arial", 10))
    combo.grid(row=row_index, column=1, padx=10, pady=5)
    inputs[feature] = combo
    row_index += 1
```



在点击预测成绩的按钮之后，将会调用 make_prediction 函数进行成绩预测。因为要确保输入的数据与训练模型时的数据一致，所以需要使用标准化处理确保

特征数据一致性，还要单独对类别特征进行编码，确保与训练模型时一致。代码如下所示。

```
def make_prediction():
    feature_values = {}
    for feature in inputs:
        if isinstance(inputs[feature], ttk.Combobox):
            feature_values[feature] = inputs[feature].get()
        else:
            feature_values[feature] = float(inputs[feature].get())

    features_df = pd.DataFrame([feature_values])

    for feature in ['Parental_Involvement', 'Access_to_Resources', 'Extracurricular_Activities',
                    'Motivation_Level', 'Family_Income', 'Teacher_Quality',
                    'Learning_Disabilities', 'Parental_Education_Level',
                    'Distance_from_Home', 'Gender']:
        value = features_df[feature].iloc[0]
        # 对获得的分类特征进行编码
        if feature in ['Parental_Involvement', 'Access_to_Resources', 'Motivation_Level',
                        'Family_Income', 'Teacher_Quality', 'Distance_from_Home']:
            features_df[f'{feature}_Low'] = 1 if value == 'Low' else 0
            features_df[f'{feature}_Medium'] = 1 if value == 'Medium' else 0
        elif feature in ['Distance_from_Home']:
            features_df[f'{feature}_Moderate'] = 1 if value == 'Moderate' else 0
            features_df[f'{feature}_Near'] = 1 if value == 'Near' else 0
        elif feature in ['Extracurricular_Activities', 'Learning_Disabilities', 'Gender']:
            features_df[f'{feature}_Yes'] = 1 if value == 'Yes' else 0
        elif feature == 'Gender':
            features_df[f'{feature}_Male'] = 1 if value == 'Male' else 0
        elif feature in ['Parental_Education_Level']:
            features_df[f'{feature}_High School'] = 1 if value == 'High School' else 0
            features_df[f'{feature}_Postgraduate'] = 1 if value == 'Postgraduate' else 0

    features_encoded = features_df.reindex(columns=df_final.drop(columns=['Exam_Score']).columns, fill_value=0)
    features_encoded[scaler_features] = scaler.transform(features_encoded[scaler_features])

    prediction = model.predict(features_encoded)
    messagebox.showinfo("预测结果", f"预测的成绩为: {round(prediction[0])}")

    main_menu()

window.mainloop()
```