House Dwelling Prediction

Abstract:

This study aims to forecast homeownership status based on important variables like income, education, and marital status. For classification, radial basis function (RBF) and polynomial kernel Support Vector Machine (SVM) techniques were used. This model, which has an accuracy of 82%, offers important insights that real estate agents, policymakers, and other stakeholders can use to better understand housing occupancy trends. In addition, the study takes into account continuous property operation costs, such as gas, electricity, and fuel, providing a thorough examination of homeownership dynamics.

Introduction:

The goal of this study is to create a technique for predicting property ownership status and pinpointing factors that matter in making this decision. Age, income, and education level are just a few of the variables linked to people and their living situations found in data from IPUMS USA that was obtained through the US Census. It also includes features of the home like the year of construction, population density in the area, and utility costs. To assess the models, we used the Support Vector Machine (SVM) model and its extensions using polynomial kernels and RBF, as well as cross-validation. The data was classified with an accuracy rate of 86%, according to the results.

Theoretical Background:

One strong supervised machine learning technique that excels at both regression and classification problems is Support Vector Machine (SVM). It works by finding the best hyperplane in a high-dimensional space to efficiently divide data points into different classes. This hyperplane, which is designed to maximize the margin, represents the separation between the hyperplane and the support vectors—the nearest data points.

The classification equivalent of Support Vector Machine (SVM) is called Support Vector Classifier (SVC), and its goal is to find the hyperplane with the highest margin between classes and the lowest classification errors. In order to convert input data into a higher-dimensional space where classes can be distinguished linearly, SVC uses a kernel function.

Kernels:

Kernels are essential components of Support Vector Machines (SVM) because they act as functions that calculate the dot product of vectors in a higher-dimensional space without requiring clear and direct data processing. They make SVM more effective in high-dimensional feature spaces by calculating dot products quickly and effectively without involving the calculation of data coordinates.

The three commonly used kernels in SVM are:

Linear:

The most efficient type of kernel for support vector machines (SVMs) is the linear kernel, which works best when data can be split linearly. It computes the dot product between two feature vectors and operates directly in the original input space. When there are a lot of features, this kernel is typically chosen.

Radial:

This kernel can capture intricate relationships between features and is appropriate for non-linearly separable data.

The original data is transformed into an infinite-dimensional space by this kernel using Gaussian radial basis functions, which captures complex correlations between the data points.

Polynomial:

Using polynomial functions, the polynomial kernel converts the data into a higher-dimensional space. Dealing with non-linear separable data benefits from this. The polynomial function of the dot products is evaluated by the polynomial kernel in order to determine the similarity between vectors.

Parameters:

Parameters are values that define the configuration of a model or algorithm and influence its behavior and performance.

Linear:

C (regularization parameter) – Regulates the trade-off between reducing model complexity for improved generalization to new data and reaching a low error on the training set. Higher values of C attempt to correctly classify all training examples (higher model complexity); lower values, on the other hand, provide more room for error in classification (simpler model).

Radial :

Gamma - Describes how well a particular training sample affects the decision boundary. The gamma value determines how closely the decision boundaries end up encircling points in the input space; a low gamma value indicates a point has a far reach, and a high gamma value indicates a point has a limited reach.

Polynomial:

Degree - sets up the polynomial's degree, which is then used to locate the ideal hyperplane. Decision boundaries become more complex as degree increases.

Tuning :

In SVM, hyperparameter tuning is crucial to maximizing the model's efficiency. It entails modifying variables such as degree, gamma, and C in the case of polynomial kernels. The objective is to determine the best hyperparameter combination that produces a highly accurate and broadly applicable model.

Methodology:

Methodology is to preprocess and clean the dataset, select relevant variables, and train various models to predict homeownership status accurately.

Step-wise Process:

1. Data Pre-Processing:

The data preprocessing phase involved several steps to prepare the dataset for analysis and modeling. Initially, the dataset was explored using functions like head() and info() to gain an understanding of its structure and contents. With 75,388 rows and multiple columns, redundancy was identified, particularly with multiple rows sharing the same serial number. To address this issue, the dataset was grouped by serial number, and for each serial number, only the row with the maximum age was retained. This process effectively reduced the dataset to 30,802 rows, eliminating redundancy and ensuring that each serial number corresponded to a unique individual with their maximum age recorded. This step was crucial in streamlining the dataset and ensuring the integrity of the data for subsequent analysis and modeling tasks.

2. Analysis:

The correlation matrix was printed to determine which variables had poor correlations with the target variable, OWNERSHP. To make the dataset more manageable, low-correlated columns were subsequently eliminated. Serial number, population density, utility costs, individual level data, total pre-tax income of individuals, household income, rooms, in-depth educational information, number of families and couples, house value, age of the home, and number of rooms were among the columns that were removed. By keeping only the most important variables, this procedure attempted to streamline the dataset and provide a more targeted and effective analysis for the tasks that followed.

3. Conversion of categorical Variables to Numerical:

Based on predetermined criteria, numerical representations of categorical variables were created. Income levels were divided into three groups for the 'HHINCOME' column: incomes under 100,000, incomes between 100,000 and 250,000, and incomes over 250,000. Values of 9 in the 'VEHICLES' column were replaced with 0, signifying the lack of vehicles, during processing.

The dataset was streamlined by these conversions, which improved its suitability for further analysis and modeling tasks.

4. Creation of New features:

To capture the educational and marital statuses, new binary columns were added. The variables "Married," "Divorced," and "NeverMarried" were used to indicate the marital status of the individuals. MARST values 1 and 2 denoted those who were married, while values 3 through 5 indicated those who were divorced. The educational levels were also represented by designating 'NoSchooling' as 1 for those who have never attended school (EDUC value 0), 'PrimarySchooling' as 1 for those who have attended primary school (EDUC value 1), 'Schooling' as 1 for those who have attended general school (EDUC value 2), 'HighSchooling' as 1 for those who have attended high school (EDUC values between 3 and 6), and 'College' as 1 for those who have attended college or higher (EDUC value 7 or greater). More insights into demographic traits were made possible by these new features, which allowed for more thorough analysis and modeling.

5. Model Building:

Training three distinct model types—linear, radial, and polynomial. While incorporating cross-validation for performance assessment was the process of model building. Thirty percent of the dataset was used for testing and seventy percent was used for training for each type of model. The default parameters were used to fit the linear model, and cross-validation was used to evaluate the model's accuracy and error rates. In a similar vein, the radial model was trained using default parameters, and its performance metrics were assessed using cross-validation. The polynomial model was also built and assessed using the same methodology. The models' capacity for generalization was evaluated through the use of cross-validation techniques, guaranteeing their robustness and reliability in predicting homeownership status.

Computational Results:

The following table presents the performance metrics for three different kernels of the Support Vector Machine (SVM) using default parameters:

Metric	Linear (Default)	Radial (Default)	Polynomial (Default)
Training Accuracy	0.82	0.821	0.819
Testing Accuracy	0.821	0.824	0.822
Training Error Rate	0.179	0.178	0.18
Testing Error Rate	0.178	0.175	0.177

Findings :

The Radial kernel has the lowest testing error rate and a marginally higher testing accuracy.

The results of the Polynomial and Linear kernels are similar, although the Polynomial kernel has a slightly higher training error rate.

Based on its testing metrics, the Radial kernel appears to have a slight advantage in terms of generalization, even though all three kernels perform similarly.

Support Vector Machine (SVM) after cross validation parameters:

Metric	**Linear - CV(C = 10)**	**Radial - CV (gamma =1)**	**Polynomial - CV(Degree = 1)**
Training Accuracy	0.821	0.823	0.82
Testing Accuracy	0.823	0.826	0.821
Training Error Rate	0.178	0.176	0.179
Testing Error Rate	0.176	0.173	0.178

Findings:

Out of the three kernels, the Radial kernel with gamma set to 1 exhibits the best overall performance, obtaining the highest testing accuracy and the lowest testing error rate.

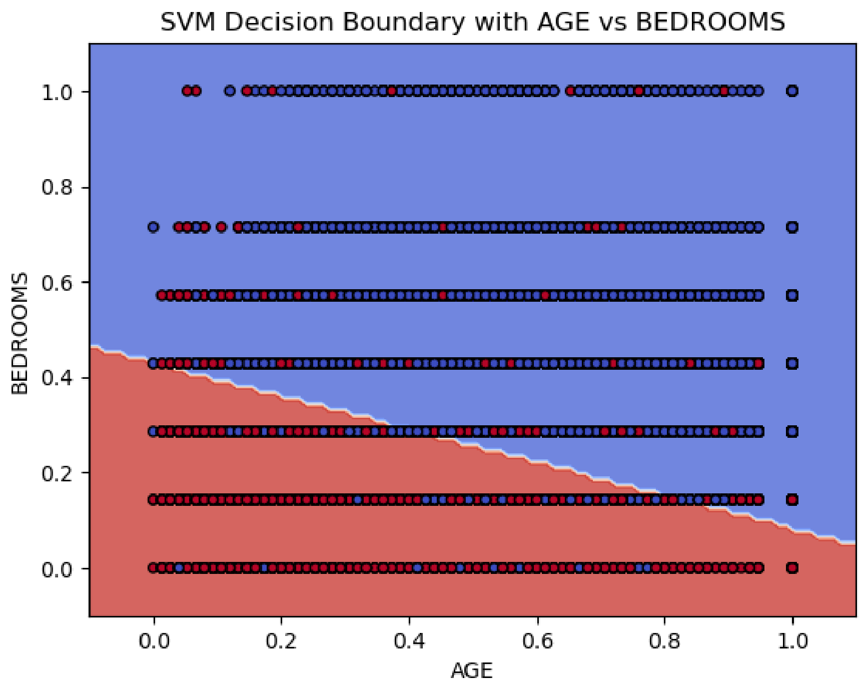
The performance metrics of the Linear (C=10) and Polynomial (Degree=1) kernels are similar, with the Linear kernel exhibiting marginally higher testing accuracy and error rates than the Polynomial kernel.

These findings suggest that the model's performance can be enhanced by carefully adjusting the hyperparameters, particularly for the Radial kernel in this instance.

Discussion:

In this study, we used Support Vector Machine (SVM) models to predict dwelling occupancy status based on specific sociodemographic factors. Features in the dataset included age, number of vehicles owned, Number of bedrooms ,household income, and marital status and education.

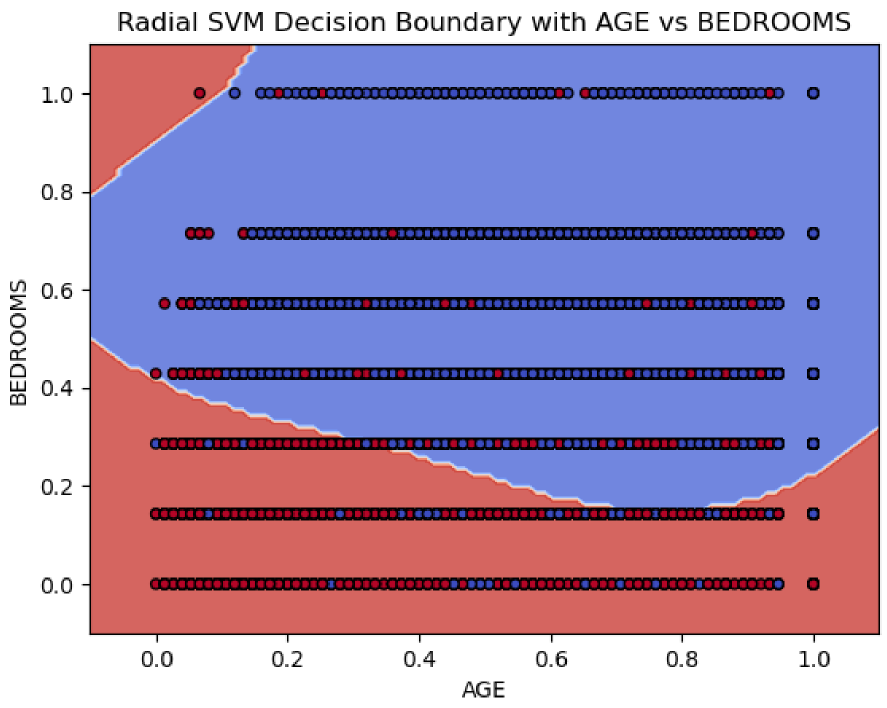
Linear Plot:



Discussion on the above plot:

The SVM plot shows that younger people are more likely to rent than own their homes when they have more bedrooms. Even with fewer bedrooms, people's chances of owning a home rise with age. This pattern implies that homeownership is highly influenced by age, possibly even more so than by home size. Homeownership rates rise as people age because older people tend to save money, that they can use to buy homes. This suggests that having financial stability, which usually increases with age, is essential to being able to buy a property.

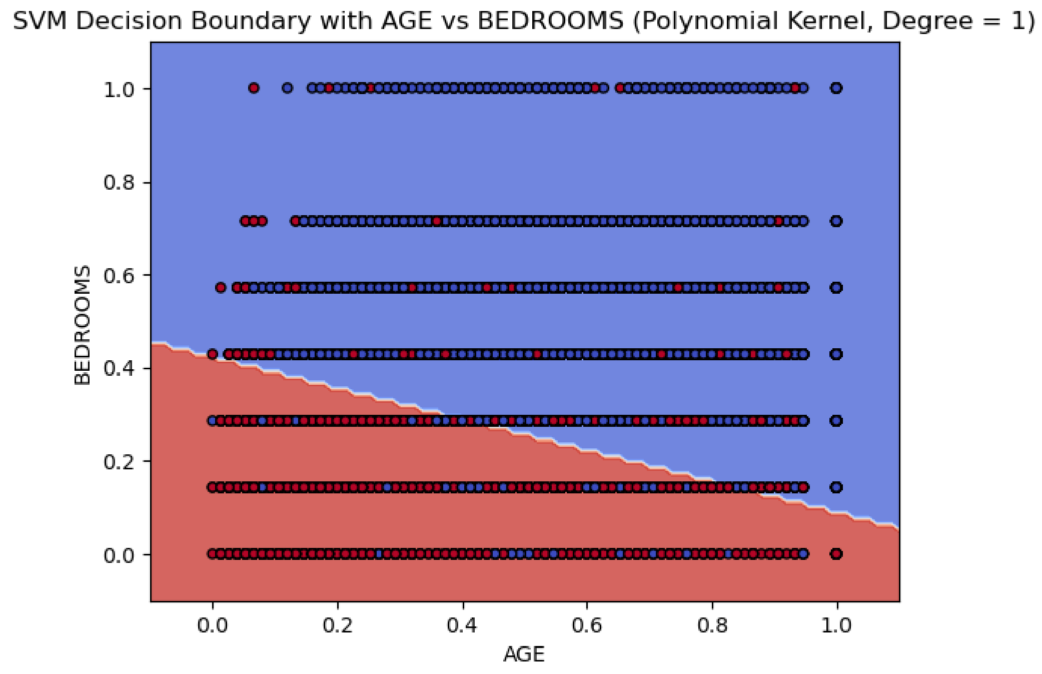
Radial plot:



Discussion on the above plot:

The 'AGE' and 'BEDROOMS' radial SVM plot highlights the impact of these two variables on home ownership or rental status. The curved line in this plot separates the owners from the renters. It demonstrates how many younger people do not own their homes, even those with lots of bedrooms. This could be the case because they are young professionals without sufficient savings to purchase a home. Even if they live in houses with fewer bedrooms, people are more likely to be homeowners as they get older. This shift might be the result of older people having more time to accumulate savings, which has made it simpler for them to become homeowners. The plot highlights that as people age, they are more likely to become homeowners.

Polynomial Plot:



Discussion on the above plot:

Due to the polynomial kernel's degree value being 1 following hyperparameter tuning, the polynomial SVM plot for "AGE" and "BEDROOMS" looks extremely similar to the linear SVM plot. This figure indicates that the polynomial kernel performs operations akin to those of a linear kernel. The plot suggests that younger people who live in homes with lots of bedrooms typically rent rather than buy. Even if a home has fewer bedrooms, owning a home becomes more likely as people get older. This pattern implies that going from renting to owning a home requires having financial stability, which usually gets better with age.

Interpretation :

In order to determine whether people own or rent their homes, the SVM decision boundary plots with linear, radial, and polynomial kernels examine the correlation between age and the number of bedrooms. No matter how many bedrooms a home has, older people are more likely to own one, according to every model, which highlights the importance of age. Age-related increases in homeownership are consistent and direct, according to the degree-one polynomial and linear models. A more in-depth look is provided by the radial model, which highlights particular age groups and bedroom configurations where homeownership trends may differ. All things considered, these models highlight the role that age plays in housing stability, implying that older age groups generally attain better financial stability, which makes homeownership possible.

Conclusions:

This study uses SVM models and demographic data to examine the factors that determine homeownership versus renting. Despite achieving a commendable 76% accuracy rate, the data selection process contained significant flaws, most notably the assumption of a particular homeownership threshold based on income. Notwithstanding these drawbacks, the study provides insightful information to real estate experts, policymakers, and urban planners. Comprehending these variables empowers interested parties to enact policies that advance equitable, easily accessible, and environmentally friendly housing options. This study aids in the development of inclusive communities and helps real estate agents predict market trends. This study, which addresses issues that are important to a wide range of stakeholder groups, offers crucial guidance for well-informed decision-making in housing and city planning with an accuracy rate of 82%.

References:

IPUMS USA. IPUMS USA Version 11.0

(https://www.ipums.org/projects/ipums-usa/d010.v11.0)

GeeksforGeeks. Support Vector Machine (SVM) Algorithm

(https://www.geeksforgeeks.org/support-vector-machine-algorithm/)

Appendix:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
import warnings
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.svm import SVC
warnings.filterwarnings("ignore")

import pandas as pd
```

```
path = "C:\\Users\\ADMIN\\Desktop\\DATA 5322 SML-2\\Assignments\\Assignment-2\\Housing.csv"
Housing =  pd.read_csv(path)


Housing.head(10)
Housing.columns
Housing.shape

null_values = Housing.isnull().sum()
print(null_values)


idx_max_age = Housing.groupby('SERIAL')['AGE'].idxmax()
df_g = Housing.loc[idx_max_age]
print(df_g)

df_g.shape

print(df_g.corr())

columns_to_drop = ['SERIAL', 'OWNERSHPD','DENSITY', 'COSTELEC', 'COSTGAS', 'COSTWATR',
'COSTFUEL',
              'PERNUM', 'PERWT', 'BIRTHYR', 'INCTOT','EDUCD','ROOMS']

df = df_g.drop(columns=columns_to_drop)

print(df.head(10))

def reassign_income(value):
    if value < 100000:
        return 1
    elif 100001 <= value <= 250000:
        return 2
    else:
        return 3

df['HHINCOME'] = df['HHINCOME'].apply(reassign_income)


def reassign_vehicles(value):
    if value == 9:
        return 0
    else:
        return value

df['VEHICLES'] = df['VEHICLES'].apply(reassign_vehicles)

df['Married'] = 0
```

```
df['Divorced'] = 0
df['NeverMarried'] = 0

df.loc[df['MARST'].isin([1, 2]), 'Married'] = 1
df.loc[df['MARST'].isin([3, 4, 5]), 'Divorced'] = 1
df.loc[df['MARST'] == 6, 'NeverMarried'] = 1


df['NoSchooling'] = 0
df['PrimarySchooling'] = 0
df['Schooling'] = 0
df['HighSchooling'] = 0
df['College'] = 0

df.loc[df['EDUC'] == 0, 'NoSchooling'] = 1
df.loc[df['EDUC'] == 1, 'PrimarySchooling'] = 1
df.loc[df['EDUC'] == 2, 'Schooling'] = 1
df.loc[df['EDUC'].between(3, 6), 'HighSchooling'] = 1
df.loc[df['EDUC'] >= 7, 'College'] = 1


df.head()


df.drop('MARST', axis=1, inplace=True)
df.drop('EDUC', axis=1, inplace=True)


df.head()

df.columns

df.shape

#Linear

from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split


selected_columns = ['OWNERSHP', 'VEHICLES', 'Married', 'NeverMarried', 'AGE','BEDROOMS',
'HighSchooling',
                'College']
```

```
X = df[selected_columns[1:]]
y = df['OWNERSHP']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

svc_default = SVC(kernel='linear')
svc_default.fit(X_train_scaled, y_train)

y_train_pred_default = svc_default.predict(X_train_scaled)
y_test_pred_default = svc_default.predict(X_test_scaled)

train_accuracy_default = accuracy_score(y_train, y_train_pred_default)
test_accuracy_default = accuracy_score(y_test, y_test_pred_default)

train_error_rate_default = 1 - train_accuracy_default
test_error_rate_default = 1 - test_accuracy_default

print("Linear SVM Model with Default Parameters:")
print("Training Accuracy:", train_accuracy_default)
print("Testing Accuracy:", test_accuracy_default)
print("Training Error Rate:", train_error_rate_default)
print("Testing Error Rate:", test_error_rate_default)
print()

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}

svc = SVC(kernel='linear')

grid_search = GridSearchCV(svc, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

best_params = grid_search.best_params_

best_svc = SVC(kernel='linear', C=best_params['C'])
best_svc.fit(X_train_scaled, y_train)

y_train_pred = best_svc.predict(X_train_scaled)
y_test_pred = best_svc.predict(X_test_scaled)

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
```

```
train_error_rate = 1 - train_accuracy
test_error_rate = 1 - test_accuracy

print("Linear SVM Model with Cross-Validation Parameters:")
print("Best Hyperparameters:", best_params)
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
print("Training Error Rate:", train_error_rate)
print("Testing Error Rate:", test_error_rate)


age_idx  = X.columns.get_loc('AGE')
bedrooms_idx = X.columns.get_loc('BEDROOMS')

X_plot = X.iloc[:, [age_idx,bedrooms_idx]]

scaler_plot = MinMaxScaler()
X_plot_scaled = scaler_plot.fit_transform(X_plot)

best_svc_plot = SVC(kernel='linear', C=best_params['C'])
best_svc_plot.fit(X_plot_scaled, y)

x_min, x_max = X_plot_scaled[:, 0].min() - 0.1, X_plot_scaled[:, 0].max() + 0.1
y_min, y_max = X_plot_scaled[:, 1].min() - 0.1, X_plot_scaled[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
Z = best_svc_plot.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)


plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X_plot_scaled[:, 0], X_plot_scaled[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
plt.xlabel('AGE')
plt.ylabel('BEDROOMS')
plt.title('SVM Decision Boundary with AGE vs BEDROOMS')
plt.show()

#Radial


selected_columns = ['OWNERSHP', 'VEHICLES', 'Married', 'NeverMarried', 'AGE','BEDROOMS',
'HighSchooling',
               'College']


X = df[selected_columns[1:]]
y = df['OWNERSHP']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

default_svc_rbf = SVC(kernel='rbf')
default_svc_rbf.fit(X_train_scaled, y_train)

y_train_pred_default = default_svc_rbf.predict(X_train_scaled)
y_test_pred_default = default_svc_rbf.predict(X_test_scaled)

train_accuracy_default = accuracy_score(y_train, y_train_pred_default)
test_accuracy_default = accuracy_score(y_test, y_test_pred_default)

train_error_rate_default = 1 - train_accuracy_default
test_error_rate_default = 1 - test_accuracy_default

print("Radial SVM Model with Default Parameters:")
print("Training Accuracy:", train_accuracy_default)
print("Testing Accuracy:", test_accuracy_default)
print("Training Error Rate:", train_error_rate_default)
print("Testing Error Rate:", test_error_rate_default)
print()

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1, 1]}

svc_rbf = SVC(kernel='rbf')

grid_search_rbf = GridSearchCV(svc_rbf, param_grid, cv=5)
grid_search_rbf.fit(X_train_scaled, y_train)

best_params_rbf = grid_search_rbf.best_params_
print("Best Hyperparameters with Radial Kernel:", best_params_rbf)

best_svc_rbf = SVC(kernel='rbf', C=best_params_rbf['C'], gamma=best_params_rbf['gamma'])
best_svc_rbf.fit(X_train_scaled, y_train)

y_train_pred_rbf = best_svc_rbf.predict(X_train_scaled)
y_test_pred_rbf = best_svc_rbf.predict(X_test_scaled)

train_accuracy_rbf = accuracy_score(y_train, y_train_pred_rbf)
test_accuracy_rbf = accuracy_score(y_test, y_test_pred_rbf)

# Calculate error rates
train_error_rate_rbf = 1 - train_accuracy_rbf
test_error_rate_rbf = 1 - test_accuracy_rbf

print("Radial SVM Model with Cross-Validation Parameters:")
print("Training Accuracy :", train_accuracy_rbf)
print("Testing Accuracy :", test_accuracy_rbf)
print("Training Error Rate :", train_error_rate_rbf)
print("Testing Error Rate :", test_error_rate_rbf)



top_variables_rbf = ['AGE','BEDROOMS']

X_top_rbf = X[top_variables_rbf]

X_train_top_rbf, X_test_top_rbf, y_train_rbf, y_test_rbf = train_test_split(X_top_rbf, y, test_size=0.3,
random_state=42)

scaler_top_rbf = MinMaxScaler()
X_train_scaled_top_rbf = scaler_top_rbf.fit_transform(X_train_top_rbf)
X_test_scaled_top_rbf = scaler_top_rbf.transform(X_test_top_rbf)

svc_top_rbf = SVC(kernel='rbf', C=best_params_rbf['C'], gamma=best_params_rbf['gamma'])

svc_top_rbf.fit(X_train_scaled_top_rbf, y_train_rbf)

x_min_rbf, x_max_rbf = X_train_scaled_top_rbf[:, 0].min() - 0.1, X_train_scaled_top_rbf[:, 0].max() + 0.1
y_min_rbf, y_max_rbf = X_train_scaled_top_rbf[:, 1].min() - 0.1, X_train_scaled_top_rbf[:, 1].max() + 0.1
xx_rbf, yy_rbf = np.meshgrid(np.linspace(x_min_rbf, x_max_rbf, 100), np.linspace(y_min_rbf,
y_max_rbf, 100))
Z_rbf = svc_top_rbf.predict(np.c_[xx_rbf.ravel(), yy_rbf.ravel()])
Z_rbf = Z_rbf.reshape(xx_rbf.shape)

plt.contourf(xx_rbf, yy_rbf, Z_rbf, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X_train_scaled_top_rbf[:, 0], X_train_scaled_top_rbf[:, 1], c=y_train_rbf,
cmap=plt.cm.coolwarm, s=20, edgecolors='k')
plt.xlabel(top_variables_rbf[0])
plt.ylabel(top_variables_rbf[1])
plt.title('Radial SVM Decision Boundary with AGE vs BEDROOMS')
plt.show()

#Polynomial

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

default_svc_poly = SVC(kernel='poly')
default_svc_poly.fit(X_train_scaled, y_train)

y_train_pred_default_poly = default_svc_poly.predict(X_train_scaled)
y_test_pred_default_poly = default_svc_poly.predict(X_test_scaled)
```

```
train_accuracy_default_poly = accuracy_score(y_train, y_train_pred_default_poly)
test_accuracy_default_poly = accuracy_score(y_test, y_test_pred_default_poly)

train_error_rate_default_poly = 1 - train_accuracy_default_poly
test_error_rate_default_poly = 1 - test_accuracy_default_poly

print("Polynomial SVM Model with Default Parameters:")
print("Training Accuracy:", train_accuracy_default_poly)
print("Testing Accuracy:", test_accuracy_default_poly)
print("Training Error Rate:", train_error_rate_default_poly)
print("Testing Error Rate:", test_error_rate_default_poly)

print()

param_grid_poly = {'degree': [0,1, 2, 3]}

svc_poly = SVC(kernel='poly')

grid_search_poly = GridSearchCV(svc_poly, param_grid_poly, cv=5)
grid_search_poly.fit(X_train_scaled, y_train)

best_degree_poly = grid_search_poly.best_params_['degree']

best_svc_poly = SVC(kernel='poly', degree=best_degree_poly)
best_svc_poly.fit(X_train_scaled, y_train)

y_train_pred_poly = best_svc_poly.predict(X_train_scaled)
y_test_pred_poly = best_svc_poly.predict(X_test_scaled)

train_accuracy_poly = accuracy_score(y_train, y_train_pred_poly)
test_accuracy_poly = accuracy_score(y_test, y_test_pred_poly)

train_error_rate_poly = 1 - train_accuracy_poly
test_error_rate_poly = 1 - test_accuracy_poly

print("Polynomial SVM Model with Cross-Validation Parameters:")
print("Best Degree:", best_degree_poly)
print("Training Accuracy (Degree={}):".format(best_degree_poly), train_accuracy_poly)
print("Testing Accuracy (Degree={}):".format(best_degree_poly), test_accuracy_poly)
print("Training Error Rate (Degree={}):".format(best_degree_poly), train_error_rate_poly)
print("Testing Error Rate (Degree={}):".format(best_degree_poly), test_error_rate_poly)


top_variables_poly = ['AGE','BEDROOMS']

X_top_poly = X[top_variables_poly]
```

```
X_train_top_poly, X_test_top_poly, y_train_poly, y_test_poly = train_test_split(X_top_poly, y,
test_size=0.3, random_state=42)

scaler_top_poly = MinMaxScaler()
X_train_scaled_top_poly = scaler_top_poly.fit_transform(X_train_top_poly)
X_test_scaled_top_poly = scaler_top_poly.transform(X_test_top_poly)

svc_top_poly = SVC(kernel='poly', degree=best_degree_poly)

svc_top_poly.fit(X_train_scaled_top_poly, y_train_poly)

x_min_poly, x_max_poly = X_train_scaled_top_poly[:, 0].min() - 0.1, X_train_scaled_top_poly[:, 0].max()
+ 0.1
y_min_poly, y_max_poly = X_train_scaled_top_poly[:, 1].min() - 0.1, X_train_scaled_top_poly[:, 1].max()
+ 0.1
xx_poly, yy_poly = np.meshgrid(np.linspace(x_min_poly, x_max_poly, 100), np.linspace(y_min_poly,
y_max_poly, 100))
Z_poly = svc_top_poly.predict(np.c_[xx_poly.ravel(), yy_poly.ravel()])
Z_poly = Z_poly.reshape(xx_poly.shape)

plt.contourf(xx_poly, yy_poly, Z_poly, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X_train_scaled_top_poly[:, 0], X_train_scaled_top_poly[:, 1], c=y_train_poly,
cmap=plt.cm.coolwarm, s=20, edgecolors='k')
plt.xlabel(top_variables_poly[0])
plt.ylabel(top_variables_poly[1])
plt.title('SVM Decision Boundary with AGE vs BEDROOMS (Polynomial Kernel, Degree =
{})'.format(best_degree_poly))
plt.show()
```

Thanks
Nikhil Raj Uppari