CLASSIFICATION OF HOUSE OWNERSHIP

ABSTRACT

This study was to classify the ownership of housing whether the dwelling is occupied by owners or renters from the data collected from US census accessed through IPUMS USA. We classify the ownership using three Support Vector Machine (SVM) models which are Linear, Radial and Polynomial kernels. The data contains many economic and demographic variables which are preprocessed and subset for accurate performance of the model. In this study, the subset data is splitted into train and test data, performed parameter tuning and then fitted the SVM models. The SVM decision boundary plots are fitted with two strong predictors from each model. All the three models of SVM have performed well and have good accuracy scores. From the overall findings it has been found that the people with old age staying in the house tend to be the owners.

INTRODUCTION

The main goal of this is to classify the house ownership using various factors such as economic and demographic variables. We will be using Support Vector Classifiers to investigate whether the dwelling is occupied by the owner or renter.

Dataset Description

The dataset is collected from the US Census and was accessed through IPUMS USA. The dataset contains a total of 75388 observations and 24 variables which are Population Density, Age, Martial Status, House Value, Household Income , Total Income , Educational details, Number of rooms and bedrooms, Maritial status, Ownership details and many more etc. Some variables in the dataset are categorical which must be encoded.

With these features we will be predicting the ownership of the person staying in the dwelling is either owner or renter using three kernels models of SVM which are linear, radial and polynomial. In SVM while classifying the data, the hyperplanes divides the data points into classes.

THEORETICAL BACKGROUND

Support Vector Machines (SVM) supervised learning algorithm in machine learning used for solving both classification and regression. The main goal of SVM algorithm is to find the best optimal hyperplane which divides the data into two classes. SVM is very effective in handling high dimensional data using three types of kernels.

The three types of kernels that used are:

1) Linear Kernel
   Linear Kernel is a very simple and commonly used kernel in SVM. The Linear kernel is applied when the data is linearly separable. It defines a decision boundary with a straight line for hyperplanes.

2) Radial Kernel
   The Radial Kernel also known as Radial Basis Function(RBF) kernel in SVM is used when the data is non-linearly separable. This kernel allows non-linear decision boundaries by transforming input features into high dimensional space.

3) Polynomial Kernel
   The polynomial kernel is a kernel in SVM used when the data is non-linearly separable. Similar to radial kernel, this kernel also allows non-linear decision boundaries by transforming input features into high dimensional space. The decision boundaries between classes are approximated by using polynomial function.

Parameter Tuning

Parameter Tuning is used in every machine learning algorithm even in SVM to get the accurate score of the model. There are many approaches for parameter tuning. In our study we will be using GridSearchCV approach for tuning. This method performs an exhaustive search over a specified parameter grid. In this approach, a range of values are taken for each parameter and the model's performance is based on every possible combination.

METHODOLOGY

Data Cleaning

Firstly, we start with loading the Housing data to the pandas dataframe using read_csv. After loading we observed that the entire dataframe contains 75388 rows and 24 variables. While doing data preprocessing we first check if there are any null values in the dataframe and we found that there are no null values in the dataframe. Next, we dropped all the duplicate values from the ‘SERIAL’ (household identification number) by grouping old member of each house. Then, unnecessary variables like ‘SERIAL’, 'OWNERSHPD' (Ownership of dwelling [detailed version]), 'COSTELEC' (Cost of electricity), 'COSTGAS' (Cost of gas), 'COSTWATR' (Cost of Water), 'COSTFUEL' (Cost of Fuel), 'PERNUM' (Person number), 'PERWT'(Person Weight) ,'VALUEH'(House Value), 'BUILTYR2'(Year built), 'BIRTHYR'(birth year), 'EDUCD' (Educational attainment [detailed version]) are dropped from the dataframe. Variables like ‘MARST’(Martial status) and ‘EDUC’ (Educational attainment [general version]) are divided into

categorical columns and then converted into dummy variable. Finally, the data frame is resulted into 30802 rows and 27 columns.

Data Splitting

After, cleaning and preprocessing of data we choose OWNERSHP(Ownership of dwelling) as the response variable and the remaining variables as predictors. Then, we split the data in to train an test split with the ratio of 50:50 and then scaled the training and testing data using MinMaxScaler to ensure that each feature contributes equally for model fitting.

Model Fitting

Next, we fit the SVM Linear kernel model to the trained data and evaluate the model on the testing data to get the accuracy score. To get the accurate score from the model we perform parameter tuning using GridSearchCV on the model with certain parameter values and then refit the model again with new parameters. So, to plot SVM for two strong predictors we find the feature importances. Similarly, we perform with other models Radial and Polynomial. In Linear, feature importance can be found by comparing the coefficients. But in Non-Linear models like Radial and Polynomial we cannot achieve feature importance by comparing coefficients because the SVMs maps the data to a higher dimension, the linear nature of the model is broken, so the coefficients are not available. So, we use permutation importance to find the feature importance.

COMPUTATIONAL RESULTS

Accuracy scores

Kernel Type	Accuracy score before tuning	Accuracy score after tuning
Linear	0.8278	0.8272
Radial	0.8285	0.8318
Polynomial	0.8255	0.8304

Strong Predictors

Top 5 strong predictors for Linear Kernel

Feature	Importance score (Coefficient)
HHINCOME	4.477
BEDROOMS	4.199
NFAMS	2.605
AGE	2.300
ROOMS	1.971

Top 5 strong predictors for Radial Kernel

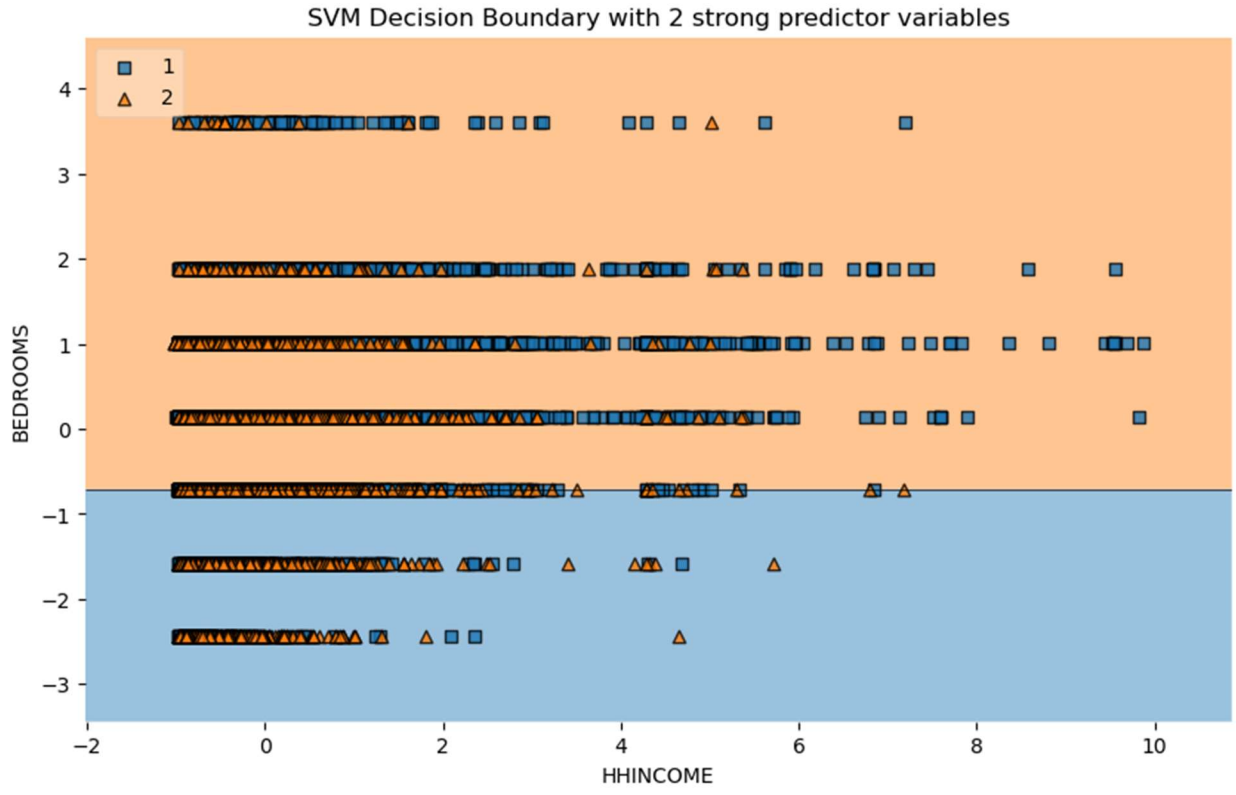
Feature	Importance score (Permutation)
BEDROOMS	0.054
AGE	0.041
ROOMS	0.017
VEHICLES	0.014
INCTOT	0.008

Top 5 strong predictors for Polynomial Kernel

Feature	Importance score (Permutation)
BEDROOMS	5.939874e-02
AGE	3.960782e-02
MARST_married,spouse_present	2.376469e-02
ROOMS	1.372638e-02
EDUC_4yrs_college	9.012402e-03

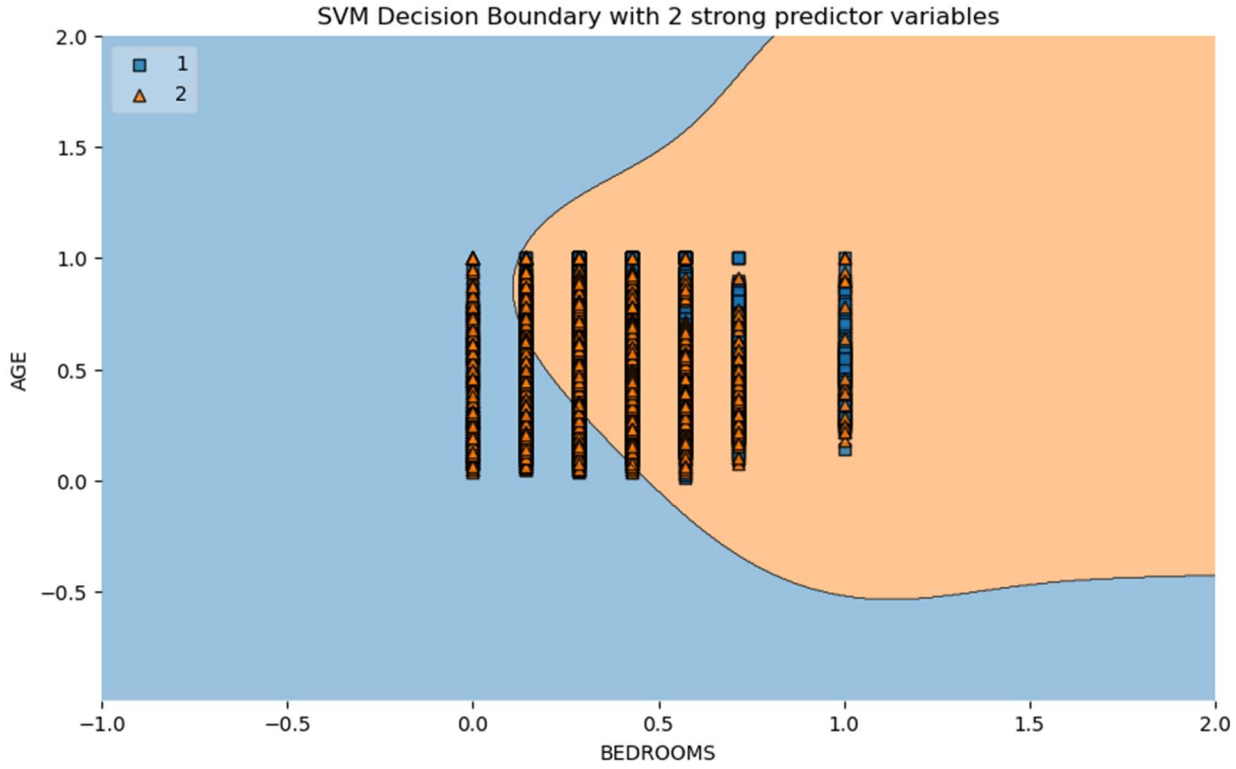
DISCUSSION

Linear Kernel



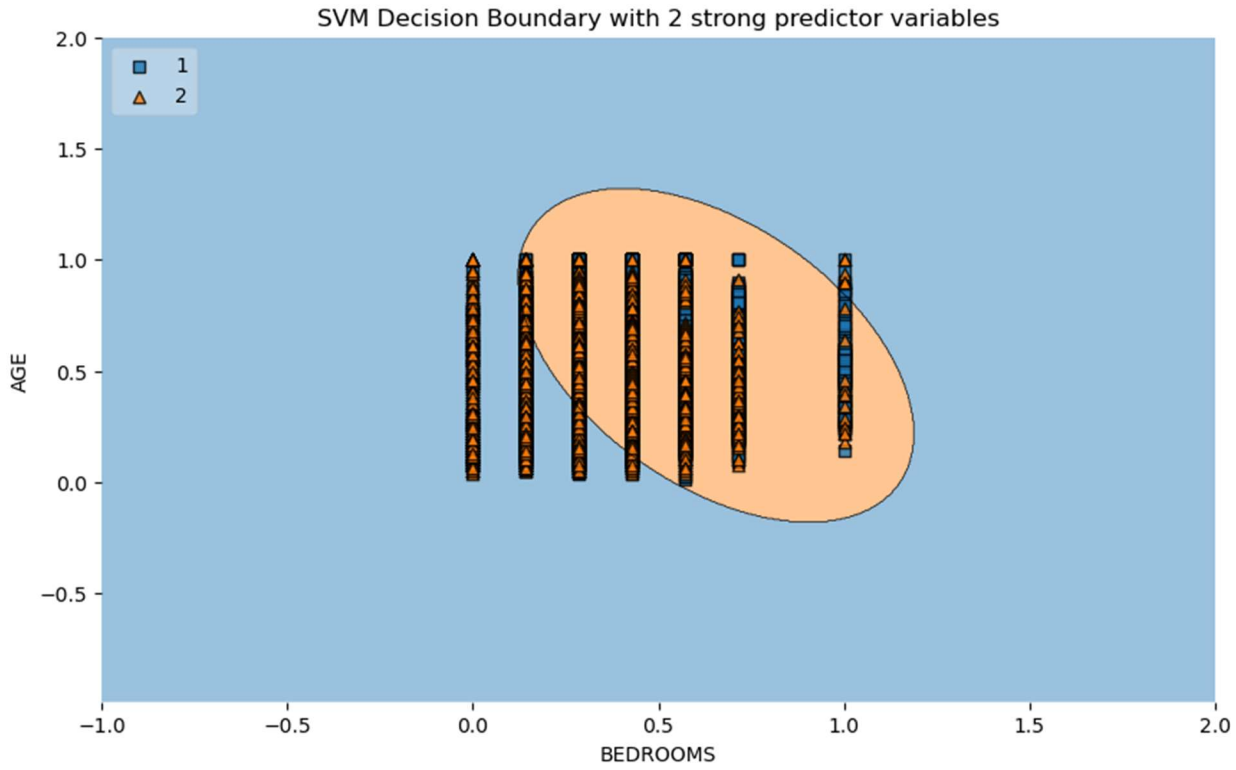
The SVM Decision Boundary plot with 2 strong predictor [HHINCOME, BEDROOMS] from Linear Kernel. In this plot we can observe that the decision boundary divides two classes where the upper part is defined as Renters (2) class and the lower part is defined as Owner (1) Class. We can see that many misclassified points on both the sides of the margin where more Renters misclassified Owners.

Radial Kernel



The SVM Decision boundary plot with two strong predictors [‘BEDROOMS’, ‘AGE’] from Radial Kernel. We can observe that the decision boundary plot is non-linear and it is constructed with C=10 and gamma=0.5. From the plot, the region which is blue in color is defined as ‘OWNERS’(1) class and the another region is defined as ‘RENTERS’ (2) class. In this plot we can observe that there are many misclassified points of data on both the regions.

Polynomial Kernel



The SVM Decision boundary plot with two strong predictors [‘BEDROOMS’, ‘AGE’] from Polynomial Kernel. We can observe that the decision boundary plot is non-linear and it is constructed with the polynomial degree of 2. From the plot, the circular region is defined as ‘RENTERS’ class and the remaining region is defined as ‘OWNERS’ class. In this plot we can observe that there are many data points misclassified on both sides of region.

CONCLUSION

In this study our main goal is to classify house ownership based on the data. The classification is done using three kernels of Support Vector Machines (SVM) which are Linear, Radial and Polynomial.

From Linear Kernel, we can observe that with C=10 (regularization parameter) gives best accuracy score which is 0.8272 (82.72%). The two strong predictors are identified as ‘HHINCOME’ (household income) and ‘BEDROOMS’(number of bedrooms).

From Radial Kernel, we can observe that with C=10 and gamma = 0.5 gives the best accuracy score which is 0.8318 (83.18%). The two strong predictors identified as ‘BEDROOMS’(number of bedrooms) and ‘AGE’ (age of the individual).

From Polynomial Kernel, we can observe that with C=5 and degree = 2 gives the best accuracy score which is 0.8304 (83.04%). The two strong predictors identified in this model are 'BEDROOMS'(number of bedrooms) and 'AGE' (age of the individual).

Based on the overall findings, it is concluded that people with more age and having good number of bedrooms in the house are more likely to be owners of a house they have occupied.

BIBLIOGRAPHY

Support Vector Machines **https://scikit-learn.org/stable/modules/svm.html**

Understanding the Support Vector Machine (SVM) model by David Fagbuyrio

https://medium.com/@davidfagb/understanding-the-support-vector-machine-svm-model-c8eb9bd54a97

Steven Ruggles, Sarah Flood, Matthew Sobek, Danika Brockman, Grace Cooper, Stephanie Richards, and Megan Schouweiler. IPUMS USA: Version 13.0 [dataset]. Minneapolis, MN: IPUMS, 2023. https://doi.org/10.18128/D010.V13.0

APPENDIX

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn.model_selection as skm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler
from sklearn.inspection import permutation_importance
from mlxtend.plotting import plot_decision_regions
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
```

In [2]:
```
df = pd.read_csv('Housing.csv')
df.head()
```

Out[2]:

	SERIAL	DENSITY	OWNERSHP	OWNERSHPD	COSTELEC	COSTGAS	COSTWATR	COSTFUEL	HHI
0	1371772	920.0	1	13	9990	9993	360	9993	
1	1371773	3640.9	2	22	1080	9993	1800	9993	
2	1371773	3640.9	2	22	1080	9993	1800	9993	
3	1371774	22.5	1	13	600	9993	9993	9993	
4	1371775	3710.4	2	22	3600	9993	9997	9993	

5 rows × 24 columns

In [3]:
```
df.shape
```

Out[3]:
```
(75388, 24)
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['SERIAL', 'DENSITY', 'OWNERSHP', 'OWNERSHPD', 'COSTELEC', 'COSTGAS',
       'COSTWATR', 'COSTFUEL', 'HHINCOME', 'VALUEH', 'ROOMS', 'BUILTYR2',
       'BEDROOMS', 'VEHICLES', 'NFAMS', 'NCOUPLES', 'PERNUM', 'PERWT', 'AGE',
       'MARST', 'BIRTHYR', 'EDUC', 'EDUCD', 'INCTOT'],
      dtype='object')
```

In [5]:
```
df.isnull().any(axis=1).sum()
```

Out[5]:
```
0
```

In [6]:
```
#sorting the values based on oldest member of each house and dropping duplicate SERIAL
df = df.sort_values(['SERIAL', 'AGE'], ascending=[True, False])
df=df.drop_duplicates('SERIAL')
df.shape
```

Out[6]:
```
(30802, 24)
```

In [7]:
```
#dropping unnecessary values
df = df.drop(['SERIAL', 'OWNERSHPD', 'COSTELEC', 'COSTGAS', 'COSTWATR', 'COSTFUEL','PE
              'PERWT','VALUEH', 'BUILTYR2','BIRTHYR','EDUCD'], axis=1)
```

In [8]:
```
df.columns
```

Out[8]:
```
Index(['DENSITY', 'OWNERSHP', 'HHINCOME', 'ROOMS', 'BEDROOMS', 'VEHICLES',
       'NFAMS', 'NCOUPLES', 'AGE', 'MARST', 'EDUC', 'INCTOT'],
      dtype='object')
```

In [10]:
```
#replacing MARST AND EDUC values
df['MARST'] = df['MARST'].replace({1:'married,spouse_present', 2:'married,spouse_abser
df['EDUC'] = df['EDUC'].replace({0:'no_school', 1:'Primary_school', 2:'Elementry_schoc
```

In [11]:
```
#converting MARST and EDUC into dummies
df = pd.get_dummies(df, columns=['MARST','EDUC'])
df.columns
```

Out[11]:
```
Index(['DENSITY', 'OWNERSHP', 'HHINCOME', 'ROOMS', 'BEDROOMS', 'VEHICLES',
       'NFAMS', 'NCOUPLES', 'AGE', 'INCTOT', 'MARST_divorced',
       'MARST_married,spouse_absent', 'MARST_married,spouse_present',
       'MARST_separated', 'MARST_single', 'MARST_widow', 'EDUC_1yr_college',
       'EDUC_2yrs_college', 'EDUC_4yrs_college', 'EDUC_5+yrscollege',
       'EDUC_Elementry_school', 'EDUC_Gr10_schooling', 'EDUC_Gr11_schooling',
       'EDUC_Gr12_schooling', 'EDUC_Gr9_schooling', 'EDUC_Primary_school',
       'EDUC_no_school'],
      dtype='object')
```

In [12]:
```
df.shape
```

Out[12]:
```
(30802, 27)
```

In [13]:
```
# Split data into X and y
X = df.drop('OWNERSHP', axis=1)
y = df['OWNERSHP']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=

# Scale the training and testing data
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

LINEAR

In [14]:
```
#model fitting
svm_linear = SVC(kernel='linear',cache_size=1000, verbose = True, max_iter = 100000, r

svm_linear.fit(X_train_scaled, y_train)

accuracy = svm_linear.score(X_test_scaled, y_test)
print("SVM Linear Accuracy :", accuracy)
```

```
[LibSVM]SVM Linear Accuracy : 0.8278683202389455
```

In [15]:
```
#parameter tuning
kfold = skm.KFold(5, random_state=1,shuffle=True)
grid = skm.GridSearchCV(svm_linear,{'C':[0.01,0.1,1,10]},refit=True,cv=kfold,scoring='
grid.fit(X_train_scaled, y_train)
grid.best_params_
```

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
```

Out[15]:
```
{'C': 10}
```

In [16]:
```
#model fitting after parameter tuning
svm_linear = SVC(kernel='linear', C = 10, cache_size=1000, verbose = True, max_iter =

svm_linear.fit(X_train_scaled, y_train)

accuracy = svm_linear.score(X_test_scaled, y_test)
print("SVM Linear Accuracy after parameter tuning:", accuracy)
```

```
[LibSVM]SVM Linear Accuracy after parameter tuning: 0.8272839426011298
```

In [17]:
```
y_pred = svm_linear.predict(X_test_scaled)
confusion_matrix(y_pred, y_test)
```

Out[17]:
```
array([[10083,  1854],
       [  806,  2658]], dtype=int64)
```

In [18]:
```
#finding the importance features
coefficients = svm_linear.coef_
importance_scores = np.abs(coefficients)

importance_df = pd.DataFrame({'Feature': X.columns, 'Importance Score': importance_scc
importance_df = importance_df.sort_values(by='Importance Score', ascending=False)
importance_df
```

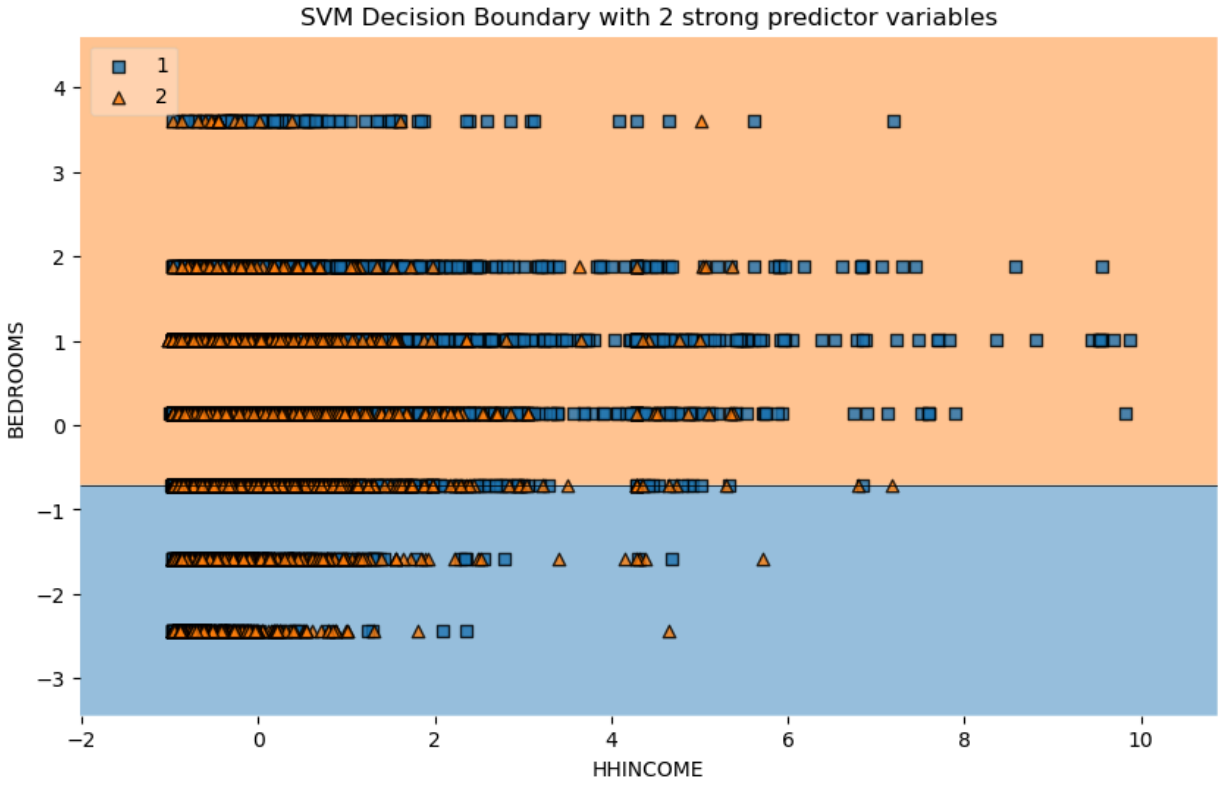
Out[18]:

	Feature	Importance Score
1	HHINCOME	4.477121
3	BEDROOMS	4.199444
5	NFAMS	2.605948
7	AGE	2.300163
2	ROOMS	1.971770
8	INCTOT	1.466320
0	DENSITY	1.109097
12	MARST_separated	0.614624
19	EDUC_Elementry_school	0.542623
11	MARST_married,spouse_present	0.483426
4	VEHICLES	0.451287
17	EDUC_4yrs_college	0.333660
25	EDUC_no_school	0.304212
18	EDUC_5+yrscollege	0.301089
16	EDUC_2yrs_college	0.249928
22	EDUC_Gr12_schooling	0.125846
23	EDUC_Gr9_schooling	0.110602
20	EDUC_Gr10_schooling	0.107615
15	EDUC_1yr_college	0.099714
9	MARST_divorced	0.076073
21	EDUC_Gr11_schooling	0.070955
13	MARST_single	0.051619
6	NCOUPLES	0.028765
24	EDUC_Primary_school	0.025772
14	MARST_widow	0.022449
10	MARST_married,spouse_absent	0.018943

In [19]:
```
#plotting svm with 2 strong predictors
strong_predictors_2 = importance_df.head(2)['Feature'].values
strong_predictors = X[strong_predictors_2]
X_train_strong, X_test_strong, y_train_strong, y_test_strong = train_test_split(strong

scaler_strong = StandardScaler()
X_train_scaled_strong = scaler_strong.fit_transform(X_train_strong)
X_test_scaled_strong = scaler_strong.transform(X_test_strong)

svm_linear_strong = SVC(kernel='linear', C=10, cache_size=1000, verbose=True, max_iter
svm_linear_strong.fit(X_train_scaled_strong, y_train_strong)
```

```
# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_scaled_strong, y_test_strong.to_numpy(), clf=svm_linear_s
plt.xlabel(strong_predictors_2[0])
plt.ylabel(strong_predictors_2[1])
plt.title('SVM Decision Boundary with 2 strong predictor variables')
plt.show()
```

```
[LibSVM]
```



RADIAL

In [33]:
```
#fitting the data with radial kernel
svm_radial = SVC(kernel='rbf', cache_size=1000, verbose = True, max_iter = 100000, rar

svm_radial.fit(X_train_scaled, y_train)

accuracy = svm_radial.score(X_test_scaled, y_test)
print("SVM Radial kernel accuracy :", accuracy)
```

```
[LibSVM]SVM Radial kernel accuracy : 0.8285176287254075
```

In [34]:
```
#parameter tuning
kfold = skm.KFold(5, random_state=1,shuffle=True)
grid = skm.GridSearchCV(svm_radial,{'C':[0.01,0.1,1,10], 'gamma':[0.5,1,2]},refit=True
grid.fit(X_train_scaled, y_train)
grid.best_params_
```

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
```

Out[34]:
```
{'C': 10, 'gamma': 0.5}
```

In [35]:
```
#fitting the model after parameter tuning
svm_radial = SVC(kernel='rbf', C=10, gamma=0.5, cache_size=1000, verbose = True, max_i

svm_radial.fit(X_train_scaled, y_train)

accuracy = svm_radial.score(X_test_scaled, y_test)
print("SVM Radial accuracy after tuning :", accuracy)
```

```
[LibSVM]SVM Radial accuracy after tuning : 0.8318940328550094
```

In [36]:
```
y_pred = svm_radial.predict(X_test_scaled)
confusion_matrix(y_pred, y_test)
```

Out[36]:
```
array([[10271,  1971],
       [  618,  2541]], dtype=int64)
```

In [37]:
```
#important features
result = permutation_importance(svm_radial, X_test_scaled, y_test, n_repeats=5, random
importance_df = pd.DataFrame({'feature': X.columns, 'importance': result.importances_m
importance_df = importance_df.sort_values(by='importance', ascending=False).reset_inde
importance_df
```

Out[37]:

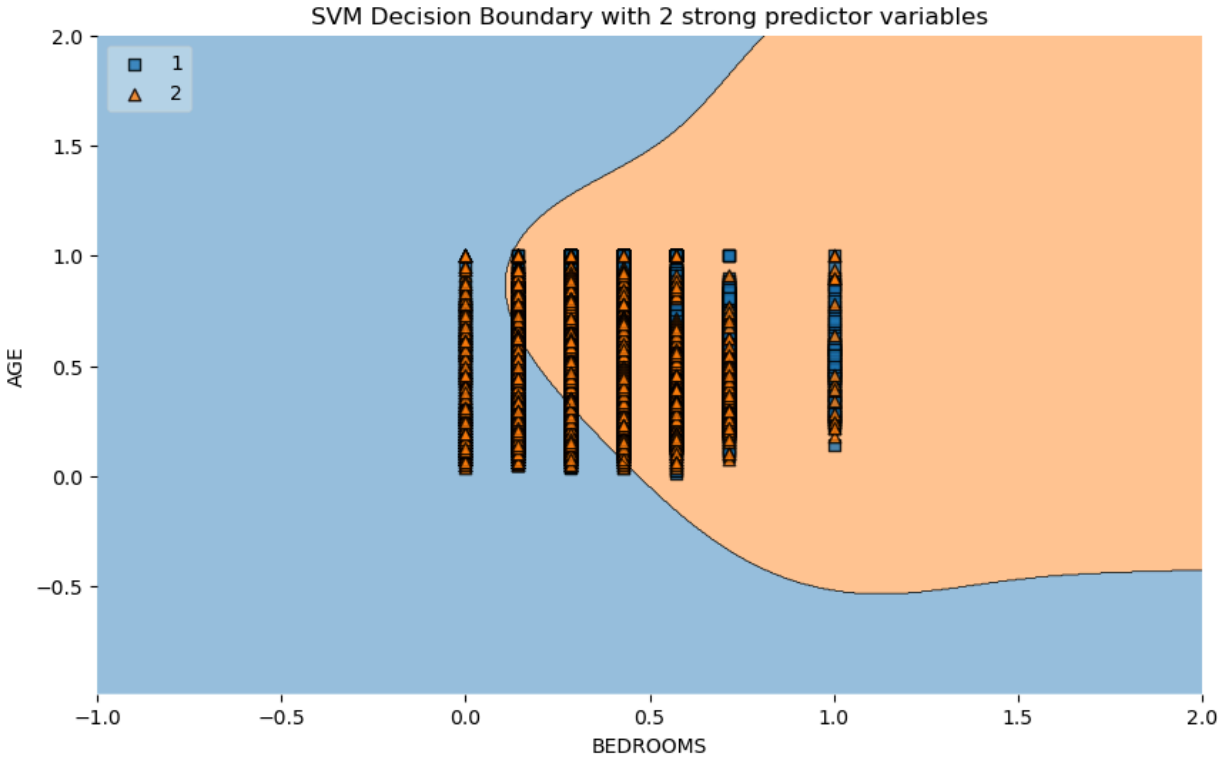
	feature	importance
0	BEDROOMS	0.055165
1	AGE	0.040361
2	ROOMS	0.017674
3	VEHICLES	0.013181
4	INCTOT	0.007337
5	MARST_single	0.005740
6	DENSITY	0.005493
7	MARST_separated	0.004753
8	EDUC_Gr12_schooling	0.004480
9	MARST_divorced	0.003961
10	EDUC_1yr_college	0.003870
11	HHINCOME	0.002857
12	NFAMS	0.002805
13	NCOUPLES	0.002701
14	EDUC_4yrs_college	0.002675
15	MARST_married,spouse_present	0.002402
16	EDUC_5+yrscollege	0.001844
17	EDUC_2yrs_college	0.001844
18	MARST_widow	0.001571
19	EDUC_no_school	0.001143
20	EDUC_Elementry_school	0.001065
21	EDUC_Gr9_schooling	0.000740
22	EDUC_Gr11_schooling	0.000675
23	EDUC_Gr10_schooling	0.000532
24	EDUC_Primary_school	0.000455
25	MARST_married,spouse_absent	0.000117

In [38]:

```
strong_predictors_2 = importance_df.head(2)['feature'].values
strong_predictors = X[strong_predictors_2]
X_train_strong, X_test_strong, y_train_strong, y_test_strong = train_test_split(strong

scaler_strong = MinMaxScaler()
X_train_scaled_strong = scaler_strong.fit_transform(X_train_strong)
X_test_scaled_strong = scaler_strong.transform(X_test_strong)


svm_radial_strong = SVC(kernel='rbf', C=10, gamma=0.5, cache_size=1000, verbose = True
svm_radial_strong.fit(X_train_scaled_strong, y_train_strong)
```

```
# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_scaled_strong, y_test_strong.to_numpy(), clf=svm_radial_s
plt.xlabel(strong_predictors_2[0])
plt.ylabel(strong_predictors_2[1])
plt.title('SVM Decision Boundary with 2 strong predictor variables')
plt.show()
```

```
[LibSVM]
```



POLYNOMIAL

In [26]:
```
#polynomial kernel model fitting
svm_polynomial = SVC(kernel='poly', cache_size=1000, verbose = True, max_iter = 100000
svm_polynomial.fit(X_train_scaled, y_train)
accuracy = svm_polynomial.score(X_test_scaled, y_test)
print("SVM Polynomial kernel accuracy :", accuracy)
```

```
[LibSVM]SVM Polynomial kernel accuracy : 0.8255308096876827
```

In [27]:
```
#parameter tuning
kfold = skm.KFold(5, random_state=1,shuffle=True)
grid = skm.GridSearchCV(svm_polynomial,{'C':[0.01,0.1,1,5], 'degree':[1,2,3]},refit=Tr
grid.fit(X_train_scaled, y_train)
grid.best_params_
```

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
```

Out[27]:
```
{'C': 5, 'degree': 2}
```

In [28]:
```
#model fitting after parameter tuning
svm_polynomial = SVC(kernel='poly', degree=2, coef0=1, C=5, gamma='scale', cache_size=
svm_polynomial.fit(X_train_scaled, y_train)
accuracy = svm_polynomial.score(X_test_scaled, y_test)
print("SVM Polynomial kernel accuracy after tuning :", accuracy)
```

```
[LibSVM]SVM Polynomial kernel accuracy after tuning : 0.830400623336147
```

In [29]:
```
y_pred = svm_polynomial.predict(X_test_scaled)
confusion_matrix(y_pred, y_test)
```

Out[29]:
```
array([[10261,  1984],
       [  628,  2528]], dtype=int64)
```

In [30]:
```
result = permutation_importance(svm_polynomial, X_test_scaled, y_test, n_repeats=5, ra
importance_df = pd.DataFrame({'feature': X.columns, 'importance': result.importances_m
importance_df = importance_df.sort_values(by='importance', ascending=False).reset_inde
importance_df
```

Out[30]:

	feature	importance
0	BEDROOMS	5.939874e-02
1	AGE	3.960782e-02
2	MARST_married,spouse_present	2.376469e-02
3	ROOMS	1.372638e-02
4	EDUC_4yrs_college	9.012402e-03
5	VEHICLES	6.506071e-03
6	INCTOT	5.545094e-03
7	EDUC_5+yrscollege	5.363288e-03
8	HHINCOME	3.155639e-03
9	DENSITY	2.817999e-03
10	NFAMS	2.545289e-03
11	NCOUPLES	2.285566e-03
12	MARST_widow	1.831050e-03
13	EDUC_2yrs_college	1.818064e-03
14	MARST_single	1.714174e-03
15	EDUC_1yr_college	9.220181e-04
16	MARST_divorced	8.311149e-04
17	EDUC_Gr12_schooling	6.882670e-04
18	EDUC_no_school	4.155574e-04
19	EDUC_Primary_school	1.038894e-04
20	MARST_separated	1.298617e-05
21	MARST_married,spouse_absent	-6.661338e-17
22	EDUC_Elementry_school	-3.895851e-05
23	EDUC_Gr10_schooling	-1.428479e-04
24	EDUC_Gr9_schooling	-2.337511e-04
25	EDUC_Gr11_schooling	-4.934744e-04

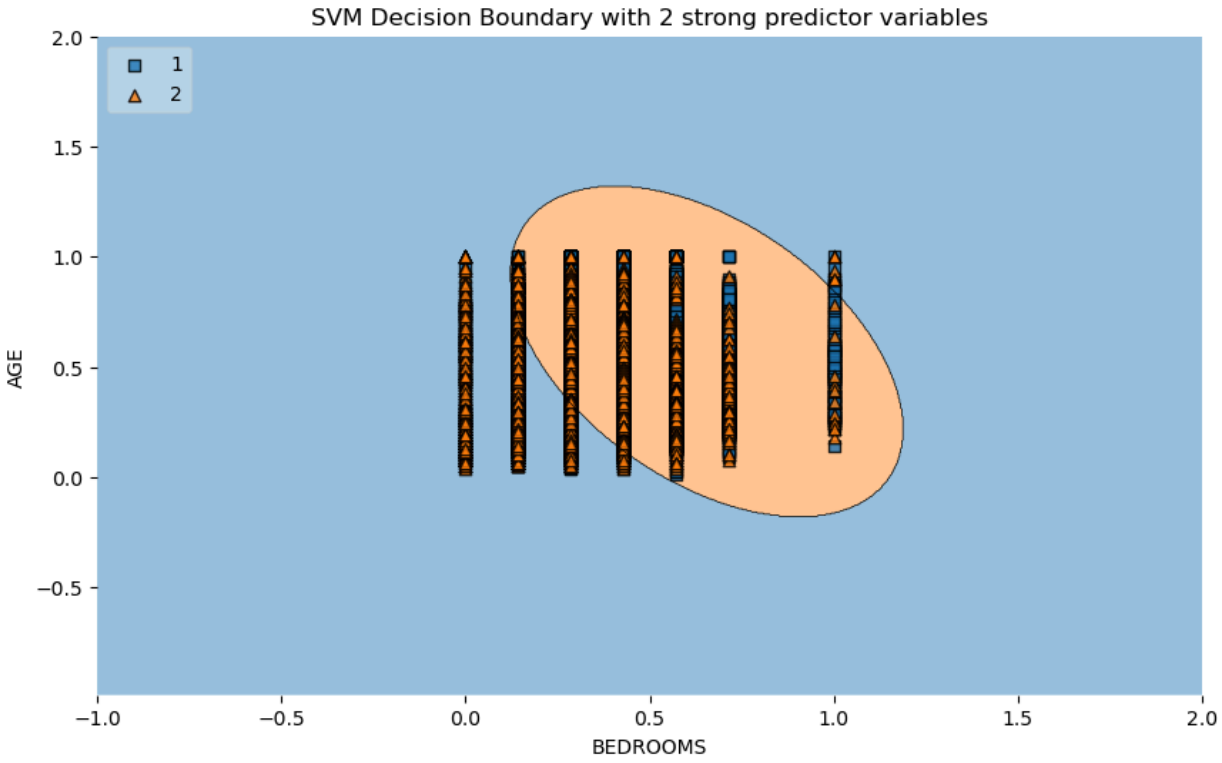
In [32]:

```
strong_predictors_2 = importance_df.head(2)['feature'].values
strong_predictors = X[strong_predictors_2]
X_train_strong, X_test_strong, y_train_strong, y_test_strong = train_test_split(strong

scaler_strong = MinMaxScaler()
X_train_scaled_strong = scaler_strong.fit_transform(X_train_strong)
X_test_scaled_strong = scaler_strong.transform(X_test_strong)

# Train the best model with the top 2 predictor variables
svm_polynomial_strong = SVC(kernel='poly', degree=2, coef0=1, C=5, gamma='scale', cach
svm_polynomial_strong.fit(X_train_scaled_strong, y_train_strong)
```

```
# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_scaled_strong, y_test_strong.to_numpy(), clf=svm_polynomi
plt.xlabel(strong_predictors_2[0])
plt.ylabel(strong_predictors_2[1])
plt.title('SVM Decision Boundary with 2 strong predictor variables')
plt.show()
```

```
[LibSVM]
```

```
C:\Users\KARRA VIVEK REDDY\anaconda3\Lib\site-packages\sklearn\svm\_base.py:297: Conv
ergenceWarning: Solver terminated early (max_iter=1000000).  Consider pre-processing
your data with StandardScaler or MinMaxScaler.
  warnings.warn(
```



In [ ]: