STATISTICAL MACHINE LEARNING 2
SUPPORT VECTOR MACHINE

Nidhi Trivedi

Abstract

This report focuses on the detailed study of data from US Census in Washington State that tells us information about their age, income, number of family member, education level, housing costs, total family income and other related details. The goal is to consider all these factors and predict if the home is being owned by the people living there or is it rented. For this analysis we will be using different SVM models like linear, radial and polynomial. The report explains how SVC model works, how best values for parameters will be considered for tuning models and other important details. There will also be graphs showing decision boundaries, which will illustrate how the models classify the data. We will also explore the relationships between different variables to check which ones are the best predictors.

Introduction

Owing a home is a dream for many families but is equally challenging to achieve for some families and individuals. This analysis aims to better understand the factors that allow people to become homeowners or make it harder for them. We are looking at data from the US Census in Washington State that includes information like people's age, income, education level, housing costs, household size, and other details. This data gives us a detailed picture of people's living situations. We will be implying support vector classifier to analyse this data to predict if the home is owned by residents or is rented. Through this analysis we will be understanding the important factors that plays important role in homeownership and can be useful in various ways. People looking to buy a home can learn what contributes to their dream of having own home, so they can plan accordingly. Policy makers can create better housing programs and can make a plan about allocating resources to make homeownership more accessible for different groups of people. Overall, this report sheds light on the complex relationship between people’s situations and their ability to own homes.

Overview

This report looks at whether people own or rent their homes using data from the US Census in Washington State. We are looking at things like how much people earn, how old they are and how many family members are there, to check if we can predict whether they own their homes or rent them. We will be using different SVC models like linear which looks at things in a straight line, radial which looks at things in a circle-like pattern and polynomial which looks at things in a curvy. In report we will be explaining how these SVM models work and how are we tuning them on the basis of best parameters. We have included some plots as well to understand how this decision making is done. With this analysis we are aiming to figure out which factors are important in terms of buying a house like what is the family income in a household, how big their family is , can be considered as important factors. The information can be helpful for people who want to buy a home, can understand how much efforts required from them. The goal is to make things clearer for everyone, so we all can work together and make sure everyone has good place to live.

Theoretical Background

For this analysis we are using a powerful machine learning model which is Support Vector Machines (SVMs) that can be used for both classification and regression problems. They are build to separate data into different groups or categories. This can be done by finding the best line that divides the groups. In our case we are using it to predict whether a home is owned or rented based on various factors. There are three types of SVC model we are going to implement **Linear** SVC which tries to find the best straight line that separates the two classes. It is best suitable when data is linearly separable, that is a single line can clearly divide the owned and rented house. **Radial** SVC is another type of SVC that uses a curved decision boundary, but it's shaped like a circle or oval. It can often capture more complex patterns in the data. **Polynomial** SVC is used when the data isn't linearly separable, and can be separated by a curved line. We can execute the model by setting the correct **kernel** parameter of SVC which helps to select the hyperplane to be used to separate the data. For linear kernel is set to "linear", for radial it is "rbf" and for polynomial model it is "poly".

Apart from selecting the kernel, there are several other important parameters that needs to be specified. These parameter controls the performance of SVC model and proper tuning is essential to achieve optimal results. One of the key parameter is '**C'** called as regularization parameter. This parameter decides how strict or lenient model will be when trying to specify data points in the training set. A high value means the model is very strict and try to fit every single data point in the training set. Low value makes it more lenient. Another critical parameter which is '**gamma'** which is for non-linear kernel(poly and radial). It is for curved decision boundaries, it controls the importance each individual has on where the curved line gets drawn. Data points very close to the boundary line have big influence and the ones that are far away don't matter much. We have additional parameter for polynomial kernel, '**degree'**. The decision boundary it creates is a curved line. It controls how bent or complex this curve can be. A low value of this means the curve has to be relatively simple, but a high degree lets the curve bend and twist a lot more.

To find the best and optimal combination of these parameters, we perform **grid search**. In this approach we provide a range of values for parameter is specified, and then SVC model is trained and evaluated using a technique called cross-validation. In this model is trained on part of data and then tested on another part to check how well it performs. After performing for all combinations, grid search looks at the performance of each set based on metrics like accuracy. It returns the best parameter that gives the best performance. Now we fit the optimal SVC model using these set of parameters. There are some important parameters of grid search that we have used , that is **estimator** which specifies the model to use, scoring defined the evaluation metrics, **refit** which determines if the best estimator is to be fitted on the entire dataset. Overall, grid search is a simple technique for optimizing model performance without the need for manual tuning.

Methodology

The data we have used for analysis is US Census in Washington State. The data was collected via the US Census and was accessed through IPUMS USA. The dataset contains 24 variables and 75388 rows, each representing different aspects of people and their housing situations. For this our target variable is **OWNERSHP**(owned/rented). The owned is represented by 1 and rented is 2.

Data Cleaning and Pre-processing

Before building out model we will start by cleaning our data and organize it well. First, we looked at all the columns in the data to understand what kind of information they contained. We found that there were no missing values, which was good. The dataset contains information about dwellings that have multiple people living in them. Each dwelling is identified by a unique number called **SERIAL**. Within each dwelling, there are individuals identified by their **PERNUM**.

We have multiple entries for single dwelling, our goal is to predict home ownership status at the home level, not at person level, so we will combine these multiple entries for each home into a single entry. For this we aggregated the entries on the basis of dwelling number , and fetched max of age for that dwelling. After performing this we were left with 30,802 unique rows.

Next, we selected only the columns (types of information) that we considered relevant for predicting homeownership status. We created a new data table containing columns such as household income, number of vehicles, population density in the area, marital status, education level, and other potentially relevant factors. For MARST(marital status) we have created separate fields for Married, divorced and unmarried. For EDUC(Education) we have created separate fields for students in less than 6th grade and above. After completing all the cleaning and pre-processing steps, we have final data frame ready for analysis. This includes columns such as 'OWNERSHP', 'HHINCOME', 'BEDROOMS', 'NFAMS', 'AGE', 'MARRIED', 'EDUC_CLG' columns in it.

With this cleaned and processed data, we were ready to start building and testing our different Support Vector Classifier (SVC) models for predicting homeownership status based on the various socioeconomic factors present in the data.

Computational Results

We have built 6 models in total which includes SVC Linear model , SVC radial model and SVC polynomial model . For each of the model we have performed grid search to find out the best parameter values for cost, gamma and degree. Using those values we have created cross validated model respectively.

Below are the computation results for every model :

For Linear model –
Accuracy of svm linear model : 0.8256776497321864
With optimised cost parameter as C : 0.1
The accuracy is still the same 0.8256776497321864.

For Radial Model-
Accuracy of radial model: 0.8188605745820484
The optimised parameter for radial model is C : 1 and gamma : 0.5
Accuracy of radial best model: 0.8285992533679597

For Polynomial Model-
Accuracy of polynomial model: 0.8263268949845805
The optimised parameter for polynomial model is C : 3 and degree : 2
Accuracy of polynomial best model: 0.8307093004382405

Important Features

For Linear model

	Features	Score
1	BEDROOMS	0.799800
4	MARRIED	0.491996
0	HHINCOME	0.400400
2	NFAMS	0.400009
5	EDUC_CLG	0.153665
3	AGE	0.030780

For radial model

	Features	Score
1	BEDROOMS	0.123331
3	AGE	0.073617
0	HHINCOME	0.012053
4	MARRIED	0.009821
2	NFAMS	0.006696
5	EDUC_CLG	0.004099

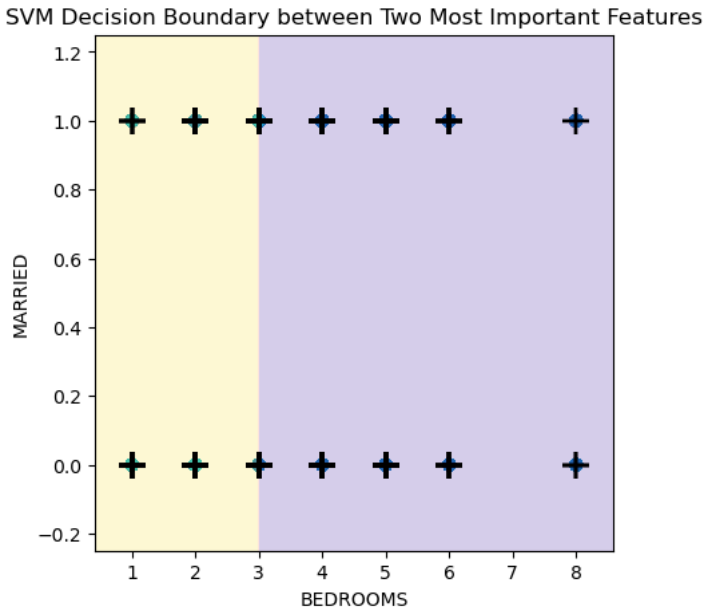
For Polynomial model

	Features	Score
7	AGE	0.046711
3	BEDROOMS	0.044357
2	ROOMS	0.018262
9	EDUC	0.005966
4	VEHICLES	0.003815
1	HHINCOME	0.001948
8	MARST	0.001745

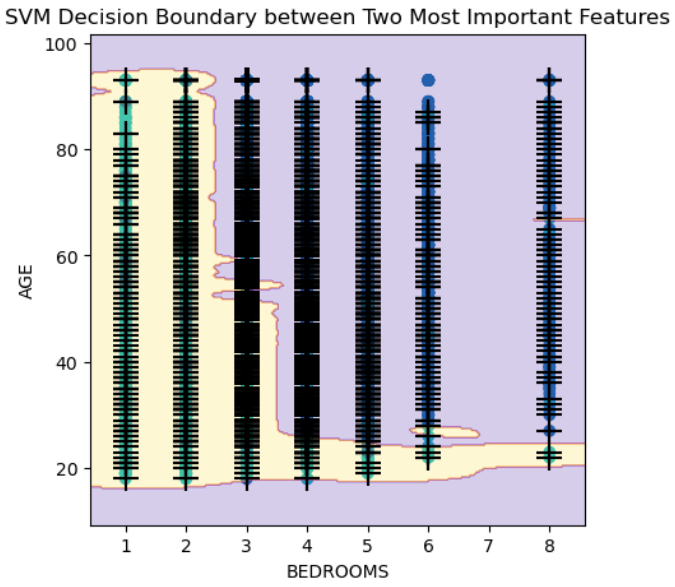
From all the above important features tables , we can observe that Married, Age and Bedrooms features are the most important factors in all the three models.

SVM graphs

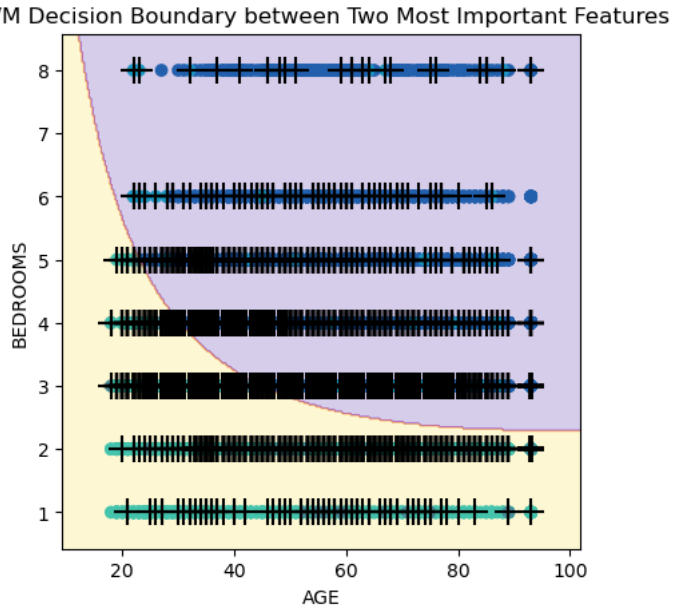
For Linear model



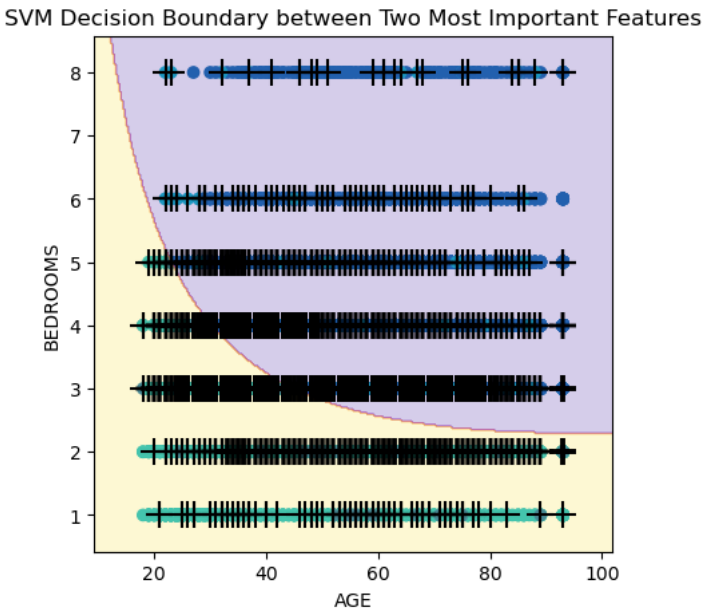
For Radial model



For polynomial



Discussion



The graph plots age and number of bedrooms, which are features used for predicting whether a house is rented or owned. With the decision boundary we can understand that houses with more number of bedrooms are more suitable for families.

From the analysis we have observed that Household income is the important feature in terms of owning a home. As company with more earning member are likely to have their owned home. They can afford down payments, qualify for loans. Education, in terms of attending college degree is also strong predictors, as higher education can lead to employment.

Buying a home has become difficult for low-moderate income households. Our analysis highlights the key factors contributing to owning a home, so buyers can stay more focused towards their end goal.

The policy makers should prioritise the development of affordable housing by providing low interest loans and down payment assistance. Financial education programs should be conducted so that it could help people plan their financial needs accordingly.

Interpretation of computational results

From the computational results above, we can find that tuning the settings has helped the non linear models to improve the accuracy but not the linear model. We can observe that polynomial model with C = 3 and degree = 2 performed the best of all

models with 83.1% accuracy. Not extremely high accuracy , but these can help us understand what factors mainly influence homeownership.

Conclusion

Based on our analysis it is clear that certain factors strongly contributes to whether someone owns or rent their home. In our analysis we have identified that key variables like income, age, marital status are important predictors of home ownership. People with good education and better income can afford home. Number of earning members in the family , can contribute to overall household income , which can lead to buying a home. Marital status is another aspect , if a couple is working there are high chances of them getting a home of their own than renting it. It is important for everyone to have a fair chance to own their home. By looking at these important things and being careful with our money, one can make it happen. And if we all work together to make housing fair and affordable, we can help more people achieve their dream of owning a home.

Bibliography/References

1. https://scikit-learn.org/stable/modules/svm.html
2. https://scikitlearn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Appendix

```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots, cm
import sklearn.model_selection as skm
from ISLP import load_data, confusion_table
from sklearn.svm import SVC
from ISLP.svm import plot as plot_svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
```

In [ ]:

```
dwelling_detail = pd.read_csv('/Users/nidhitrivedi/Downloads/Housing.csv')
```

In [ ]:

```
dwelling_detail.head(5)
```

In [ ]:

```
dwelling_detail.shape
```

In [ ]:

```
unique_value_counts = dwelling_detail.nunique()

print("Number of unique values:")
print(unique_value_counts)
```

In [ ]:

```
dwelling_detail =
dwelling_detail.loc[dwelling_detail.groupby('SERIAL')['AGE'].idxmax()]
```

In [ ]:

```
dwelling_detail.shape
```

In [ ]:

```
dwelling_detail.head(5)
```

In [ ]:

```
#dropping the unnecessary columns
required_columns = ['DENSITY', 'OWNERSHP', 'HHINCOME', 'ROOMS',
                    'BEDROOMS', 'VEHICLES', 'NFAMS', 'NCOUPLES', 'AGE',
'MARST', 'EDUC']

dwelling_detail_selected = dwelling_detail[required_columns]

dwelling_detail_selected.head(5)
```

In [ ]:

```
dwelling_detail_selected.shape
```

In [ ]:

```
max_hhincome = dwelling_detail_selected['HHINCOME'].max()
max_rooms = dwelling_detail_selected['ROOMS'].max()
max_BEDROOMS = dwelling_detail_selected['BEDROOMS'].max()
max_VEHICLES = dwelling_detail_selected['VEHICLES'].max()
max_NFAMS = dwelling_detail_selected['NFAMS'].max()
max_ncouples = dwelling_detail_selected['NCOUPLES'].max()
max_density = dwelling_detail_selected['DENSITY'].max()


print("Maximum value of HHINCOME:", max_hhincome)
print("Maximum value of max_rooms:", max_rooms)
print("Maximum value of max_BEDROOMS:", max_BEDROOMS)
print("Maximum value of max_VEHICLES:", max_VEHICLES)
print("Maximum value of max_NFAMS:", max_NFAMS)
print("Maximum value of max_ncouples:", max_ncouples)
print("Maximum value of max_density:", max_density)
```

In [ ]:

```
dwelling_detail_selected['HHINCOME'] = [0 if value >= -7100 and value <=
100000 else
                                        (1 if value > 100000 and value
<= 700000 else
                                         (2 if value > 700000 else 3))
                                        for value in
dwelling_detail_selected['HHINCOME']]
print(dwelling_detail_selected['HHINCOME'].value_counts())


dwelling_detail_selected['VEHICLES'] = [0 if value == 9 else
                                       (1 if value > 0 and value <= 3 else
                                        (2 if value > 4 and value <= 7
else 3))
                                       for value in
dwelling_detail_selected['VEHICLES']]
print(dwelling_detail_selected['VEHICLES'].value_counts())

dwelling_detail_selected['MARRIED'] = (dwelling_detail_selected['MARST'] <=
2).astype(int)
dwelling_detail_selected['DIVORCED'] =
dwelling_detail_selected['MARST'].between(2, 5).astype(int)
```

```
dwelling_detail_selected['UNMARRIED'] = (dwelling_detail_selected['MARST']
> 4).astype(int)


dwelling_detail_selected['EDUC_SCHOOL'] = (dwelling_detail_selected['EDUC']
<= 6).astype(int)
dwelling_detail_selected['EDUC_CLG'] = (dwelling_detail_selected['EDUC']
>6).astype(int)

dwelling_detail_selected['DENSITY'] = [0 if value >= 0 and value <= 5000
else
                                       (1 if value >5000 and value <=
10000 else 2)
                                       for value in
dwelling_detail_selected['DENSITY']]
print(dwelling_detail_selected['DENSITY'].value_counts())
```

In [ ]:

```
dwelling_detail_selected.head(5)
```

In [ ]:

```
dwelling_detail_selected.shape
```

In [ ]:

```
dwelling_detail_selected.columns
```

will be creating subset of columns for linear analysis For Linear model we will be considering

In [ ]:

```
required_columns = ['OWNERSHP', 'HHINCOME',
                    'BEDROOMS', 'NFAMS', 'AGE', 'MARRIED', 'EDUC_CLG']

dwelling_detail_linear = dwelling_detail_selected[required_columns]

dwelling_detail_linear.head(5)
```

In [ ]:

```
dwelling_detail_linear.shape
```

In [ ]:

```
dwelling_detail_linear.columns
```

this is our SVC linear model

In [ ]:

```
X = dwelling_detail_linear.drop('OWNERSHP', axis=1)
y = dwelling_detail_linear['OWNERSHP']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

svm_linear = SVC(kernel="linear", C=0.4)
svm_linear.fit(X_train, y_train)
```

In [ ]:

```
ypredict_linear = svm_linear.predict(X_test)
test_accuracy = accuracy_score(y_test, ypredict_linear)
print("Accuracy of svm linear model :", test_accuracy)
```

In [ ]:

```
kfold_linear = skm.KFold(3,
                  random_state=0,
                  shuffle=True)
grid = skm.GridSearchCV(svm_linear,
                        {'C':[0.1,0.5,1,3,5,6]},
                        refit=True,
```

```
                        cv=kfold_linear,
                        scoring='accuracy');
grid.fit(X_train, y_train)
grid.best_params_
```

In [ ]:

```
svm_linear_best = SVC(kernel="linear", C=0.1)
svm_linear_best.fit(X_train, y_train)

ypredict_linear_best = svm_linear_best.predict(X_test)
test_accuracy_linear_best = accuracy_score(y_test, ypredict_linear_best)
print("Accuracy of svm linear best model :", test_accuracy_linear_best)


coeff = svm_linear_best.coef_
importance_scores = abs(coeff)
df_importance_feature = pd.DataFrame({'Features': X_train.columns, 'Score':
importance_scores[0]})
df_importance_feature = df_importance_feature.sort_values(by='Score',
ascending=False)
print(df_importance_feature)
```

In [ ]:

```
svm_linear_best = svm_linear_best.fit(X_train[['BEDROOMS', 'MARRIED']],
y_train)

fig, ax = plt.subplots(figsize=(5, 5))
plot_svm(X_train[['BEDROOMS', 'MARRIED']], y_train, svm_linear_best, ax=ax)
plt.xlabel('BEDROOMS')
plt.ylabel('MARRIED')
plt.title('SVM Decision Boundary between Two Most Important Features')
plt.show()
```

This is radial model

In [ ]:

```
X = dwelling_detail_linear.drop('OWNERSHP', axis=1)
y = dwelling_detail_linear['OWNERSHP']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

svm_rbf = SVC(kernel="rbf", gamma=2, C=0.5)
svm_rbf.fit(X_train, y_train)
```

In [ ]:

```
ypredict_radial = svm_rbf.predict(X_test)
test_accuracy = accuracy_score(y_test, ypredict_radial)
print("Accuracy of radial model:", test_accuracy)
```

In [ ]:

```
kfold_radial = skm.KFold(5,
                  random_state=0,
                  shuffle=True)
grid = skm.GridSearchCV(svm_rbf,
                        {'C':[0.1,1,5,7,10],
                         'gamma':[0.5,1,3,5]},
                        refit=True,
                        cv=kfold_radial,
                        scoring='accuracy');
grid.fit(X_train, y_train)
grid.best_params_
```

The best param for this model is C = 1 and gamma = 0.5.

In [ ]:
```
svm_radial_best = SVC(kernel="rbf", gamma=0.5, C=1)
svm_radial_best.fit(X_train, y_train)
ypredict_radial_best = svm_radial_best.predict(X_test)
test_accuracy_radial_best = accuracy_score(y_test, ypredict_radial_best)
print("Accuracy of radial best model:", test_accuracy_radial_best)
```

In [ ]:
```
from sklearn.inspection import permutation_importance
perm_importance = permutation_importance(svm_radial_best, X_train, y_train,
n_repeats=1, random_state=42)
importance_scores = perm_importance.importances_mean
df_importance_feature = pd.DataFrame({'Features': X_train.columns, 'Score':
importance_scores})
df_importance_feature = df_importance_feature.sort_values(by='Score',
ascending=False)
print(df_importance_feature)
```

In [ ]:
```
top_features = df_importance_feature.iloc[:2]['Features'].tolist()
svm_radial_best.fit(X_train[top_features], y_train)

fig, ax = plt.subplots(figsize=(5, 5))
plot_svm(X_train[top_features], y_train, svm_radial_best, ax=ax)
plt.xlabel(top_features[0])
plt.ylabel(top_features[1])
plt.title('SVM Decision Boundary between Two Most Important Features')
plt.show()
```

This is poly model

In [ ]:
```
X = dwelling_detail_selected.drop('OWNERSHP', axis=1)
y = dwelling_detail_selected['OWNERSHP']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

In [ ]:
```
svm_poly = SVC(kernel="poly", degree=3, C=0.5)
svm_poly.fit(X_train, y_train)
ypredict_poly = svm_poly.predict(X_test)
test_accuracy = accuracy_score(y_test, ypredict_poly)
print("Accuracy of polynomial model:", test_accuracy)
```

In [ ]:
```
kfold_poly = skm.KFold(5,
                  random_state=0,
                  shuffle=True)
grid = skm.GridSearchCV(svm_poly,
                        {'C':[0.1,1,2,3],
                         'degree':[2,3,4]},
                        refit=True,
                        cv=kfold_poly,
                        scoring='accuracy');
grid.fit(X_train, y_train)
grid.best_params_
```

In [ ]:
```
svm_poly_best = SVC(kernel="poly", degree=2, C=2)
svm_poly_best.fit(X_train, y_train)
ypredict_best_poly = svm_poly_best.predict(X_test)
```

```
test_accuracy = accuracy_score(y_test, ypredict_best_poly)
print("Accuracy of polynomial best model:", test_accuracy)
```

In [ ]:

```
perm_importance = permutation_importance(svm_poly_best, X_train, y_train,
n_repeats=1, random_state=42)
importance_scores = perm_importance.importances_mean
df_importance_feature = pd.DataFrame({'Features': X_train.columns, 'Score':
importance_scores})
df_importance_feature = df_importance_feature.sort_values(by='Score',
ascending=False)
print(df_importance_feature)
```

In [ ]:

```
top_features = df_importance_feature.iloc[:2]['Features'].tolist()
svm_poly_best.fit(X_train[top_features], y_train)

fig, ax = plt.subplots(figsize=(5, 5))
plot_svm(X_train[top_features], y_train, svm_poly_best, ax=ax)
plt.xlabel(top_features[0])
plt.ylabel(top_features[1])
plt.title('SVM Decision Boundary between Two Most Important Features')
plt.show()
```