DATA 5322 Statistical Machine Learning II

Spring Quarter 2024

Vatsal Dalal

April 29, 2024

Predicting Homeownership with Support Vector Models: A Washington State Census Analysis

Abstract:

This report examines the use of support vector machines (SVMs) for predicting homeownership based on demographic and housing data from the US Census in Washington State. Utilizing key variables such as income, age, education, electricity cost, and construction year, we explore the performance of different SVM kernels: linear, polynomial, and radial basis function (RBF). The RBF SVM emerged as the best model, achieving the 84% accuracy in predicting whether a home is owned or rented. A plot of the SVM decision boundary visualizes the classifier's performance, demonstrating its effectiveness with two significant predictor variables. This project shows that SVMs can be a useful tool for predicting homeownership, offering insights into the factors that drive homeownership trends.

Introduction:

Support Vector Machines (SVMs) are popular tools in machine learning, used to classify and predict different outcomes. In this report, we use SVMs to classify whether a home is owned by a homeowner or rented by a tenant based on various demographic and housing-related factors. Our goal is to understand what factors are most important in determining whether someone owns their home or rents it.

The dataset used in this project is from the United States Census Bureau, accessed through IPUMS USA, containing information about people and their housing situations. It includes variables such as the number of rooms, utility costs (water, gas, electricity), and personal details like age, income, and marital status. The primary target variable is OWNERSHP, indicating whether a home is owned (1) or rented (2). Each household has a unique identifier (SERIAL), and each person in a household has a unique number (PERNUM). In total, the dataset has 24 variables and 75,388 observations. Our task is to use support vector machines (SVMs) to predict whether a home is owned or rented based on these variables. Given the data's complexity, we preprocess it to select the most relevant features and employ different SVM kernels—linear, polynomial, and radial basis function (RBF)—to build classification models. This approach helps us identify key factors that influence homeownership, offering insights into trends that could be valuable for policymakers dealing with housing affordability and access.

Theoretical Background:

Support Vector Machines (SVMs) are robust supervised learning algorithms used in classification and regression tasks. They aim to find a hyperplane that optimally separates different classes, allowing for a flexible approach to various data distributions. This section discusses SVM with linear, polynomial, and radial basis function (RBF) kernels and outlines the concept of permutation importance for feature selection.

1. Linear Kernel:

The linear kernel is the simplest form of SVM, assuming that the data is linearly separable. It finds a straight-line hyperplane (or its higher-dimensional equivalent) that divides the data into classes while maximizing the margin. Linear kernels are computationally efficient and well-suited for large datasets with high-dimensional features. They are often the first choice for SVM when data has a clear linear structure.

2. Radial Basis Function (RBF) Kernel:

The RBF kernel, also known as the Gaussian kernel, is used when the data cannot be linearly separated. It maps the data into a high-dimensional space using a Gaussian function, allowing SVM to create complex, non-linear decision boundaries. The RBF kernel calculates similarity based on the Euclidean distance between points, making it versatile for various data structures. Its flexibility allows SVM to handle intricate patterns, but it requires careful tuning to avoid overfitting.

3. Polynomial Kernel:

The polynomial kernel introduces polynomial terms to the hyperplane, enabling SVM to model non-linear relationships. It maps the data into a higher-dimensional space using a polynomial function, allowing for more complex decision boundaries. The polynomial kernel's degree parameter (degree of the polynomial) controls the level of complexity. While polynomial kernels offer flexibility, they can increase the risk of overfitting if not tuned properly.

4. Tuning Parameters:

SVMs require tuning of several parameters to achieve optimal performance. The primary parameters include:

Regularization Parameter (C): This parameter controls the trade-off between achieving a low training error and a low testing error. A higher C value places greater emphasis on correct classification during training, which might lead to overfitting, while a lower C value promotes a wider margin, potentially increasing generalization.

Kernel Parameter (γ for RBF kernel): For RBF kernels, the parameter γ controls the width of the Gaussian function. A smaller γ value leads to a wider kernel, resulting in smoother decision boundaries, while a larger γ value produces narrower kernels and more complex boundaries. Finding the right balance is crucial to avoid overfitting or underfitting.

Degree Parameter (Degree for Polynomial Kernel): For polynomial kernels, the degree parameter controls the complexity of the polynomial. A higher degree results in more complex decision boundaries, allowing the model to fit more complex patterns. This can lead to overfitting if the degree is too high. A lower degree produces simpler decision boundaries, potentially reducing overfitting risk but risking underfitting if too low.

5. Permutation Importance for Feature Selection:

Permutation importance is a technique used to assess the importance of features in a model by measuring the impact on model performance when the values of a specific feature are randomly shuffled. This technique provides a straightforward way to identify which features have the most significant influence on the model's predictions. Permutation importance is especially useful for SVM because it allows us to determine which features contribute most to classification accuracy. This insight can guide feature selection, allowing us to focus on the most relevant variables, thereby improving model performance and interpretability.

Methodology:

1. Data Preprocessing:

We are building the svm model to classify whether home is owned or rented based on various demographic and housing-related factors. The dataset contained 75,388 rows and 24 columns. This large dataset had to be cleaned up.

To start, we focused on ensuring each household had only one entry, representing the oldest person. This was achieved by sorting the data by 'SERIAL' (a unique household identifier) and 'AGE'(person age), then keeping the oldest record for each 'SERIAL', removing any duplicates. We also removed some columns that weren't needed for analysis, like 'OWNERSHPD'(detail status of ownership), 'BIRTHYR(birth year of the person)', 'VALUEH'(value of home), 'EDUC'(Educational attainment), 'EDUCD'(Educational attainment detailed), 'PERWT'(person weight), and 'PERNUM'(person number in the house).

After that, we removed any invalid or zero values from the data to clean it up. In order to make the rows less useful for analysis, the values for "ROOMS" (the number of rooms in the house), "BUILTYR2" (the age of the structure), "BEDROOMS" (the number of bedrooms in the house), and "VEHICLES" (the number of vehicles in the house) were removed. Rows with age values of 999 and marital status values of 9 were also eliminated because they appeared to be placeholder or incorrect values.

We mapped the numerical codes of the 'MARST' (marital status) column to categories such as 'Married,' 'Separated/Divorced,' and 'Single' to ease data transformation. Furthermore, we took 'MARST' and turned it into distinct columns that represented each category using binary values, creating "dummy" columns.

After all these steps, the dataset shrank to 30,802 rows and 18 columns. This reduction was substantial, indicating that a lot of unnecessary or incorrect data had been removed. Then I separated the features and the target variable. To do this, we dropped the 'OWNERSHP' column from

the features dataset (denoted as X), keeping only the target variable in y for subsequent modeling tasks: then making it ready for further analysis or machine learning tasks.

2. SVM with Linear Kernel:

For linear SVM, I split the data into 70:30 ratio for model training, the data is standardized using **StandardScaler**. The scaler is fitted to the training data and then applied to both the training and testing subsets to ensure consistent scaling across all features.

After, a hyperparameter tuning process is conducted using GridSearchCV to optimize the SVM model's parameters, specifically focusing on the regularization parameter C with a linear kernel. This involves creating a parameter grid with different C values and applying 10-fold cross-validation to identify the optimal combination for the highest accuracy. The best parameters and model are obtained from GridSearchCV, and the resulting model is evaluated for accuracy on the scaled test set.

Then Feature importance is assessed using permutation importance to determine which features contribute most significantly to the model's performance. a new SVM model is trained using only the top 5 features identified through permutation importance, and the accuracy is calculated on the test set. This reduced-feature model is further analyzed through a confusion matrix to understand its prediction capabilities.

For plotting decision boundaries for the linear SVM with the top 2 features, the new SVM model trained.

3. SVM with radial kernel:

The SVM with a radial kernel follows a process similar to that of a linear SVM. However, the key difference is that hyperparameter tuning is performed using GridSearchCV, which employs a parameter grid that includes both C (regularization) and gamma (the width of the RBF kernel). A 10-fold cross-validation is used to identify the best combination of C and gamma for achieving the highest accuracy. Once the optimal model is obtained, it is evaluated on a test set to measure its accuracy, with a focus on analysis through feature importance, confusion matrices, and reduced-feature analysis.

4. SVM with Polynomial Kernel:

The SVM with a polynomial kernel follows a process similar to that of a linear SVM. However, the key difference is that hyperparameter tuning is carried out using GridSearchCV, which uses a parameter grid that includes both C (regularization) and degree (the degree of the polynomial kernel). A 10-fold cross-validation is used to find the best combination of C and degree for the highest accuracy. Once the optimal model is obtained, it is evaluated on a test set to measure its accuracy, with a focus on analysis through feature importance, confusion matrices, and reduced-feature analysis.

Computational Results:

1. Linear SVM:

Below the Fig. 1: shows the linear Support Vector Machine (SVM) model were derived using a **GridSearchCV** approach with a range of regularization values: 'C': [0.01, 0.1, 1, 10], and the kernel = 'linear'. The optimal parameter combination was found to be C = 1, achieving an **accuracy of 85.07%** on the test set. This high accuracy demonstrates the model's effectiveness in separating the classes with a linear decision boundary.

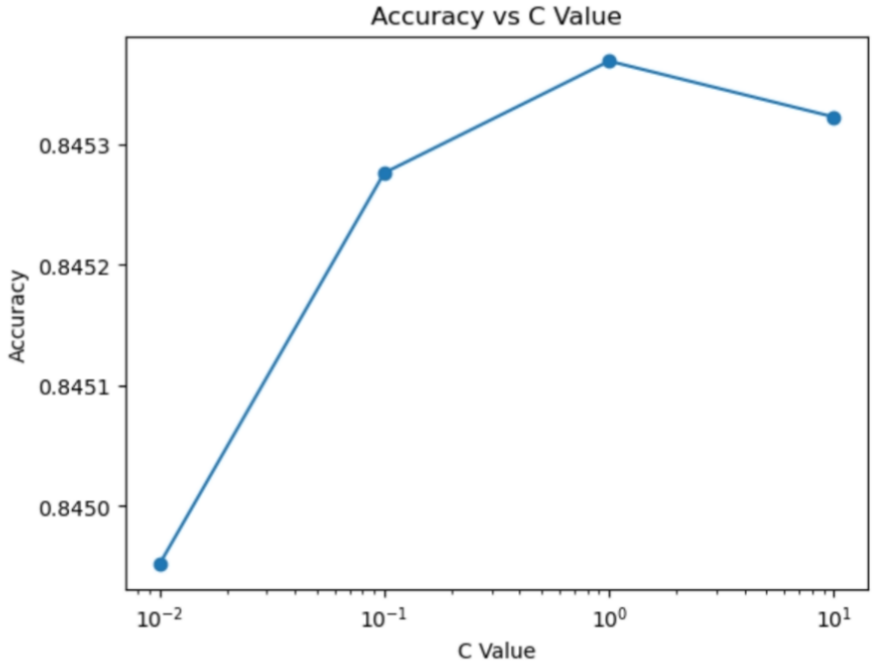

Figure 1: Linear SVM cost value vs accuracy

Permutation feature importance analysis revealed the most significant contributors to the linear Support Vector Machine (SVM) model's performance. To validate the relevance of these key features, a reduced-feature linear SVM was trained using only **the top five**, achieving an **accuracy of 84.04%.** This slight decrease from the original model suggests that these features are crucial to model performance. A further reduced model, using just **the top two features**, achieved an **accuracy of 81.19%**, indicating these core features are important but may require additional context or features to maintain optimal accuracy.

2. Radial SVM:

For the Support Vector Machine (SVM) model with a Radial Basis Function (RBF) kernel were derived using GridSearchCV with a parameter grid of 'C': [0.01, 0.1, 1, 10] and 'gamma': [0.01, 0.1, 1, 10]. The best parameter combination for the RBF kernel was **C = 1 and gamma = 0.1,** achieving an **accuracy of 85.59%** on the test set. This suggests that the RBF kernel's flexibility allows it to capture more complex patterns in the data, providing a slightly higher accuracy compared to linear SVM.

The accuracy of the RBF SVM model with the **top 5 most** important features **was 84.04%,** indicating that a reduced-feature approach maintained strong predictive performance. However, the accuracy dropped to **81.05%** when using only the **top 2 features**, suggesting that while these features are significant, additional features are needed for optimal results with an RBF kernel.

3. Polynomial Kernel:

Below Fig. 2: for the Support Vector Machine (SVM) model with a polynomial kernel were derived using **GridSearchCV** to find the best parameter configuration. The optimal polynomial SVM model achieved an **accuracy of 84.98%** on the test set.

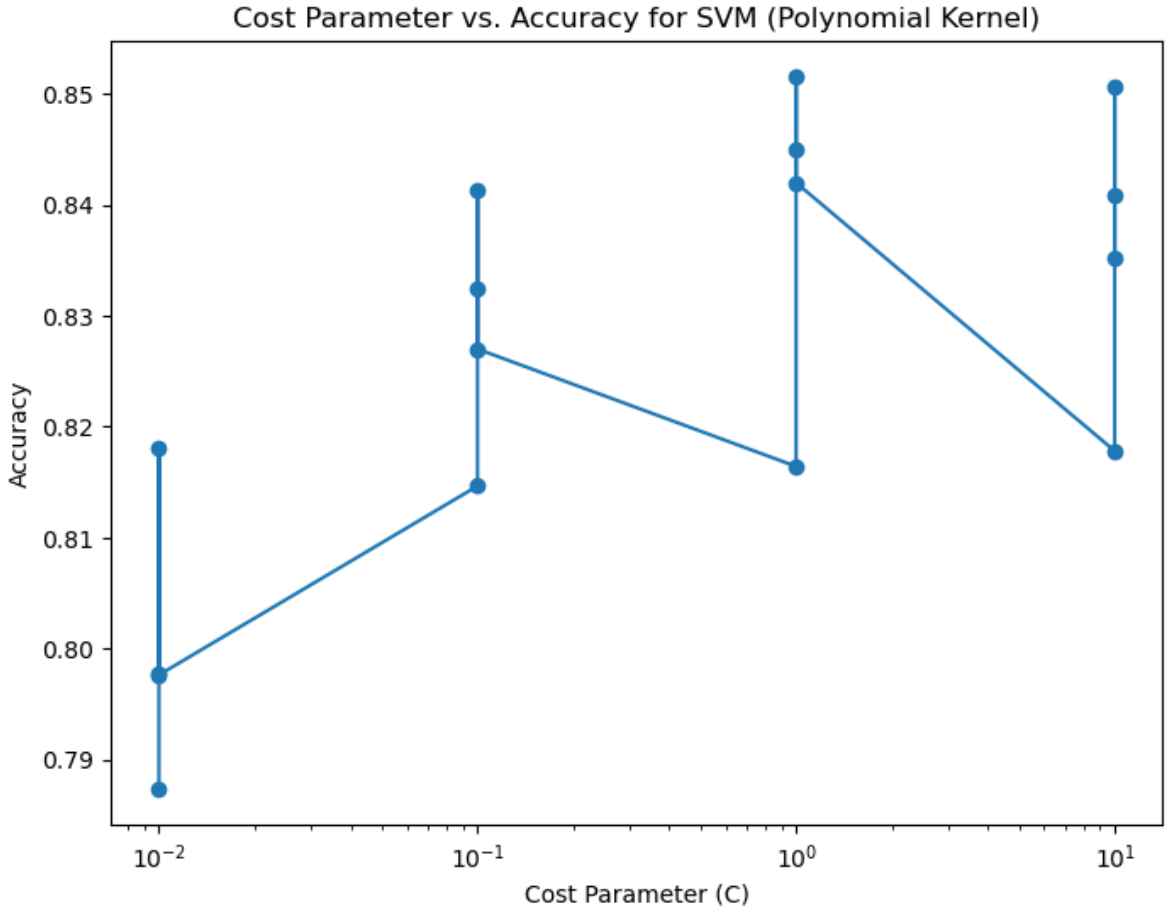

Fig 2: Polynomial kernel cost vs accuracy

To assess feature importance, a permutation analysis was conducted. The accuracy with **the top 5** most important features for the polynomial SVM was **83.32%,** showing that a reduced set of features can still maintain strong performance. However, when using only **the top** 2 most influential features, the **accuracy dropped to 75.87%.**

Discussion:

The 'VALUEH' (value of home) feature resulted in a model accuracy of 100%, indicating it might be uniquely distinctive in this dataset. This raises concerns about overfitting, suggesting the model might be memorizing rather than generalizing. To avoid this, we dropped 'VALUEH' from our analysis to ensure a more robust and generalizable model.

1. Linear SVM:

For linear SVM Permutation feature importance helps identify which features contribute most to the SVM model's predictive power. The top 5 features, ranked by their importance, were:

- BEDROOMS
- AGE
- COSTWATR
- ROOMS
- DENSITY
- HHINCOME
- COSTGAS

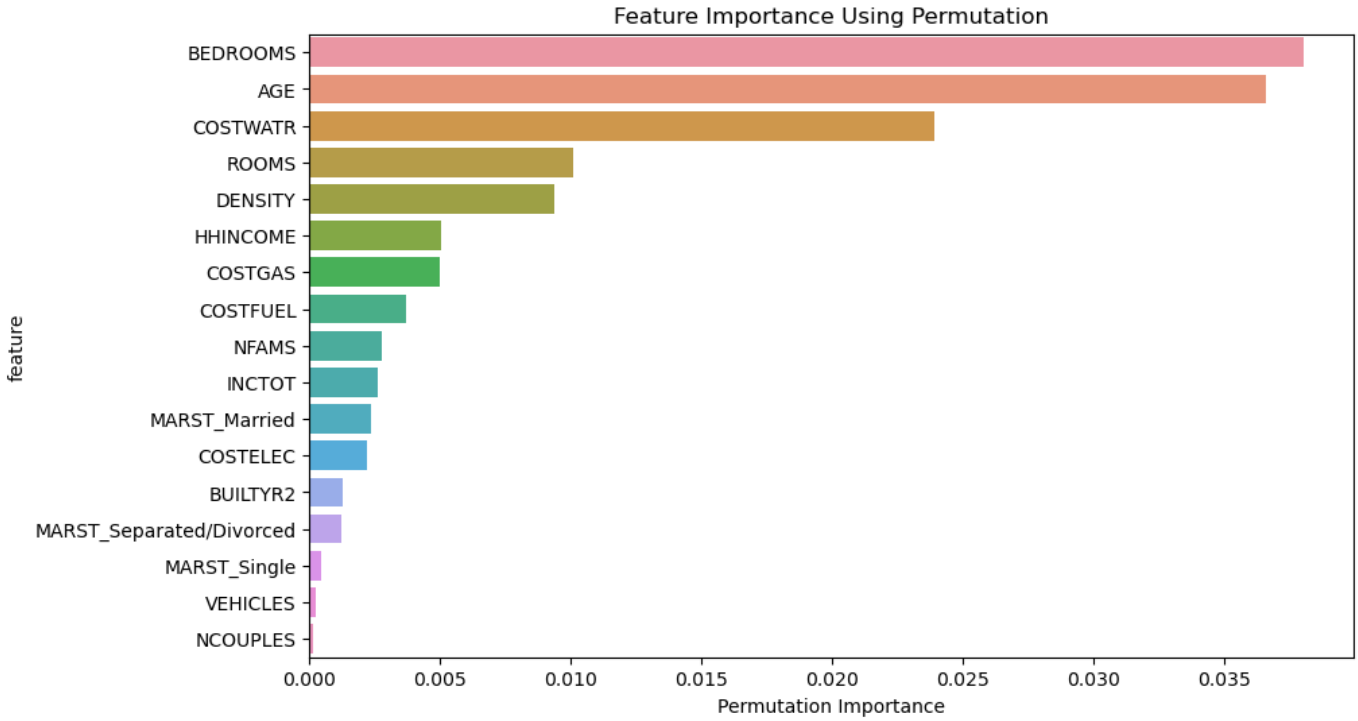

Fig 3: Most Important features for linear SVM

The importance plot (Fig. 3) shows how these features vary in their contribution to the model's performance. BEDROOMS and AGE emerged as the most significant features, suggesting that these factors play a critical role in classifying the dataset accurately.

A confusion matrix provides a deeper insight into the model's classification accuracy by illustrating true positives, false positives, true negatives, and false negatives. For the linear SVM with the top 5 features, the confusion matrix is as follows:

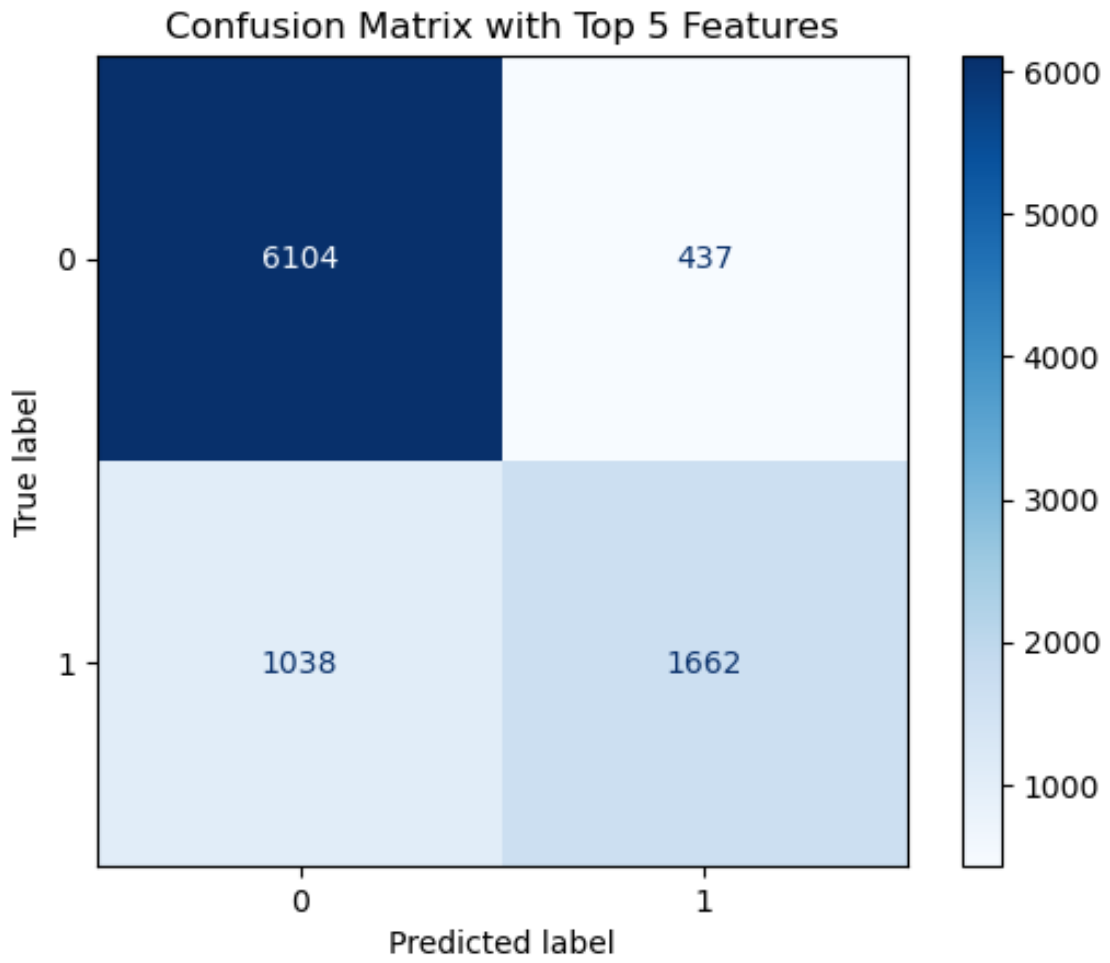

Fig 4: Top 5 feature Linear SVM model confusion matrix

The Fig 4 confusion matrix for top 5 features indicates that while the model performs well in correctly classifying instances (true positives and true negatives), there are a considerable number of false negatives (1038) and false positives (437), suggesting room for improvement in distinguishing between classes.

To evaluate the role of reduced features, a linear SVM was trained with only the top 5 important features, achieving an accuracy of 84.04%, suggesting that these features are robust predictors. However, using just the top 2 features, BEDROOMS and AGE, resulted in an accuracy of 81.19%, indicating that these two features alone might not be enough to maintain high predictive performance.

For the model trained with only the top 2 features, the confusion matrix is as follows:

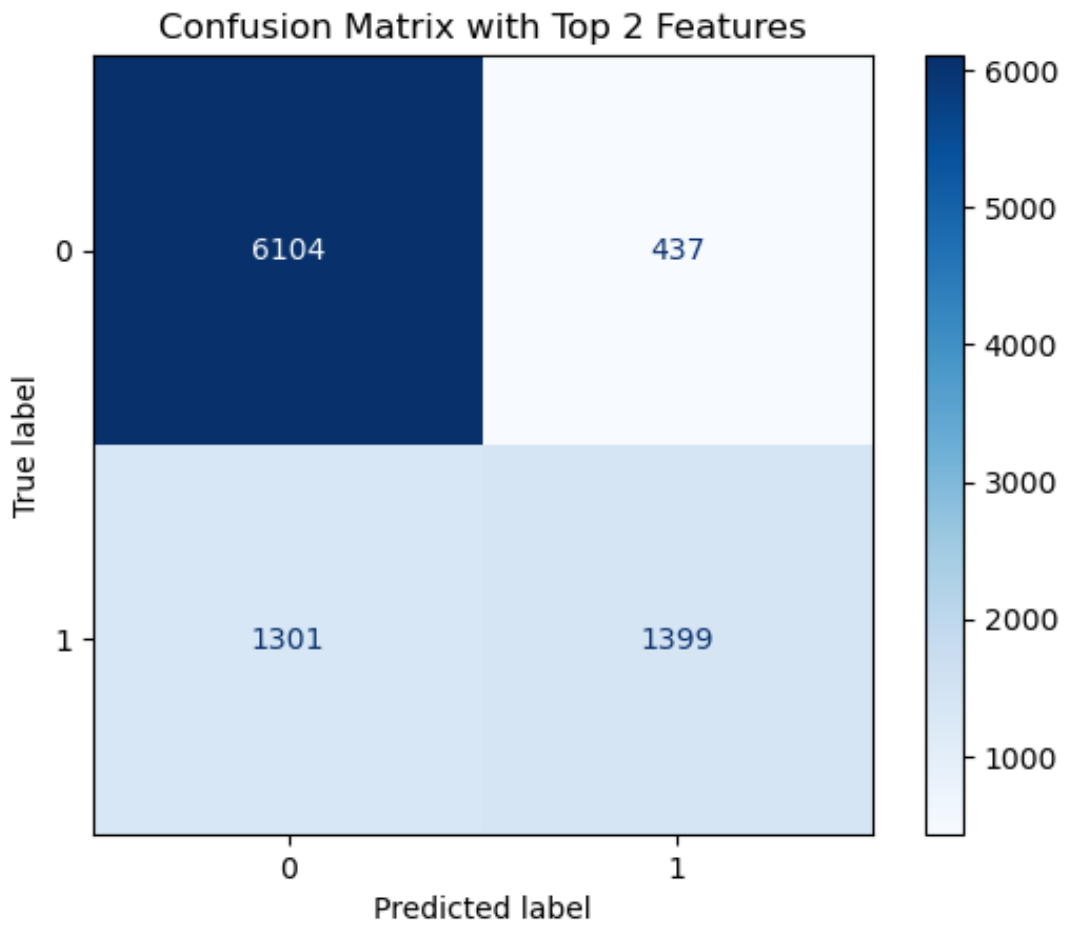

Fig 5: Confusion matrix for top 2 features linear SVM

From fig 5 The confusion matrix with only two features shows a higher number of false negatives (1301), indicating that reducing the feature set can impact the model's ability to classify correctly.

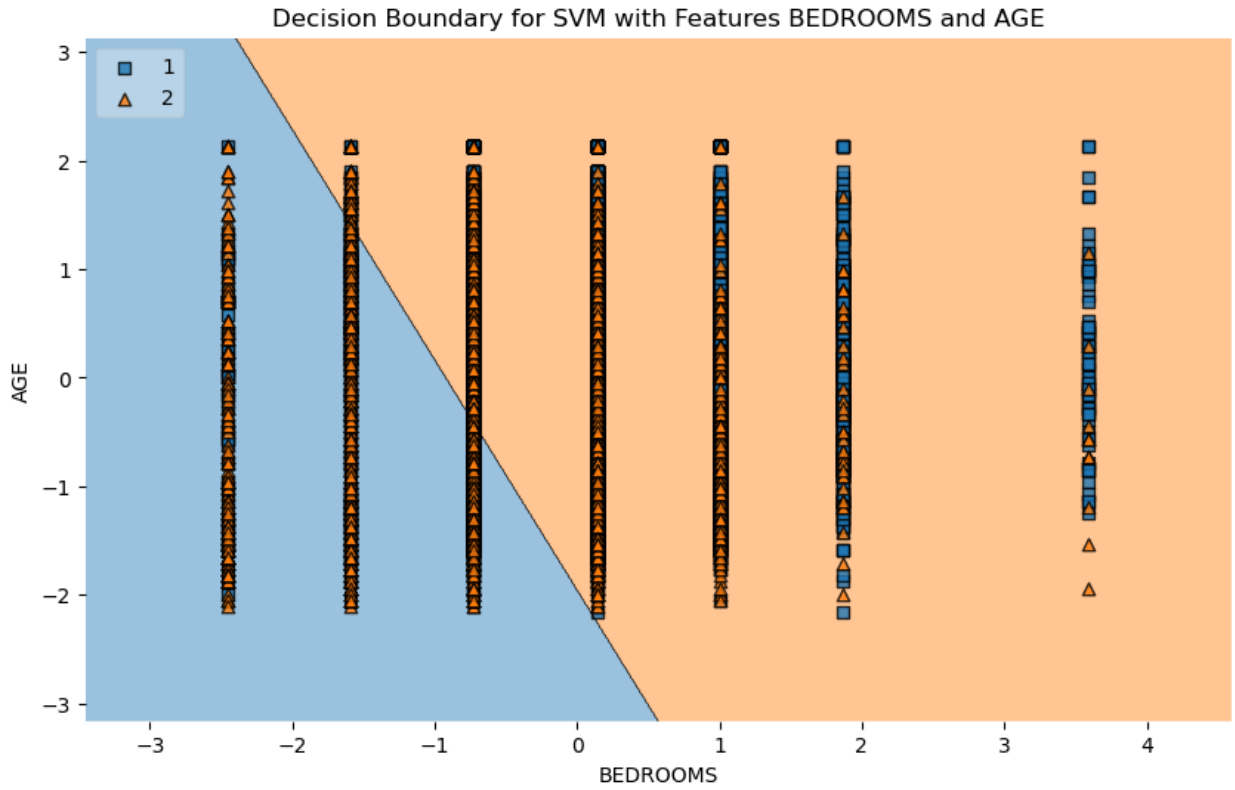

Fig 6: Decision Boundary plot for top 2 features

From above fig 6 decision boundary we can see that the blue region is for owned and the orange region for rented, The decision boundary separating the two areas shows that units with fewer bedrooms and lower ages tend to be renter-occupied (orange area), while units with more bedrooms and higher ages tend to be owner-occupied (blue area). However, there are misclassified points that highlight the boundary's limitations. In the bottom-right section, some orange triangles (renter-occupied) appear in an area typically associated with owner-occupied properties, indicating more bedrooms and older units. Meanwhile, in the top-left section, a few blue squares (owner-occupied) are found in a zone where you'd expect renter-occupied units with fewer bedrooms and newer age.

These misclassifications suggest that the model's ability to predict occupancy status isn't perfect, possibly due to other influencing factors beyond the two used in this model, or data errors. Analyzing these outliers can provide insights into how to improve the model's accuracy or indicate other factors that might be affecting housing occupancy status.

2. Radial SVM:

For radial svm Permutation feature importance analysis identifies the most significant features in the RBF SVM model. The top 5 features for RBF were:

- COSTWATR

- BEDROOMS
- AGE
- ROOMS
- DENSITY

The accuracy achieved with these top 5 features was 84.04%, indicating that a reduced set of key features provides strong predictive performance. The drop from the full-featured model's accuracy (85.59%) suggests that while these features are important, including more features could enhance the model's performance.

The importance plot (Fig. 7) shows how these features vary in their contribution to the model's performance. BEDROOMS and COSTWATER emerged as the most significant features.

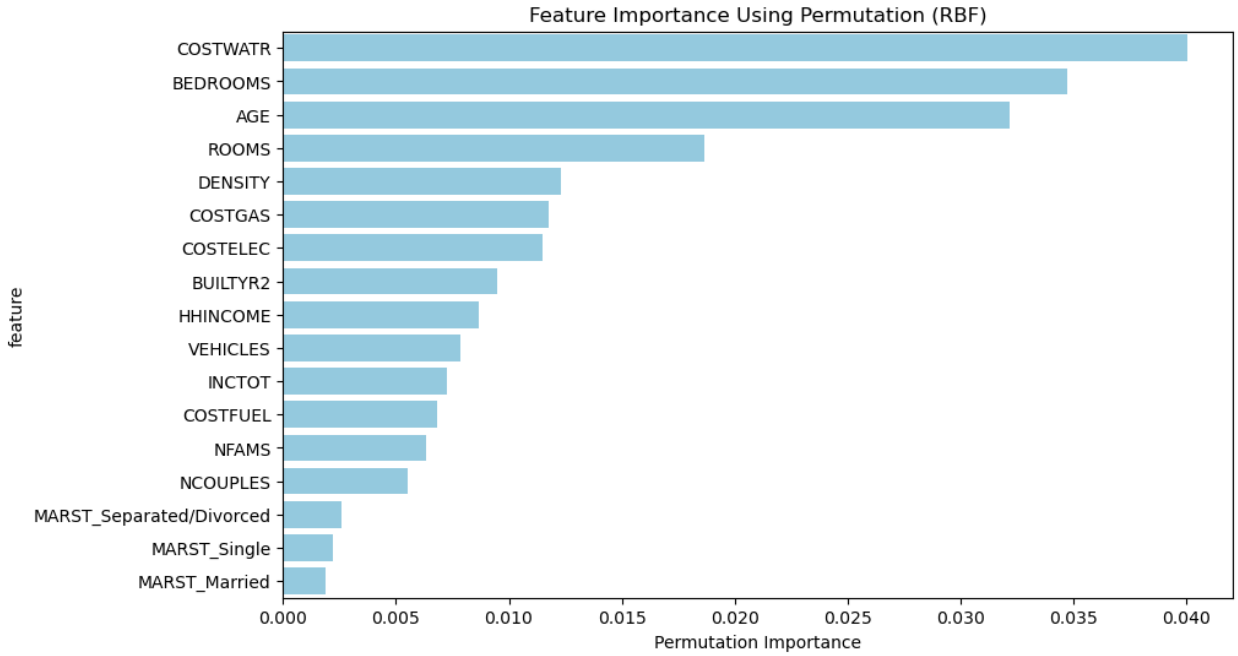

Fig 7: important features for radial svm

The below fig 8 confusion matrix provides insights into the RBF SVM's accuracy in terms of correct and incorrect classifications. Here's the confusion matrix for the RBF SVM with the top 5 features:

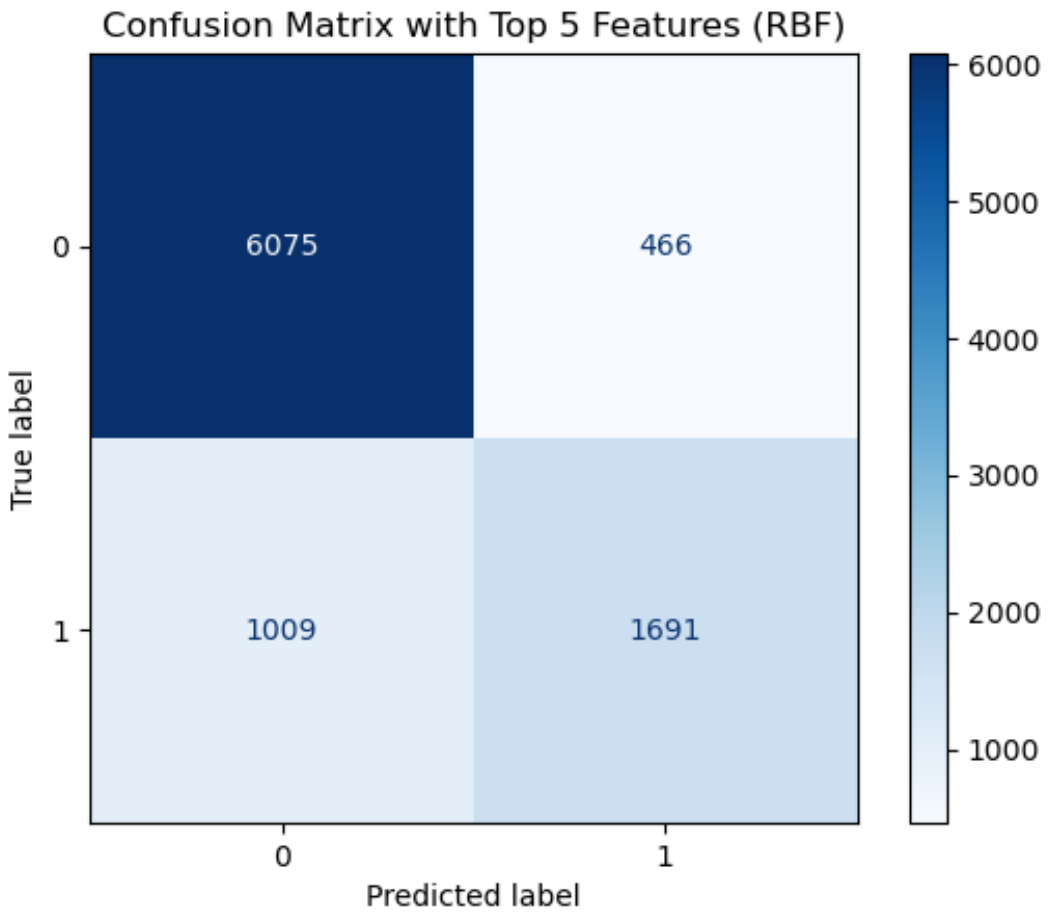

Fig 8: confusion matrix for radial SVM with top 5 features

The confusion matrix reveals a relatively high number of false positives (466) and false negatives (1009), suggesting the model's challenges in achieving perfect classification. However, the strong true positive (1619) and true negative (6075) counts indicate that the model performs well overall. The RBF SVM confusion matrix with the top 5 features showed more true positives (1619 vs. 1162) and slightly fewer false positives (466 vs. 437) compared to the linear SVM. This suggests that the RBF kernel might have a better capacity to capture complex patterns and minimize misclassifications.

For the model trained with only the top 2 features ('COSTWATR' and 'BEDROOMS'), the accuracy dropped to 81.05%. Here's the confusion matrix:

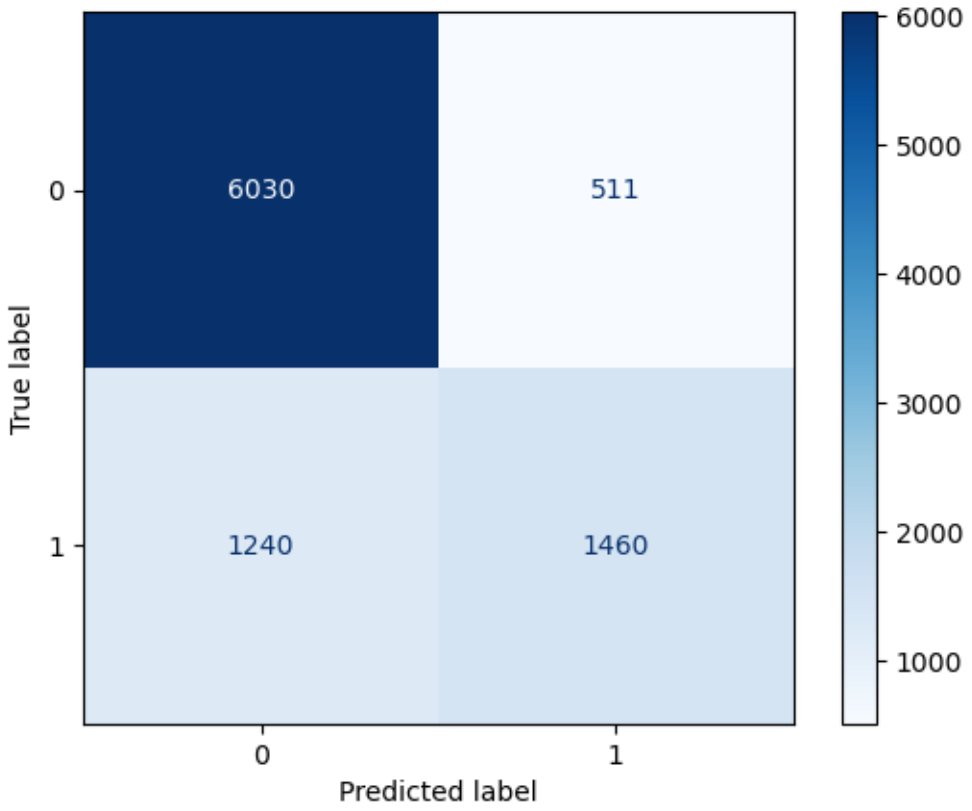

Fig 9: Confusion matrix for top 2 feature radial svm

From the fig 9 The increased number of false negatives (1240) compared to the top 5-feature model (1009) indicates the importance of using a broader set of key features to maintain optimal accuracy.

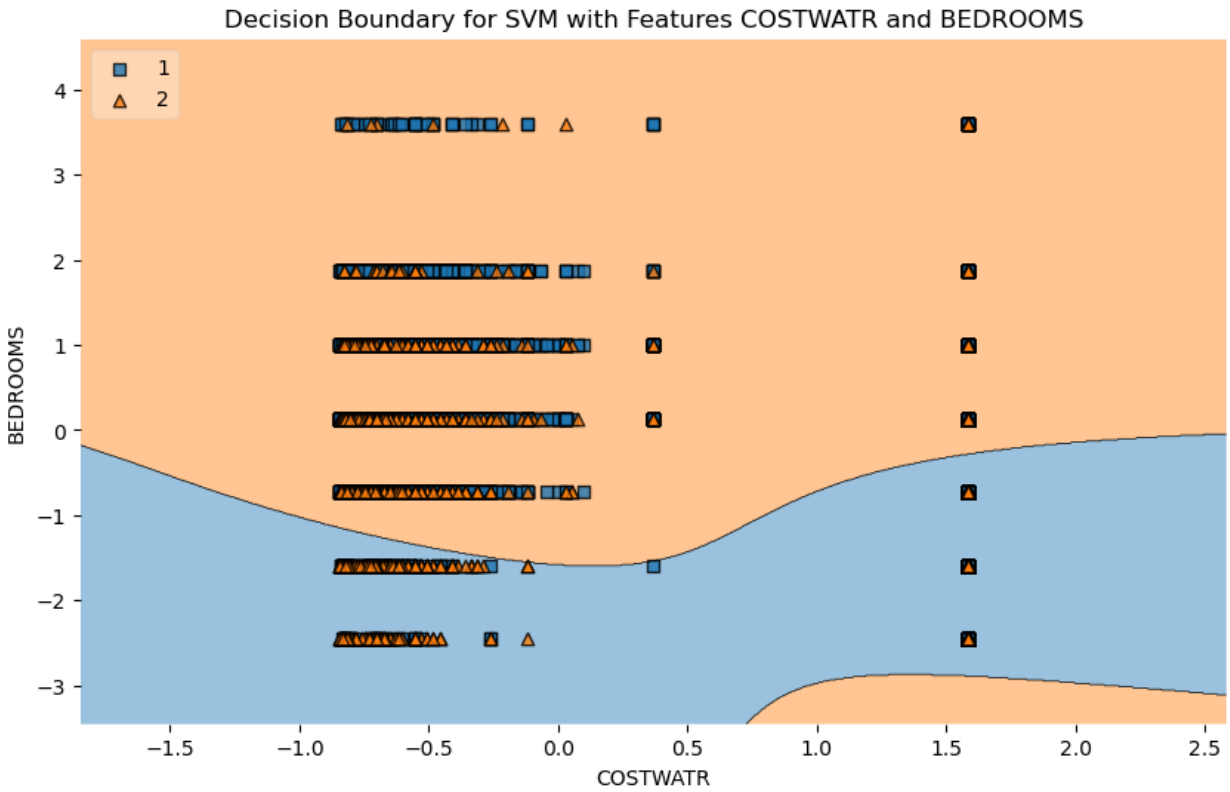

Fig 10: Decision Boundary for radial SVM

The above fig 10 plot shows the decision boundary for a support vector machine (SVM) model that classifies housing units into two categories (1 and 2, likely owner-occupied and renter-occupied) based on the features COSTWATR and BEDROOMS. The decision boundary, represented by the blue and orange shaded areas, separates the two classes based on these two features. The blue area (middle – lower) corresponds to class 1, while the orange area (top) corresponds to class 2. From the decision boundary, we can observe the following patterns:

- Lower COSTWATR and fewer BEDROOMS are associated with class 1 (likely owner-occupied units).
- Higher COSTWATR and more BEDROOMS are associated with class 2 (likely renter-occupied units).

Yet, some misclassifications occur, with points in the orange area showing traits typical of Class 1 (higher COSTWATR, more BEDROOMS), and points in the blue area displaying traits of Class 2 (lower COSTWATR, fewer BEDROOMS). These misclassifications might result from other unseen factors, data noise, or model errors.

3. Polynomial SVM:

Permutation feature importance, we identified the top 5 features for the polynomial SVM: COSTWATR, AGE, BEDROOMS, ROOMS, and COSTGAS. These features showed significant impact on the model's performance, with the polynomial SVM achieving an accuracy of 83.32% when trained with these top 5 features. This result suggests that the model retains robust predictive capability with a reduced feature set.

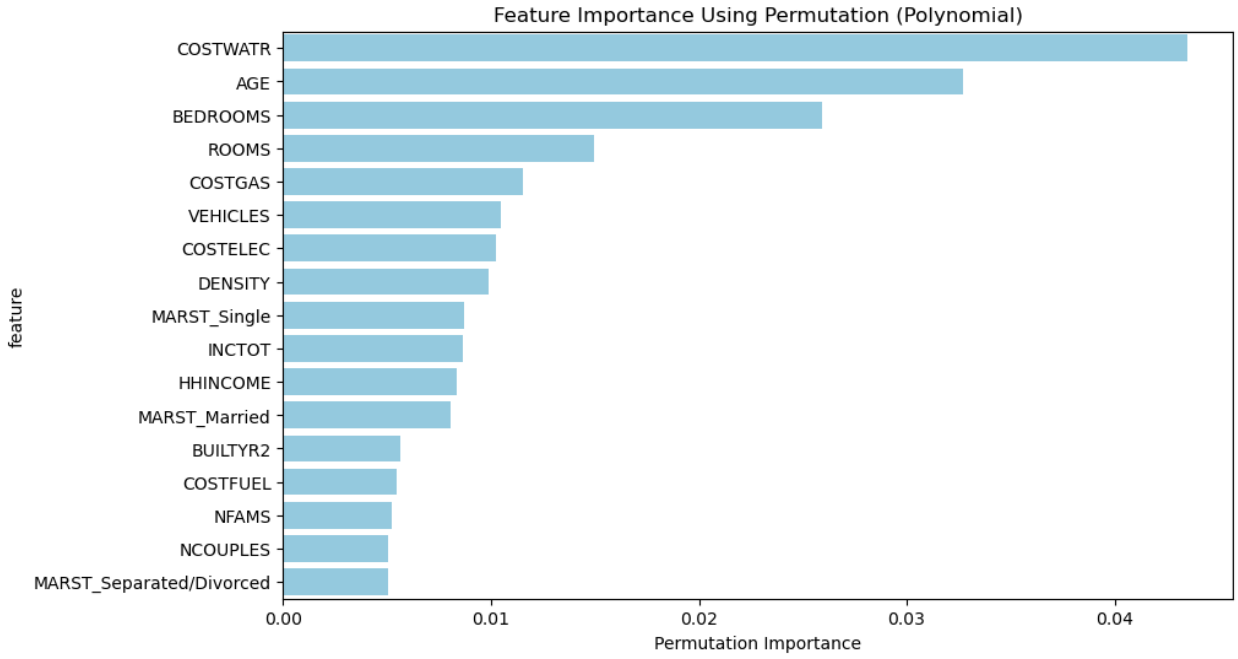

Fig 11 feature importance for polynomial SVM

The importance plot (Fig. 11) shows how these features vary in their contribution to the model's performance. COSTWATER and AGE emerged as the most significant features, suggesting that these factors play a critical role in classifying the dataset accurately.

The polynomial SVM's accuracy with the top 5 features (83.32%) is slightly lower than the linear (84.04%) and radial (84.04%) SVMs. This suggests that the polynomial SVM may require more tuning or a broader set of features to achieve optimal accuracy.

The confusion matrix offers insights into the polynomial SVM's ability to classify instances correctly and identify patterns in misclassification. Here's below fig 12 the confusion matrix for the polynomial SVM with the top 5 features:

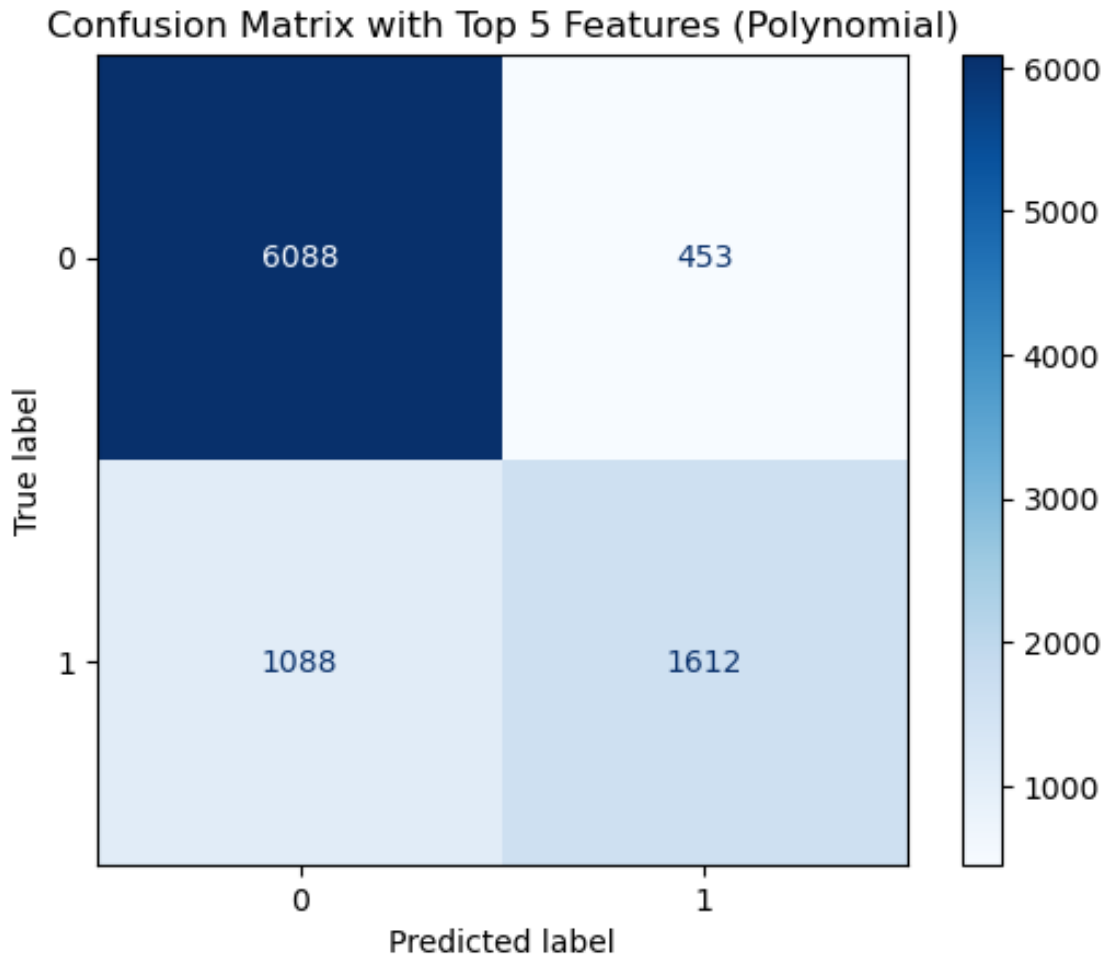

Fig 12: Confusion Matrix for top 5 features polynomial SVM

This matrix indicates a relatively high number of false positives (453) and false negatives (1088). However, the model still shows a strong count of true positives (1612) and true negatives (6088), indicating good overall performance.

For the model trained with the top 2 features, below is fig 13 the confusion matrix is as follows:

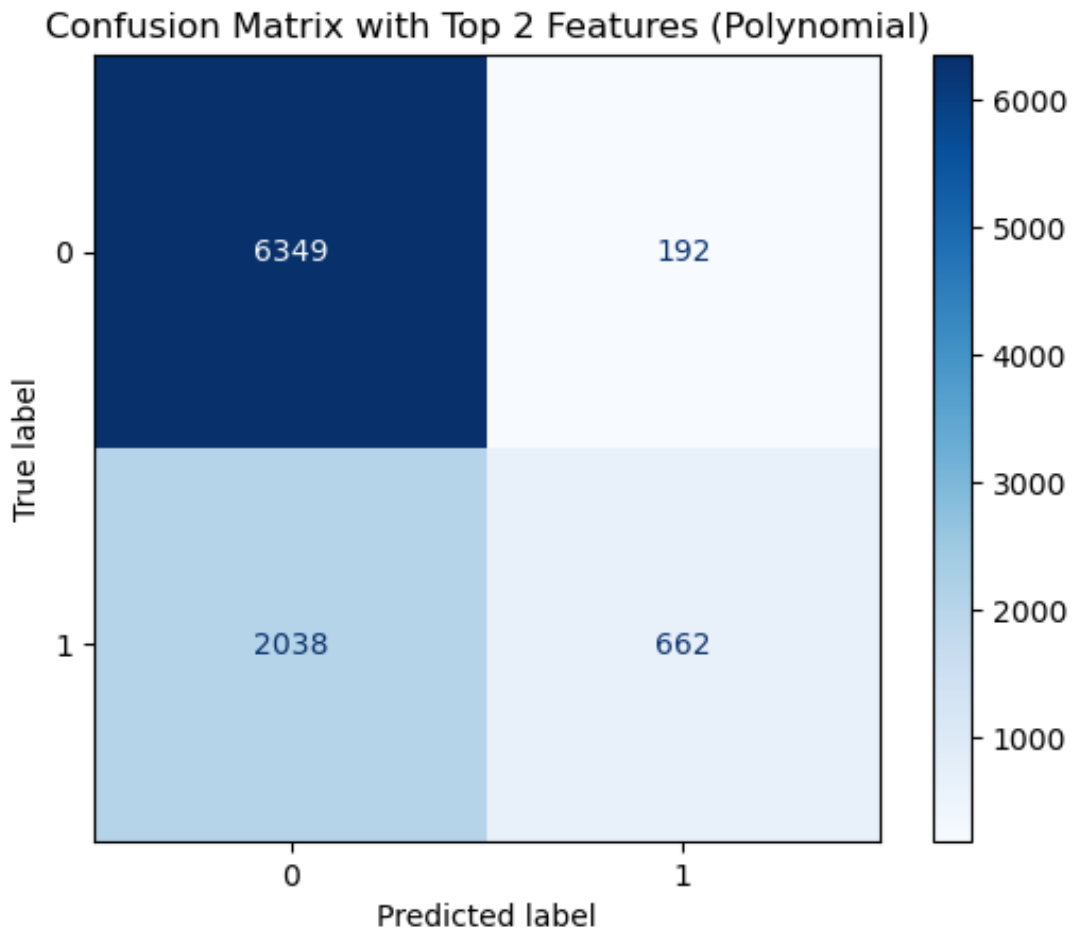

Fig 13: Confusion matrix for top 2 features polynomial SVM

This confusion matrix highlights the significant increase in false negatives (2038), suggesting that relying on just two features is not sufficient for the polynomial SVM, reinforcing the need for a broader feature set to maintain accuracy.

The confusion matrix for the polynomial SVM with the top 5 features shows similar patterns to those of the linear and radial SVMs, with a balanced distribution of true positives and true negatives. However, the model with only the top 2 features showed a significant increase in false negatives, indicating that the polynomial SVM may be more sensitive to reduced feature sets.

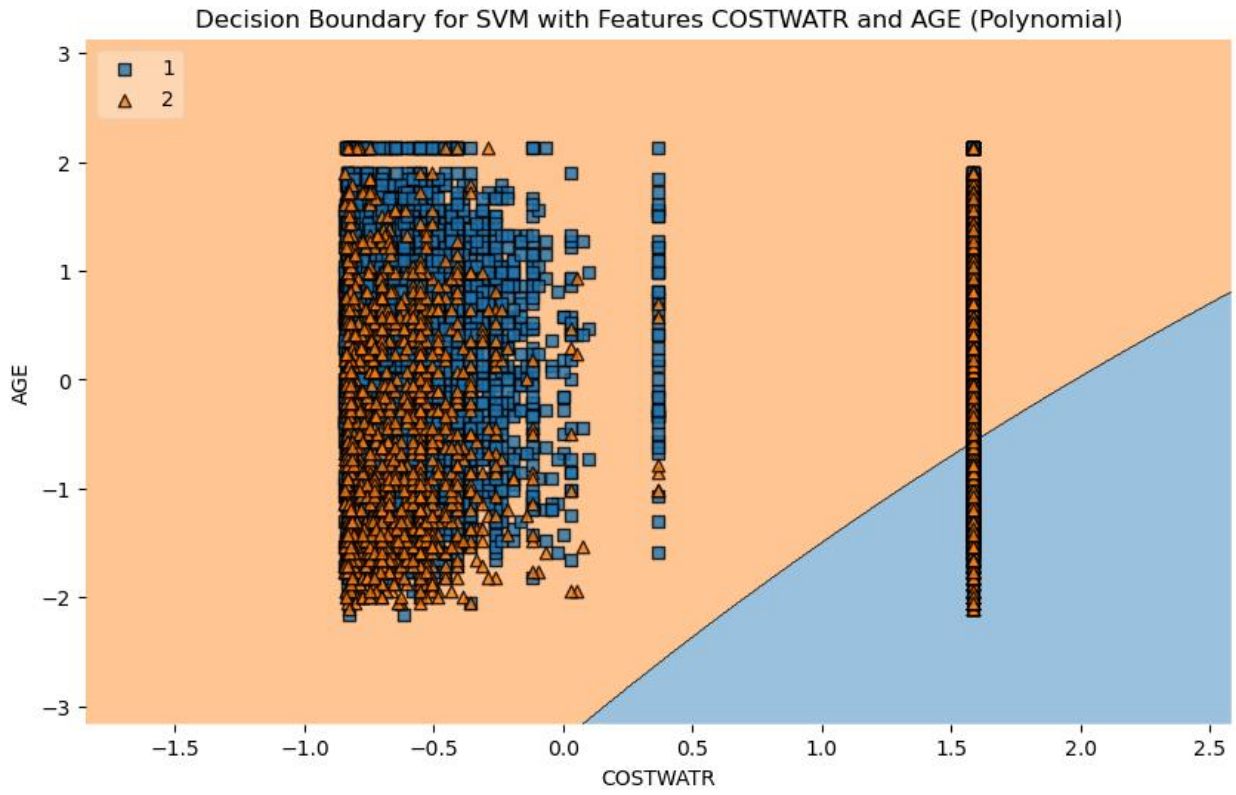

Fig 14: Decision boundary for polynomial SVM

The above fig 14 shows the decision boundary plot for polynomial SVM. The decision boundary is no longer a straight line as in the previous plots but instead forms a curved, non-linear shape separating the two classes. This is due to the use of a polynomial kernel, which allows the SVM to model more complex, non-linear relationships between the features and the class labels. The blue and orange data points represent the owners and renters One notable aspect of this plot is the high degree of overlap between the two classes in the central region of the feature space. This suggests that for certain combinations of COSTWATR and AGE values, the model may have difficulty distinguishing between owner-occupied and renter-occupied units based on these two features alone.

Conclusion:

Throughout this project, we examined the application of Support Vector Machine (SVM) models in predicting home ownership, focusing on key demographic factors. Our analysis used various SVM kernels, including linear, radial (RBF), and polynomial, to understand their effectiveness in predicting whether a home is owned or rented.

Permutation feature importance revealed that certain features consistently played a significant role in model accuracy across all SVM variants. The most critical features included 'BEDROOMS', 'AGE', 'COSTWATR', 'ROOMS', and 'DENSITY'. Among these, 'BEDROOMS' and 'AGE' emerged as the top two features, indicating that they hold considerable predictive power in distinguishing between owned and rented homes.

Considering accuracy and robustness, the best model was the Radial SVM. It achieved the highest accuracy of 85.59% with the optimal parameter combination, demonstrating its flexibility in capturing complex, non-linear relationships in the dataset. The RBF SVM outperformed both the linear and polynomial SVMs, making it the preferred choice for this prediction task.

References:

1. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-018-2451-4
2. Dataset : D010.V13.0 | IPUMS
3. Scikit learn: https://scikit-learn.org/stable/
4. https://scikit-learn.org/stable/modules/svm.html

Appendix:

Data Preprocessing:

```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.feature_selection import RFE
from sklearn.feature_selection import RFECV
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.inspection import permutation_importance
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler, LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions

!pip install mlxtend

data = pd.read_csv("./Housing.csv")
data
```

```
# Considering only elder person as household owner
df = data.sort_values(['SERIAL', 'AGE'], ascending=[True, False]).drop_duplicates('SERIAL')
df.shape

df.columns

df_final = df.drop(['SERIAL','OWNERSHPD','BIRTHYR', 'VALUEH','EDUC',
'EDUCD','BIRTHYR','PERWT','PERNUM'],axis = 1)
df_final

df_final = df_final[~df_final['ROOMS'].isin([00])]
df_final = df_final[~df_final['BUILTYR2'].isin([00])]
df_final = df_final[~df_final['BEDROOMS'].isin([00])]
df_final = df_final[~df_final['VEHICLES'].isin([0,])]
df_final = df_final[~df_final['AGE'].isin([999])]
df_final = df_final[~df_final['MARST'].isin([9])]

marital_mapping = {
   1: 'Married',  # Married, spouse present
   2: 'Married',  # Married, spouse absent
   3: 'Separated/Divorced',  # Separated
   4: 'Separated/Divorced',  # Divorced
   5: 'Single',  # Widowed
   6: 'Single'  # Never married/single
}
# Apply the mapping to create a new categorical column
df_final['MARST'] = df_final['MARST'].map(marital_mapping)
df_final = pd.get_dummies(df_final, columns=['MARST'])
df_final.columns
df_final

# Split features and target
X = df_final.drop("OWNERSHP", axis=1)  # All features except target
y = df_final["OWNERSHP"]  # Target variable["OWNERSHP"]  # Target variable

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

scaler = StandardScaler()

# Fit the scaler on the training set and transform both training and test sets
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)  # Transform on the test set
```

Linear SVM:

```
param_grid = {'C': [0.01, 0.1, 1, 10], 'kernel': ['linear']}

# Initialize GridSearchCV
grid_search = GridSearchCV(SVC(verbose=True,shrinking=False,), param_grid, scoring='accuracy',
```

```
cv=10, n_jobs=-1)

# Fit the GridSearchCV on the training data
grid_search.fit(X_train_scaled, y_train)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Print the best parameters
print("Best parameters:", best_params)

# Evaluate the best model on the testing data
accuracy = best_estimator.score(X_test_scaled, y_test)
print("Accuracy with the best model: %.2f%%" % (accuracy * 100))


results = grid_search.cv_results_

# Get the unique C values
C_values = param_grid['C']

# Extract the mean test scores for the different C values
mean_test_scores = results['mean_test_score']

# Find indices for the linear kernel and varying C values
C_indices = [i for i, param in enumerate(results['params']) if param['kernel'] == 'linear']

# Get the mean scores corresponding to these indices
C_mean_scores = [mean_test_scores[i] for i in C_indices]

# Plot C values versus accuracy
plt.plot(C_values, C_mean_scores, marker='o', linestyle='-', label='Accuracy')

# Label the axes and title
plt.xlabel('C Value')
plt.ylabel('Accuracy')
plt.title('Accuracy vs C Value')
plt.xscale('log')  # Optional, to better visualize a range of C values
plt.show()


best_svm = grid_search.best_estimator_

# Calculate permutation feature importance
permutation_results = permutation_importance(best_svm, X_train_scaled, y_train, n_repeats=10,
random_state=42)
importance_means = permutation_results.importances_mean  # Average importance over all repeats

# Create a DataFrame for feature importance
feature_importance_df = pd.DataFrame(
   {"feature": X.columns, "importance": importance_means}
```

```
).sort_values(by="importance", ascending=False)

feature_importance_df

# Plot the feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x="importance", y="feature", data=feature_importance_df)
plt.xlabel("Permutation Importance")
plt.title("Feature Importance Using Permutation")
plt.show()

top_5_features = feature_importance_df["feature"].head(7).tolist()

print("Top 5 features from permutation importance:", top_5_features)

# Filter the dataset to include only the top 5 features
X_top5 = X[top_5_features]

# Split the new dataset into training and testing sets
X_train_top5, X_test_top5, y_train_top5, y_test_top5 = train_test_split(
    X_top5, y, test_size=0.3, random_state=42)

# Scale the new dataset
scaler_top5 = StandardScaler()
X_train_top5_scaled = scaler_top5.fit_transform(X_train_top5)
X_test_top5_scaled = scaler_top5.transform(X_test_top5)

# Train an SVM model with the top 5 features
svm_top5 = SVC(kernel='linear', C=1, random_state=42)  # Adjust C and other parameters as needed
svm_top5.fit(X_train_top5_scaled, y_train_top5)

# Calculate accuracy
accuracy_top5 = accuracy_score(y_test_top5, svm_top5.predict(X_test_top5_scaled))
print(f"Accuracy with top 5 features: {accuracy_top5 * 100:.2f}%")

# Confusion Matrix for the model with top 5 features
conf_matrix_top5 = confusion_matrix(y_test_top5, svm_top5.predict(X_test_top5_scaled))
disp_top5 = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_top5)
disp_top5.plot(cmap='Blues')
plt.title("Confusion Matrix with Top 5 Features")
plt.show()

for var in top_5_features:
    plt.figure(figsize=(10, 6))  # Set figure size
    sns.countplot(x=var, hue='OWNERSHP', data=df_final)  # Use 'OWNERSHP' for the hue
    plt.title(f"Count Plot for {var} with Hue 'OWNERSHP'")  # Title indicating the variable
    plt.show()  # Display the plot
```

```
top_2_features = feature_importance_df["feature"].head(2).tolist()

print("Top 2 features from permutation importance:", top_2_features)

# Filter the dataset to include only the top 5 features
X_top2 = X[top_2_features]
# Split the new dataset into training and testing sets
X_train_top2, X_test_top2, y_train_top2, y_test_top2 = train_test_split(
   X_top2, y, test_size=0.3, random_state=42)

# Scale the new dataset
scaler_top2 = StandardScaler()
X_train_top2_scaled = scaler_top5.fit_transform(X_train_top2)
X_test_top2_scaled = scaler_top5.transform(X_test_top2)

# Train an SVM model with the top 5 features
svm_top2 = SVC(kernel='linear', C=1, random_state=42)  # Adjust C and other parameters as needed
svm_top2.fit(X_train_top2_scaled, y_train_top2)

# Calculate accuracy
accuracy_top2 = accuracy_score(y_test_top2, svm_top2.predict(X_test_top2_scaled))
print(f"Accuracy with top 2 features: {accuracy_top2 * 100:.2f}%")

# Confusion Matrix for the model with top 2 features
conf_matrix_top2 = confusion_matrix(y_test_top2, svm_top2.predict(X_test_top2_scaled))
disp_top5 = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_top2)
disp_top5.plot(cmap='Blues')
plt.title("Confusion Matrix with Top 2 Features")
plt.show()

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_top2_scaled, y_test_top2.to_numpy(), clf=svm_top2, legend=2)
plt.xlabel(top_2_features[0])
plt.ylabel(top_2_features[1])
plt.title(f"Decision Boundary for SVM with Features {top_2_features[0]} and {top_2_features[1]}")
plt.show()
```

Radial kernel:

```
# New parameter grid for RBF kernel with different gamma values
param_grid_rbf = {'C': [0.01, 0.1, 1, 10], 'gamma': [0.01, 0.1, 1, 10]}

# Initialize GridSearchCV with RBF kernel
grid_search_rbf = GridSearchCV(SVC(kernel='rbf', shrinking=False,verbose=True), param_grid_rbf,
scoring='accuracy', cv=10, n_jobs=-1)

# Fit the GridSearchCV on the training data
grid_search_rbf.fit(X_train_scaled, y_train)
```

```
# Get the best parameters and the best estimator
best_params_rbf = grid_search_rbf.best_params_
best_estimator_rbf = grid_search_rbf.best_estimator_

print("Best parameters for RBF kernel:", best_params_rbf)

# Evaluate the best RBF model on the test set
accuracy_rbf = best_estimator_rbf.score(X_test_scaled, y_test)
print(f"Accuracy with the best RBF model: {accuracy_rbf * 100:.2f}%")

# Calculate permutation feature importance for the best RBF model
permutation_results_rbf = permutation_importance(
    best_estimator_rbf, X_train_scaled, y_train, n_repeats=10, random_state=42
)
importance_means_rbf = permutation_results_rbf.importances_mean

# Create a DataFrame for feature importance
feature_importance_df_rbf = pd.DataFrame(
    {"feature": X.columns, "importance": importance_means_rbf}
).sort_values(by="importance", ascending=False)

# Plot the feature importance for the RBF model
plt.figure(figsize=(10, 6))
sns.barplot(x="importance", y="feature", data=feature_importance_df_rbf, color="skyblue")
plt.xlabel("Permutation Importance")
plt.title("Feature Importance Using Permutation (RBF)")
plt.show()

# Get the top 5 features from the RBF model's permutation importance
top_5_features_rbf = feature_importance_df_rbf["feature"].head(5).tolist()
print("Top 5 features from permutation importance (RBF):", top_5_features_rbf)

# Filter the dataset to include only the top 5 features
X_top5_rbf = X[top_5_features_rbf]

# Split and scale the data with top 5 features
X_train_top5_rbf, X_test_top5_rbf, y_train_top5_rbf, y_test_top5_rbf = train_test_split(
    X_top5_rbf, y, test_size=0.3, random_state=42
)

scaler_top5_rbf = StandardScaler()
X_train_top5_rbf_scaled = scaler_top5_rbf.fit_transform(X_train_top5_rbf)
X_test_top5_rbf_scaled = scaler_top5_rbf.transform(X_test_top5_rbf)

# Train an SVM with RBF kernel using the top 5 features
svm_top5_rbf = SVC(kernel='rbf', C=1, gamma='scale', random_state=42)  # Adjust parameters as
needed
svm_top5_rbf.fit(X_train_top5_rbf_scaled, y_train_top5_rbf)

# Calculate accuracy for the RBF model with top 5 features
```

```
accuracy_top5_rbf = accuracy_score(y_test_top5_rbf, svm_top5_rbf.predict(X_test_top5_rbf_scaled))
print(f"Accuracy with top 5 features (RBF): {accuracy_top5_rbf * 100:.2f}%")

# Plot confusion matrix for the RBF model with top 5 features
conf_matrix_top5_rbf = confusion_matrix(y_test_top5_rbf,
svm_top5_rbf.predict(X_test_top5_rbf_scaled))
disp_top5_rbf = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_top5_rbf)
disp_top5_rbf.plot(cmap='Blues')
plt.title("Confusion Matrix with Top 5 Features (RBF)")
plt.show()

# Assuming 'df_final' contains your dataset with 'OWNERSHP'
# Plot count plots for the top 5 features with hue 'OWNERSHP'
for var in top_5_features_rbf:
   plt.figure(figsize=(10, 6))
   sns.countplot(x=var, hue='OWNERSHP', data=df_final)  # Ensure 'df_final' has the expected structure
   plt.title(f"Count Plot for {var} with Hue 'OWNERSHP'")
   plt.show()

# Filter the dataset to include only the top 5 features
top_2_features_rbf = feature_importance_df_rbf["feature"].head(2).tolist()

print("Top 2 features from permutation importance:", top_2_features_rbf)

X_top2_rbf = X[top_2_features_rbf]

# Split and scale the data with top 5 features
X_train_top2_rbf, X_test_top2_rbf, y_train_top2_rbf, y_test_top2_rbf = train_test_split(
   X_top2_rbf, y, test_size=0.3, random_state=42)

scaler_top2_rbf = StandardScaler()
X_train_top2_rbf_scaled = scaler_top2_rbf.fit_transform(X_train_top2_rbf)
X_test_top2_rbf_scaled = scaler_top2_rbf.transform(X_test_top2_rbf)

# Train an SVM with RBF kernel using the top 2 features
svm_top2_rbf = SVC(kernel='rbf', C=1, gamma='scale', random_state=42)  # Adjust parameters as
needed
svm_top2_rbf.fit(X_train_top2_rbf_scaled, y_train_top2_rbf)

# Calculate accuracy for the RBF model with top 2 features
accuracy_top2_rbf = accuracy_score(y_test_top2_rbf, svm_top2_rbf.predict(X_test_top2_rbf_scaled))
print(f"Accuracy with top 5 features (RBF): {accuracy_top2_rbf * 100:.2f}%")

# Plot confusion matrix for the RBF model with top 2 features
conf_matrix_top2_rbf = confusion_matrix(y_test_top2_rbf,
svm_top2_rbf.predict(X_test_top2_rbf_scaled))
disp_top2_rbf = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_top2_rbf)
disp_top2_rbf.plot(cmap='Blues')
plt.title("Confusion Matrix with Top 5 Features (RBF)")
plt.show()
```

```
# Assuming 'df_final' contains your dataset with 'OWNERSHP'
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_top2_rbf_scaled, y_test_top2_rbf.to_numpy(), clf=svm_top2, legend=2)
plt.xlabel(top_2_features_rbf[0])
plt.ylabel(top_2_features_rbf[1])
plt.title(f"Decision Boundary for SVM with Features {top_2_features_rbf[0]} and
{top_2_features_rbf[1]}")
plt.show()
```

Polynomial Kernel

```
param_grid_poly = {
   'C': [0.01, 0.1, 1, 10],
   'degree': [2, 3, 4, 5]  # Polynomial degrees to test
}

# Initialize GridSearchCV with polynomial kernel
grid_search_poly = GridSearchCV(SVC(kernel='poly',shrinking=False, verbose=True),
param_grid_poly, scoring='accuracy', cv=10, n_jobs=-1)

# Fit the GridSearchCV on the training data
grid_search_poly.fit(X_train_scaled, y_train)

# Get the best parameters and best estimator
best_params_poly = grid_search_poly.best_params_
best_estimator_poly = grid_search_poly.best_estimator_

print("Best parameters for polynomial kernel:", best_params_poly)

# Evaluate the best polynomial model on the test set
accuracy_poly = best_estimator_poly.score(X_test_scaled, y_test)
print(f"Accuracy with the best polynomial model: {accuracy_poly * 100:.2f}%")

# Calculate permutation feature importance for the best polynomial model
permutation_results_poly = permutation_importance(
   best_estimator_poly, X_train_scaled, y_train, n_repeats=10, random_state=42
)
importance_means_poly = permutation_results_poly.importances_mean

# Create a DataFrame for feature importance
feature_importance_df_poly = pd.DataFrame(
   {"feature": X.columns, "importance": importance_means_poly}
).sort_values(by="importance", ascending=False)

# Plot the feature importance for the polynomial model
plt.figure(figsize=(10, 6))
sns.barplot(x="importance", y="feature", data=feature_importance_df_poly, color="skyblue")
plt.xlabel("Permutation Importance")
plt.title("Feature Importance Using Permutation (Polynomial)")
plt.show()
```

```
# Get the top 2 features from the polynomial model's permutation importance

# Get the unique C values
C_values = [param['C'] for param in grid_search_poly.cv_results_['params']]

# Extract the mean test scores for each C value
mean_test_scores = grid_search_poly.cv_results_['mean_test_score']

# Plot C values versus accuracy
plt.figure(figsize=(8, 6))
plt.plot(C_values, mean_test_scores, marker='o', linestyle='-', label='Accuracy')
plt.xlabel('Cost Parameter (C)')
plt.ylabel('Accuracy')
plt.title('Cost Parameter vs. Accuracy for SVM (Polynomial Kernel)')
plt.xscale('log')  # Optional, to better visualize a range of C values
plt.show()

# Get the top 2 features from the polynomial model's permutation importance
top_5_features_poly = feature_importance_df_poly["feature"].head(5).tolist()

print("Top 5 features from permutation importance (Polynomial):", top_5_features_poly)

# Filter the dataset to include only the top 2 features
X_top5_poly = X[top_5_features_poly]

# Split and scale the data with top 2 features
X_train_top5_poly, X_test_top5_poly, y_train_top5_poly, y_test_top5_poly = train_test_split(
    X_top5_poly, y, test_size=0.3, random_state=42
)

scaler_top5_poly = StandardScaler()
X_train_top5_poly_scaled = scaler_top5_poly.fit_transform(X_train_top5_poly)
X_test_top5_poly_scaled = scaler_top5_poly.transform(X_test_top5_poly)

# Train an SVM with a polynomial kernel using the top 2 features
svm_top5_poly = SVC(kernel='poly', C=1, gamma='scale', degree=best_params_poly['degree'],
random_state=42)  # Adjust parameters as needed
svm_top5_poly.fit(X_train_top5_poly_scaled, y_train_top5_poly)

# Calculate accuracy for the polynomial model with top 2 features
accuracy_top5_poly = accuracy_score(y_test_top5_poly,
svm_top5_poly.predict(X_test_top5_poly_scaled))
print(f"Accuracy with top 5 features (Polynomial): {accuracy_top5_poly * 100:.2f}%")

# Plot confusion matrix for the polynomial model with top 2 features
conf_matrix_top5_poly = confusion_matrix(y_test_top5_poly,
svm_top5_poly.predict(X_test_top5_poly_scaled))
disp_top5_poly = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_top5_poly)
disp_top5_poly.plot(cmap='Blues')
```

```
plt.title("Confusion Matrix with Top 5 Features (Polynomial)")
plt.show()

# Plot decision boundary for the polynomial model with top 2 features
for var in top_5_features_poly:
    plt.figure(figsize=(10, 6))
    sns.countplot(x=var, hue='OWNERSHP', data=df_final)  # Ensure 'df_final' has the expected structure
    plt.title(f"Count Plot for {var} with Hue 'OWNERSHP'")
    plt.show()


top_2_features_poly = feature_importance_df_poly["feature"].head(2).tolist()

print("Top 2 features from permutation importance (Polynomial):", top_2_features_poly)

# Filter the dataset to include only the top 2 features
X_top2_poly = X[top_2_features_poly]

# Split and scale the data with top 2 features
X_train_top2_poly, X_test_top2_poly, y_train_top2_poly, y_test_top2_poly = train_test_split(
    X_top2_poly, y, test_size=0.3, random_state=42
)

scaler_top2_poly = StandardScaler()
X_train_top2_poly_scaled = scaler_top2_poly.fit_transform(X_train_top2_poly)
X_test_top2_poly_scaled = scaler_top2_poly.transform(X_test_top2_poly)

# Train an SVM with a polynomial kernel using the top 2 features
svm_top2_poly = SVC(kernel='poly', C=1, gamma='scale', degree=best_params_poly['degree'],
random_state=42)  # Adjust parameters as needed
svm_top2_poly.fit(X_train_top2_poly_scaled, y_train_top2_poly)

# Calculate accuracy for the polynomial model with top 2 features
accuracy_top2_poly = accuracy_score(y_test_top2_poly,
svm_top2_poly.predict(X_test_top2_poly_scaled))
print(f"Accuracy with top 2 features (Polynomial): {accuracy_top2_poly * 100:.2f}%")

# Plot confusion matrix for the polynomial model with top 2 features
conf_matrix_top2_poly = confusion_matrix(y_test_top2_poly,
svm_top2_poly.predict(X_test_top2_poly_scaled))
disp_top2_poly = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_top2_poly)
disp_top2_poly.plot(cmap='Blues')
plt.title("Confusion Matrix with Top 2 Features (Polynomial)")
plt.show()

# Plot decision boundary for the polynomial model with top 2 features
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_top2_poly_scaled, y_test_top2_poly.to_numpy(), clf=svm_top2_poly,
legend=2)
plt.xlabel(top_2_features_poly[0])
plt.ylabel(top_2_features_poly[1])
```

```
plt.title(f"Decision Boundary for SVM with Features {top_2_features_poly[0]} and
{top_2_features_poly[1]} (Polynomial)")
plt.show()
```