```
pip install ucimlrepo
```

CS613 Project -Francis Kurian

Support Vector Machines (SVM)

Support Vector Machines (SVM) are a class of supervised machine learning algorithms used for classification and regression tasks. SVM aims to find a decision boundary that maximizes the margin between different classes while minimizing classification errors. The "support vectors" are the data points closest to the decision boundary, and they play a crucial role in determining the boundary's position.

Project Summary:

The project demonstrates the use of SVM in a classification task using the heart disease dataset. Here is a summary of the project's main components:

Data Preparation: The heart disease dataset is fetched and prepared. Missing values are handled, and the data is split into features (x) and the target variable (y). Additionally, the data is divided into training and testing sets.

Base Kernel Comparison: The code compares SVM models with different default kernels ('linear,' 'poly,' 'rbf,' 'sigmoid') by calculating accuracy for each kernel. The accuracy is presented as a percentage on a bar chart, allowing for a visual comparison of model performance.

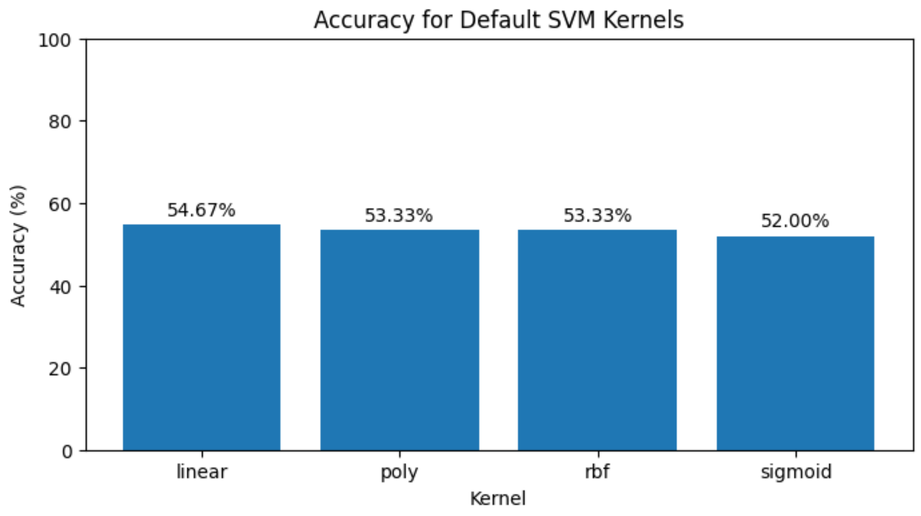
Kernel and Degree Tuning: The code then proceeds to explore SVM models with various kernel types and degrees. It calculates the mean error and accuracy for different combinations of kernels and degrees and presents the results using line plots. This analysis helps identify the impact of kernel choice and degree on model performance.

```
import warnings
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from ucimlrepo import fetch_ucirepo
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
warnings.simplefilter(action='ignore', category=FutureWarning)
# Fetch the heart disease dataset
heart_disease = fetch_ucirepo(id=45)

# Prepare the data
X = heart_disease.data.features
y = heart_disease.data.targets

combined_df = pd.concat([X, y], axis=1)
combined_df = combined_df.dropna()
combined_df.reset_index(drop=True, inplace=True)

# Split the data into features (x) and the target variable (y)
x = combined_df.iloc[:, 0:13].values
y = combined_df['num'].values

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)

# Standardize the features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
```

Baseline SVM Model by Kernel Types

```
# Create a list to store accuracy for different kernels
default_accuracy = []

# Create a DataFrame to store results for default settings
default_results_df = pd.DataFrame(columns=['Kernel', 'Accuracy'])
```

```python

# Evaluate each kernel with default settings
for kernel in kernels:
    default_svm = SVC(kernel=kernel)
    default_svm.fit(x_train, y_train)
    default_pred = default_svm.predict(x_test)

    # Calculate accuracy for default settings and convert to percentage
    default_acc = accuracy_score(y_test, default_pred) * 100

    # Append accuracy to the list and the DataFrame
    default_accuracy.append(default_acc)
    default_results_df = default_results_df.append({'Kernel': kernel, 'Accuracy': default_acc}, ignore_index=True)

# Visualize accuracy for default settings in percentage with labeled bars
plt.figure(figsize=(8, 4))
bars = plt.bar(default_results_df['Kernel'], default_results_df['Accuracy'])
plt.title('Accuracy for Default SVM Kernels')
plt.xlabel('Kernel')
plt.ylabel('Accuracy (%)')
plt.ylim(0, 100)

# Label the bars with their corresponding percentage values
for bar in bars:
    height = bar.get_height()
    plt.annotate(f'{height:.2f}%',  # Format the label as a percentage with two decimal places
                 xy=(bar.get_x() + bar.get_width() / 2, height),
                 xytext=(0, 3),  # Offset for the label position
                 textcoords='offset points',
                 ha='center', va='bottom')

plt.show()

```

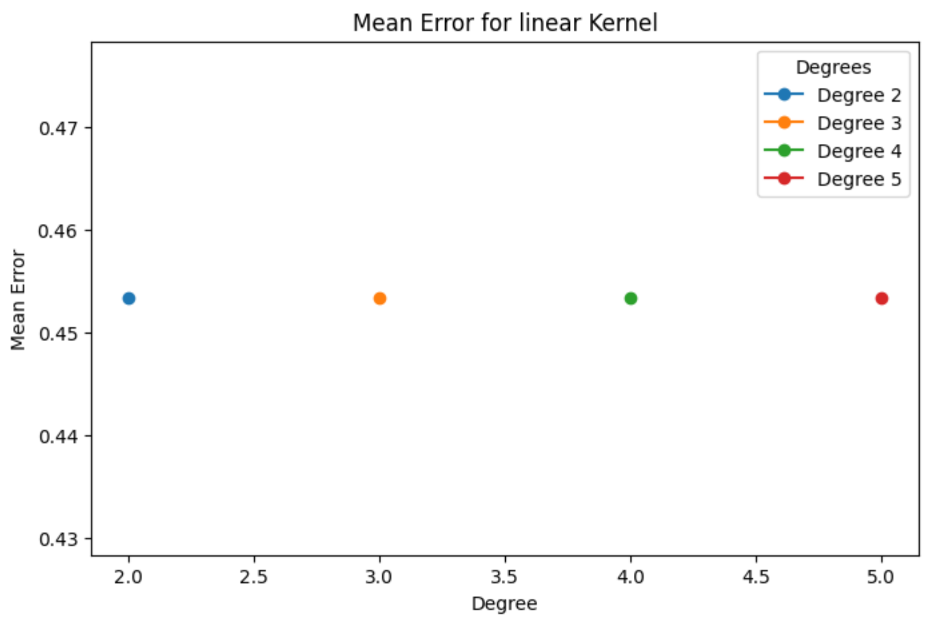


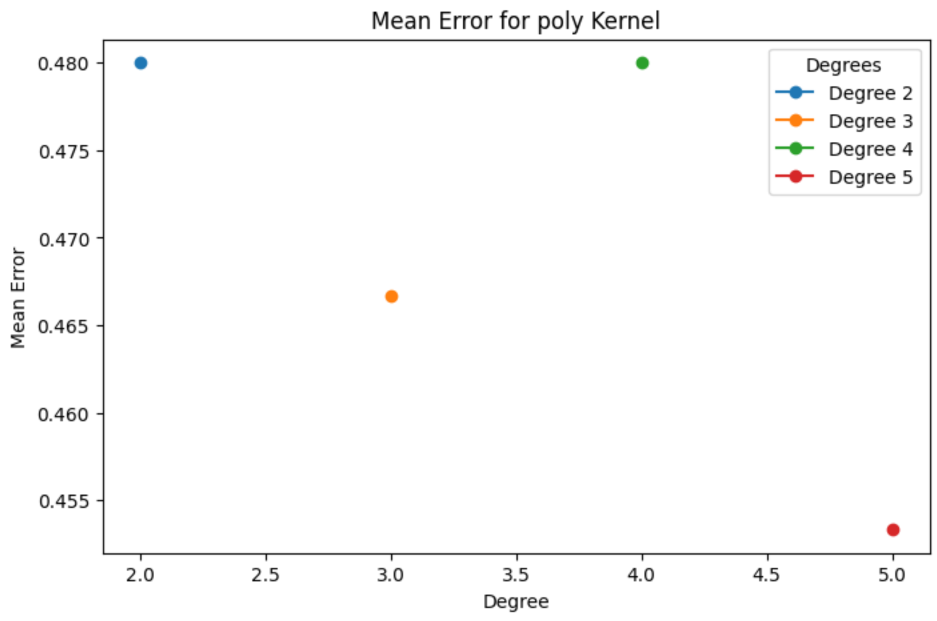
Mean Error Comparison by Kernel & Degrees

- Degrees Hyperparameter is only applicable to Kernel =polynomial.

```python
# Create empty lists to store error and accuracy
error = []
accuracy = []

# Create a DataFrame to store results
results_df = pd.DataFrame(columns=['Kernel', 'Degree', 'Mean Error', 'Accuracy'])

# Define the list of kernels and degrees to iterate over

degrees = [2, 3, 4, 5]

for kernel in kernels:
    for degree in degrees:
        svm = SVC(kernel=kernel, degree=degree)
        svm.fit(x_train, y_train)
        pred_i = svm.predict(x_test)

        # Calculate mean error and accuracy
        mean_error = np.mean(pred_i != y_test)
        acc = accuracy_score(y_test, pred_i)

        # Append values to the lists
        error.append(mean_error)
        accuracy.append(acc)

        # Append values to the DataFrame
        results_df = results_df.append({'Kernel': kernel, 'Degree': degree, 'Mean Error': mean_error, 'Accuracy': acc},
                                       ignore_index=True)

results_df.to_csv("svm_results.csv", index=False)

```

```
for kernel in kernels:
    plt.figure(figsize=(8, 5))
    for degree in degrees:
        subset = results_df[(results_df['Kernel'] == kernel) & (results_df['Degree'] == degree)]
        plt.plot(subset['Degree'], subset['Mean Error'], marker='o', label=f'Degree {degree}')

    plt.title(f'Mean Error for {kernel} Kernel')
    plt.xlabel('Degree')
    plt.ylabel('Mean Error')
    plt.legend(title='Degrees')
    plt.show()
```



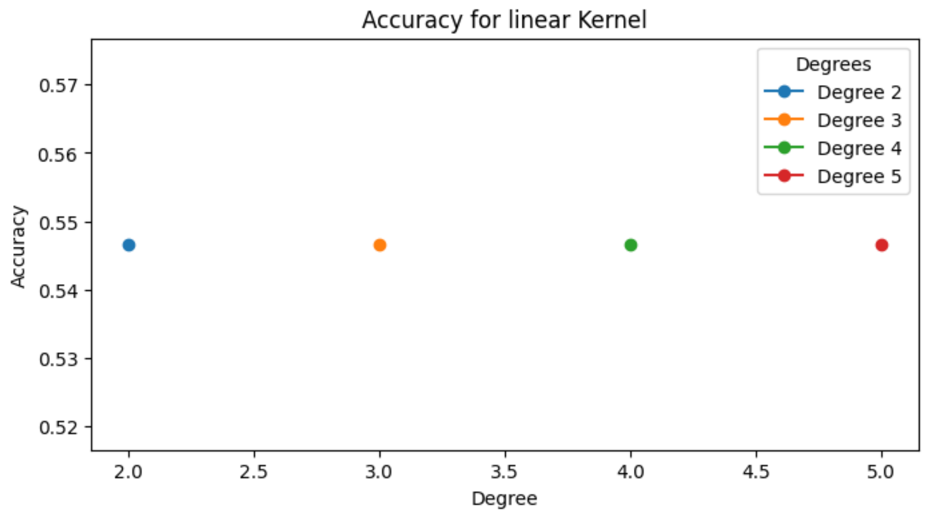


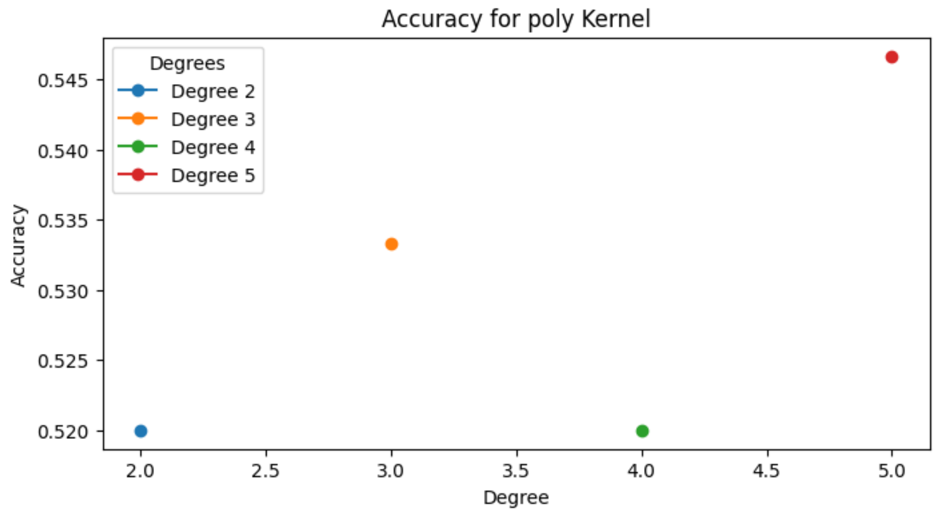
Mean Error for rbf Kernel

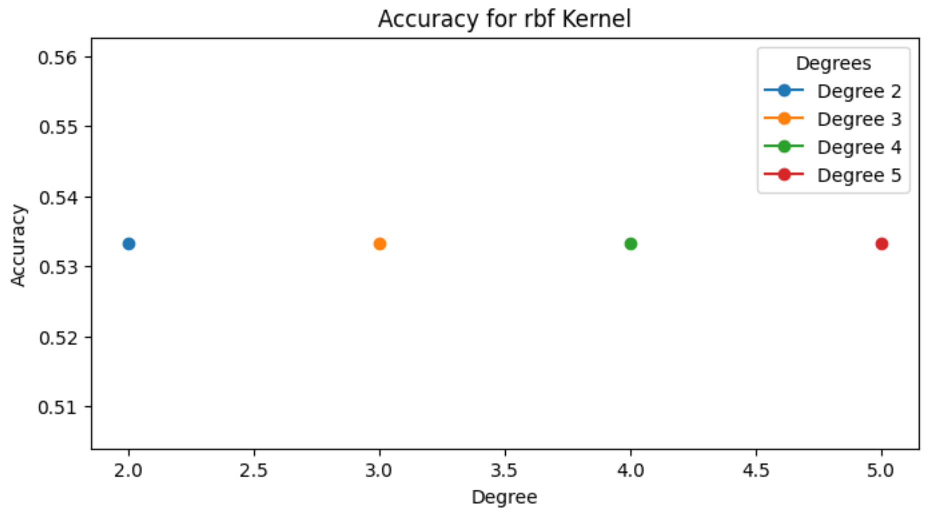
Accuracy Comparison by Kernel & Degrees

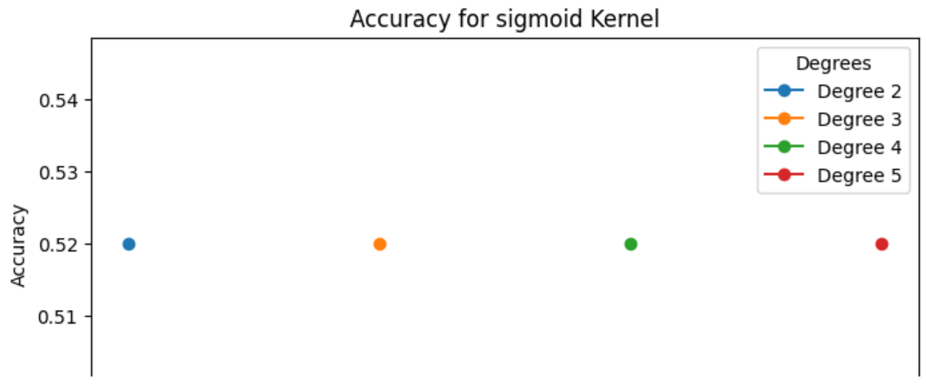
- Degrees Hyperparameter is only applicable to Kernel =polynomial.

```
# Visualize accuracy for different kernels and degrees
for kernel in kernels:
    plt.figure(figsize=(8, 4))
    for degree in degrees:
        subset = results_df[(results_df['Kernel'] == kernel) & (results_df['Degree'] == degree)]
        plt.plot(subset['Degree'], subset['Accuracy'], marker='o', label=f'Degree {degree}')

    plt.title(f'Accuracy for {kernel} Kernel')
    plt.xlabel('Degree')
    plt.ylabel('Accuracy')
    plt.legend(title='Degrees')
    plt.show()
```


Accuracy for linear Kernel
Degrees
Degree 2
Degree 3
Degree 4
Degree 5
Accuracy
0.57
0.56
0.55
0.54
0.53
0.52
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Degree


Accuracy for poly Kernel
Degrees
Degree 2
Degree 3
Degree 4
Degree 5
Accuracy
0.545
0.540
0.535
0.530
0.525
0.520
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Degree


Accuracy for rbf Kernel
Degrees
Degree 2
Degree 3
Degree 4
Degree 5
Accuracy
0.56
0.55
0.54
0.53
0.52
0.51
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Degree


Accuracy for sigmoid Kernel
Degrees
Degree 2
Degree 3
Degree 4
Degree 5
Accuracy
0.54
0.53
0.52
0.51

Summary

The project serves as a comprehensive example of SVM usage in a classification task, considering different kernels and degrees. It showcases how to evaluate SVM models and provides visualizations to aid in understanding the performance of different kernel configurations.

- Linear an Poly Kernel with degree 5 achieve the highest level of accuracy -54%
- Degree Hyper Parameter is only applicable to Kernel =Polynomial