Support Vector Machine

ISLP Ch 9

Dan Schumacher

Imports

In [114…

```
import ISLP
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
import matplotlib.pyplot as plt
from ISLP.svm import plot as plot_svm
from sklearn.linear_model import LogisticRegression
```

5. We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

(a) Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them. For instance, you can do this as follows: rng = np.random.default_rng(5) x1 = rng.uniform(size=500) - 0.5 x2 = rng.uniform(size=500) - 0.5 y = x1**2 - x2**2 > 0

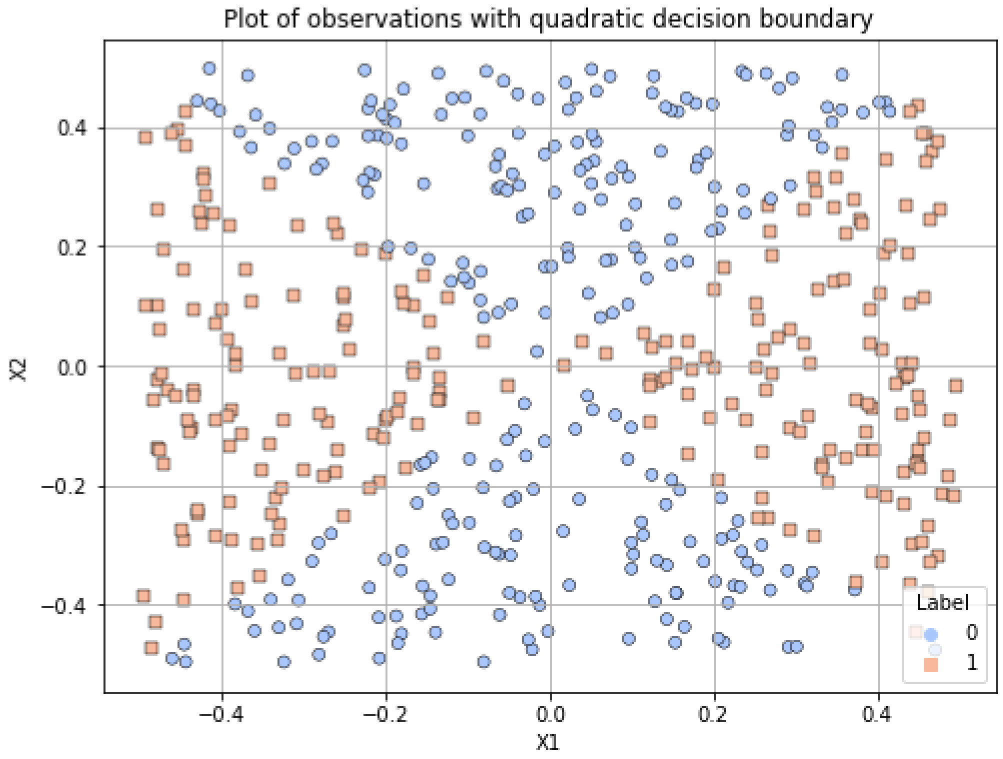
In [115…

```
# Seed for reproducibility
np.random.seed(27)

# Generate x1 and x2
x1 = np.random.uniform(size=500) - 0.5
x2 = np.random.uniform(size=500) - 0.5

# Generate labels y based on the quadratic decision boundary
y = (x1**2 - x2**2 > 0).astype(int)
```

(b) Plot the observations, colored according to their class labels. Your plot should display X1 on the x-axis, and X2 on the yaxis.

In [116…

```
# Seed for reproducibility
np.random.seed(11)

# Generate x1 and x2
x1 = np.random.uniform(size=500) - 0.5
x2 = np.random.uniform(size=500) - 0.5

# Generate labels y based on the quadratic decision boundary
y = (x1**2 - x2**2 > 0).astype(int)

# Create a DataFrame for easier plotting with Seaborn
data = pd.DataFrame({'X1': x1, 'X2': x2, 'Label': y})

# Create the plot using Seaborn
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data, x='X1', y='X2', hue='Label', palette='coolwarm', style='Lab

# Labeling the axes
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Plot of observations with quadratic decision boundary')
plt.grid(True)

# Show the plot
plt.show()
```



(c) Fit a logistic regression model to the data, using X1 and X2 as predictors.

In [117…

```
model = LogisticRegression()
X = data[['X1', 'X2']]
Y = data['Label']
model.fit(X, Y)
```

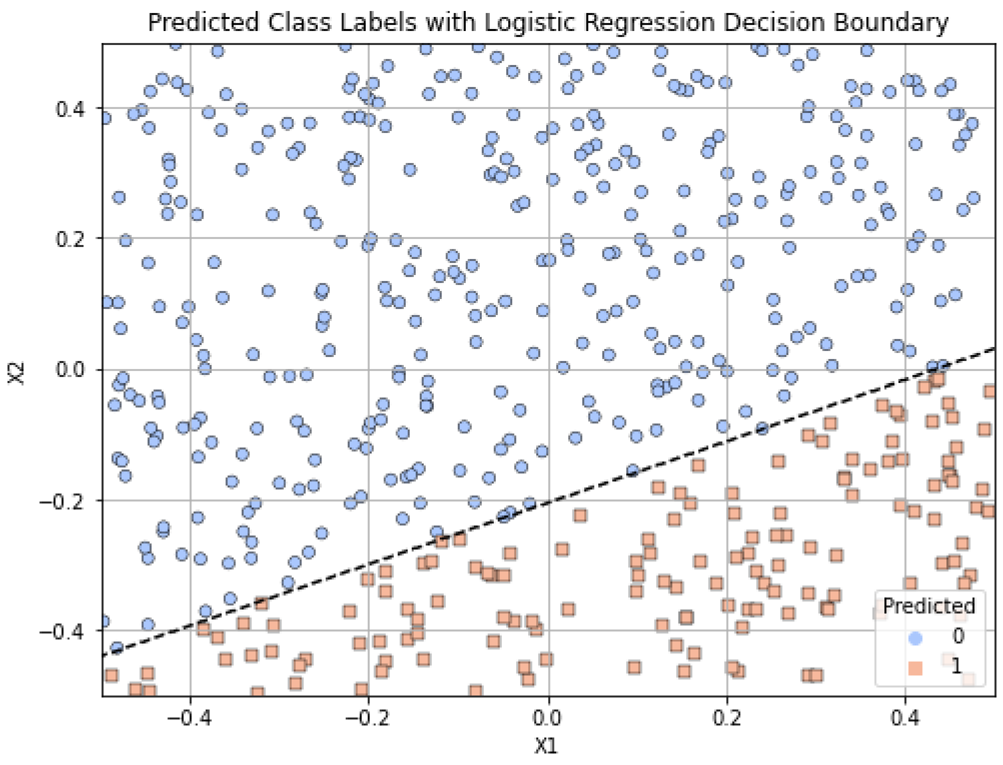
Out[117]:

```
▾ LogisticRegression
LogisticRegression()
```

(d) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The ecision boundary should be linear.

In [118…

```python
# Predict class labels for the observations
predicted_labels = model.predict(X)

# Update the DataFrame to include the predicted labels
data['Predicted'] = predicted_labels

# Plotting the observations based on predicted class labels
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data, x='X1', y='X2', hue='Predicted', palette='coolwarm', style=

# Create a mesh grid for plotting decision boundary
xx, yy = np.meshgrid(np.linspace(-0.5, 0.5, 500), np.linspace(-0.5, 0.5, 500))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.contour(xx, yy, Z, levels=[0.5], linestyles=['dashed'], colors=['black'])

# Labeling and showing the plot
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Predicted Class Labels with Logistic Regression Decision Boundary')
plt.grid(True)
plt.show()
```

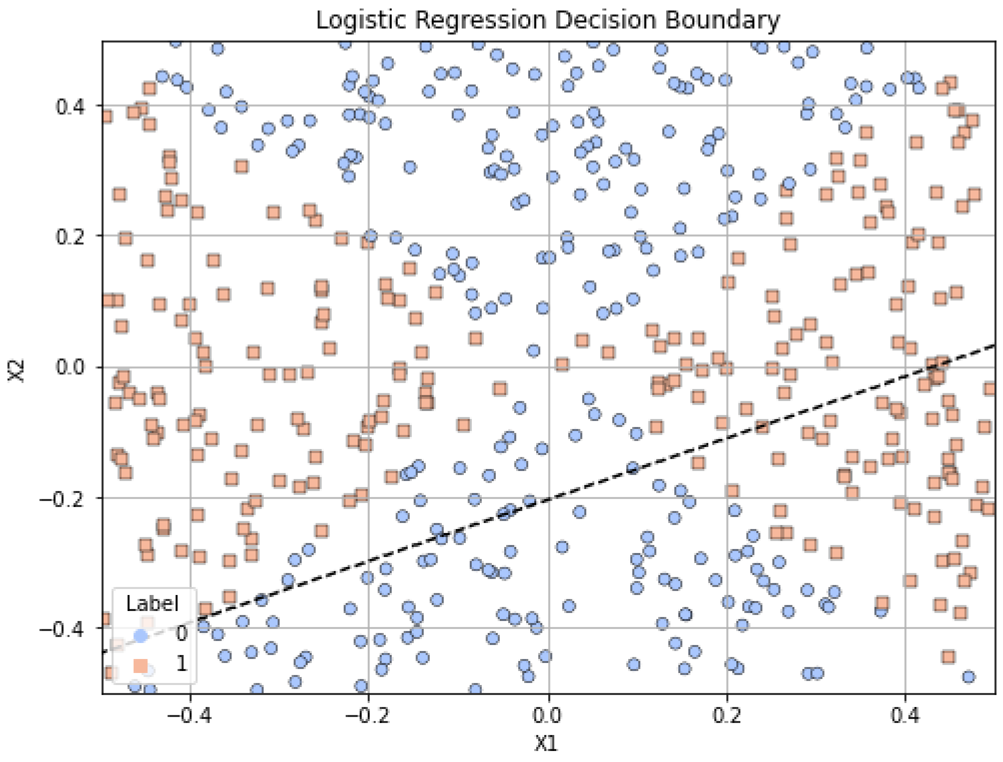


------------------- this isn't what it asked for, but I think this is more useful. -------------------

In [119…

```
warnings.filterwarnings(action='ignore', category=UserWarning, module='sklearn')

# Optional: Plotting the decision boundary
# Create a mesh grid
xx, yy = np.meshgrid(np.linspace(-0.5, 0.5, 500), np.linspace(-0.5, 0.5, 500))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data, x='X1', y='X2', hue='Label', palette='coolwarm', style='Lab
plt.contour(xx, yy, Z, levels=[0], linestyles=['dashed'], colors=['black'])
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Logistic Regression Decision Boundary')
plt.grid(True)
plt.show()
```

```
C:\Users\dansc\AppData\Local\Temp\ipykernel_5368\388126494.py:12: UserWarning: No con
tour levels were found within the data range.
  plt.contour(xx, yy, Z, levels=[0], linestyles=['dashed'], colors=['black'])
```



(e) Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors (e.g. X12, X1×X2, log(X2), nd so forth).

In [120…

```
from sklearn.linear_model import LogisticRegression

feature_dict = {
    'x1s': x1**2,
    'x2s': x2**2,
    'x1x2': x1*x2,
    'logx1': np.log(x1 + 0.5 + 1e-10), # Adding 0.5 to shift all values positive and a
    'logx2': np.log(x2 + 0.5 + 1e-10)
}

X_non_lin = pd.DataFrame(feature_dict)
Y = data['Label']

model = LogisticRegression()
model.fit(X_non_lin, Y)
```

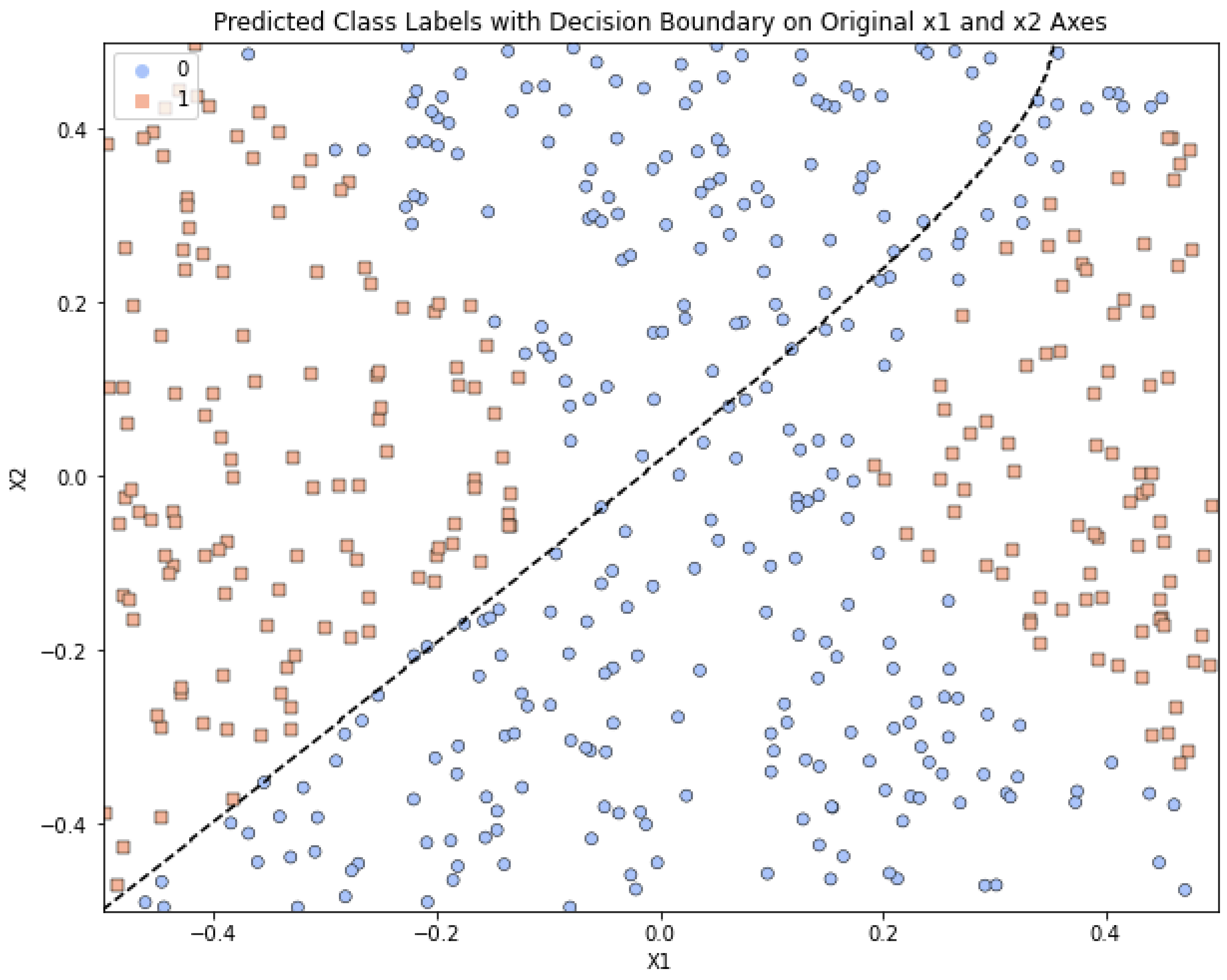
Out[120]:

```
▾ LogisticRegression
LogisticRegression()
```

(f) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The ecision boundary should be obviously non-linear. If it is not, hen repeat (a)–(e) until you come up with an example in which he predicted class labels are obviously non-linear.

In [121…

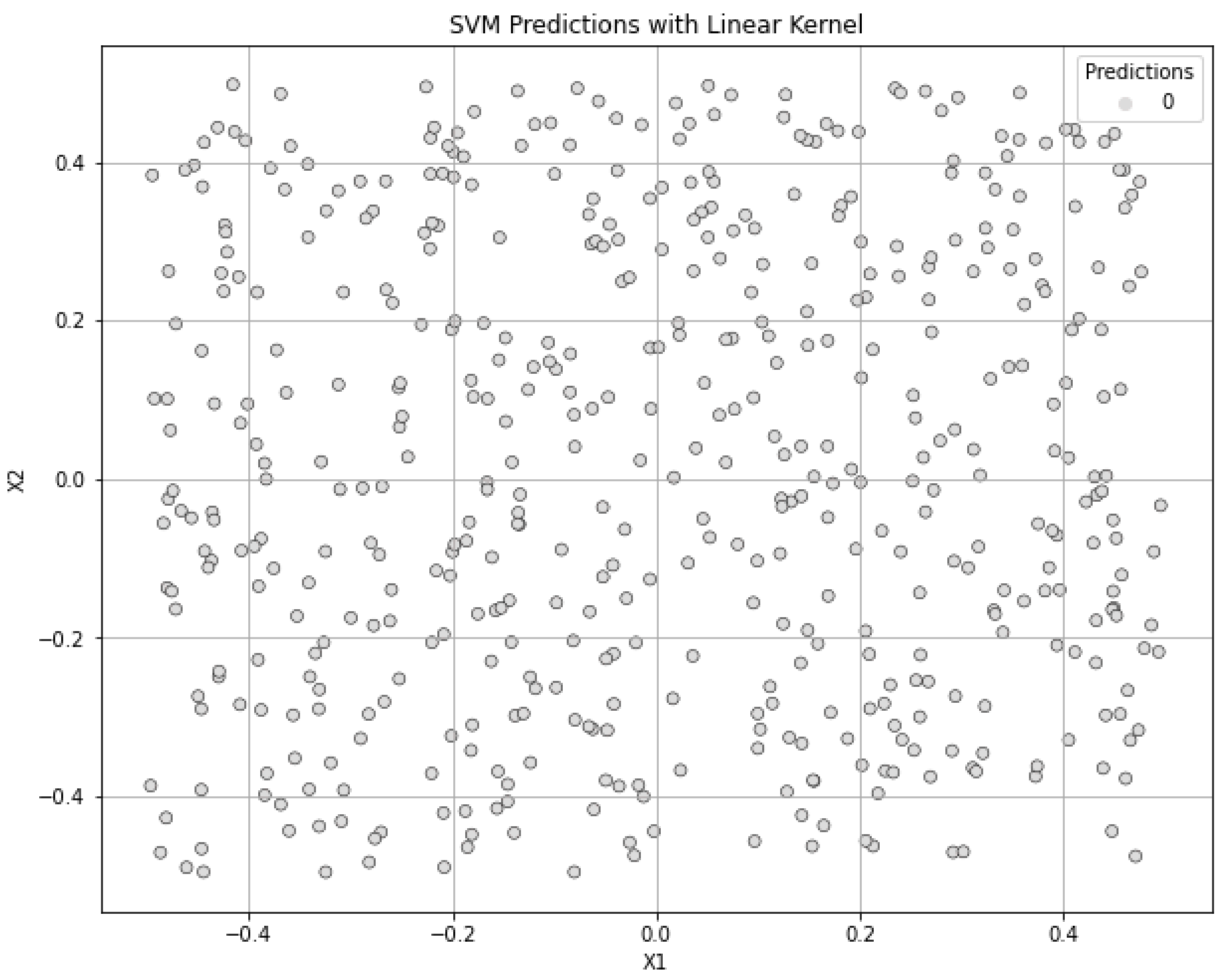
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.linear_model import LogisticRegression

# Predict the class labels for the original data
predicted_labels = model.predict(X_non_lin)

# Create a mesh grid for x1 and x2 directly
xx, yy = np.meshgrid(np.linspace(-0.5, 0.5, 500), np.linspace(-0.5, 0.5, 500))

# Transform this grid for prediction
grid = pd.DataFrame({
    'x1s': (xx.ravel() - 0.5)**2,
    'x2s': (yy.ravel() - 0.5)**2,
    'x1x2': (xx.ravel() - 0.5) * (yy.ravel() - 0.5),
    'logx1': np.log(xx.ravel() - 0.5 + 0.5 + 1 + 1e-10),
    'logx2': np.log(yy.ravel() - 0.5 + 0.5 + 1 +  1e-10)
})

Z = model.predict(grid).reshape(xx.shape)

# Plotting the predicted labels
plt.figure(figsize=(10, 8))
sns.scatterplot(x=x1, y=x2, hue=predicted_labels, style=predicted_labels, markers=['o'
plt.contour(xx, yy, Z, levels=[0.5], linestyles=['dashed'], colors=['black'])
plt.title('Predicted Class Labels with Decision Boundary on Original x1 and x2 Axes')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



(g) Fit a support vector classifier to the data with X1 and X2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted lass labels.

In [122…

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC

# Assuming X and Y have been defined and are available
svm = SVC(kernel='linear')
svm.fit(X, Y)
preds = svm.predict(X)

# Create a DataFrame for plotting
graph_me = X.copy()
graph_me['Predictions'] = preds  # Adding a new column for predicted labels

# Plotting
plt.figure(figsize=(10, 8))
sns.scatterplot(
    x='X1',
    y='X2',
    data=graph_me,
    hue='Predictions',  # Use this for coloring by predictions
    style='Predictions',  # Use this to vary markers by predictions
    palette='coolwarm',
#     markers=['o', 's'],
    edgecolor='k'
)
plt.title('SVM Predictions with Linear Kernel')
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid(True)
plt.show()
```



- plotting w/ linear kernel only predicts 0s! not good.

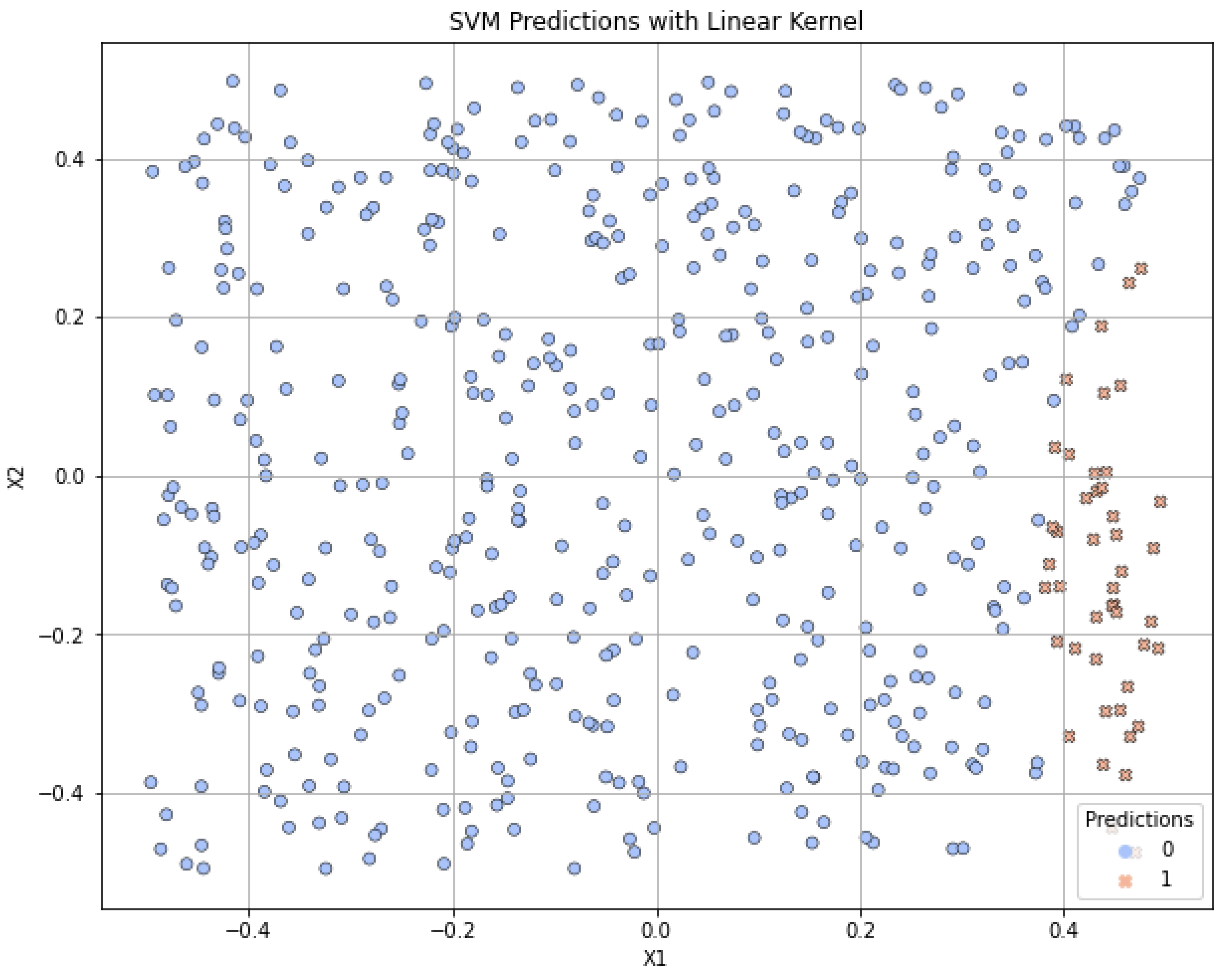
(h) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

Ploy Kernel

In [123…

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC

# Assuming X and Y have been defined and are available
svm = SVC(kernel='poly')
svm.fit(X, Y)
preds = svm.predict(X)

# Create a DataFrame for plotting
```

```
graph_me = X.copy()
graph_me['Predictions'] = preds  # Adding a new column for predicted labels

# Plotting
plt.figure(figsize=(10, 8))
sns.scatterplot(
    x='X1',
    y='X2',
    data=graph_me,
    hue='Predictions',  # Use this for coloring by predictions
    style='Predictions',  # Use this to vary markers by predictions
    palette='coolwarm',
#     markers=['o', 's'],
    edgecolor='k'
)
plt.title('SVM Predictions with Linear Kernel')
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid(True)
plt.show()
```

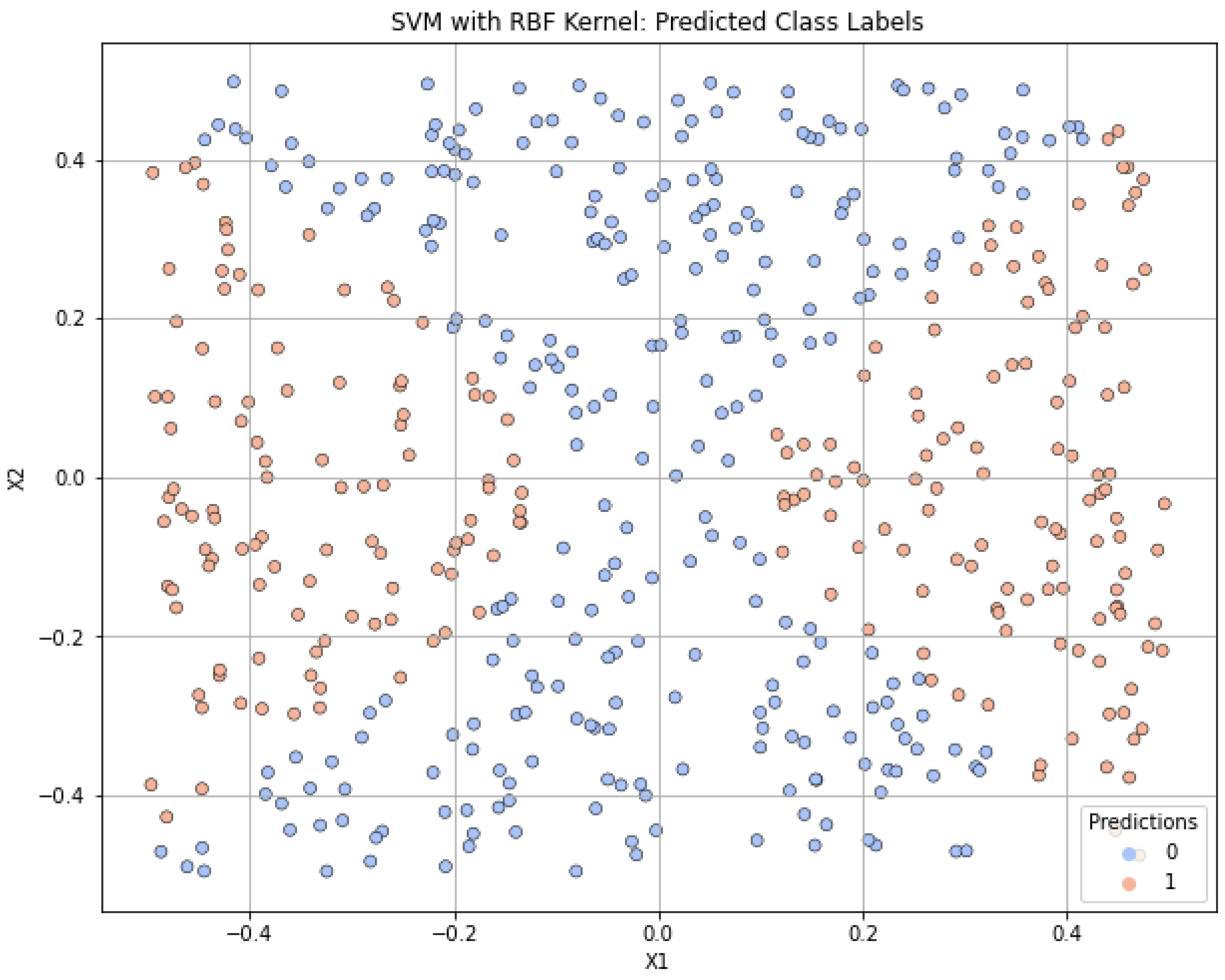


- Now we are getting 2 classes, but still not good results

In [124…

```
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
```

```python
from sklearn.pipeline import make_pipeline

# Check class distribution

# Use a non-linear kernel with parameter tuning
# Scaling the features
svm_pipeline = make_pipeline(StandardScaler(), SVC(kernel='rbf', C=1.0, gamma='scale')
svm_pipeline.fit(X, Y)

# Predict class labels
preds = svm_pipeline.predict(X)

# Update the DataFrame for plotting
graph_me['Predictions'] = preds

# Check the new distribution of predictions

# Plotting (the previous plotting code can be reused here)
plt.figure(figsize=(10, 8))
sns.scatterplot(
    x='X1',
    y='X2',
    data=graph_me,
    hue='Predictions',  # Color by the new predictions
    palette='coolwarm',
    edgecolor='k'
)
plt.title('SVM with RBF Kernel: Predicted Class Labels')
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid(True)
plt.show()
```



(i) Comment on your results

- The RBF kernel gave the best predictions returning a shape that most closely refelects the classifications of the actual data.

8. In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the Auto data set.

(a) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

In [125…

```
auto_df = ISLP.load_data('auto')
median_mpg = np.median(auto_df['mpg'])
auto_df['above_med_gas_mileage'] = [1 if x > median_mpg else 0 for x in auto_df['mpg']
```

(b) Fit a support vector classifier to the data with various values of C, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment

on your results. Note you will need to fit the classifier without the gas mileage variable to produce sensible results.

In [126…

```python
X = auto_df.drop(['mpg', 'above_med_gas_mileage', 'name'], axis=1)  # predictor featur
y = auto_df['above_med_gas_mileage']  # target

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define the SVC model
svc = SVC()

# Setup parameter grid for both linear and non-linear kernels
param_grid = [
    {'kernel': ['linear'], 'C': [0.01, 0.1, 1, 10, 100]},

]

# Setup GridSearchCV
grid_search = GridSearchCV(svc, param_grid, cv=5, scoring='accuracy', verbose=0)

# Fit GridSearchCV
grid_search.fit(X_scaled, y)
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

# Results from the grid search
results = pd.DataFrame(grid_search.cv_results_)
print(results[[ 'param_C', 'mean_test_score', 'std_test_score']])
```

```
Best parameters: {'C': 0.01, 'kernel': 'linear'}
Best cross-validation score: 0.90
  param_C  mean_test_score  std_test_score
0    0.01         0.900389        0.025223
1     0.1         0.875073        0.046895
2       1         0.852288        0.101522
3      10         0.857449        0.104920
4     100         0.857449        0.104920
```

- A 90% accuracy out of the box isn't bad at all! Maybe we can do better but still this is excellent for the first model

In [127…

```python
# Splitting data into train and test sets for final evaluation
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random

# Using the best estimator directly
best_svc_linear = grid_search.best_estimator_
best_svc_linear.fit(X_train, y_train)

# Predict on test data
y_pred = best_svc_linear.predict(X_test)

# Print performance metrics
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

Out[127]:

	0	1
0	33	9
1	1	36

- Seems like we have identify false positives for most of our mistakes

In [128…

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.79      0.87        42
           1       0.80      0.97      0.88        37

    accuracy                           0.87        79
   macro avg       0.89      0.88      0.87        79
weighted avg       0.89      0.87      0.87        79
```

(c) Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of gamma and degree and C. Comment on your results.

In [129…

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

# Scaling the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define the SVC model
svc = SVC()

# Setup parameter grid
# Note: 'gamma' can be a scale which is 1/(n_features * X.var()) as 'auto',
# 'scale' which is 1/n_features as default, or a float value.
param_grid = [
    {'kernel': ['rbf'], 'C': [0.1, 1, 10, 100], 'gamma': [0.01, 0.1, 1, 'scale']},
    {'kernel': ['poly'], 'C': [0.1, 1, 10, 100], 'gamma': [0.01, 0.1, 1, 'scale'], 'de
]

# Setup GridSearchCV
grid_search = GridSearchCV(svc, param_grid, cv=5, scoring='accuracy', verbose=0)

# Fit GridSearchCV
grid_search.fit(X_scaled, y)

# Fit GridSearchCV
grid_search.fit(X_scaled, y)
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
Best cross-validation score: 0.91
```

- This improved results by 1% Whoop!

In [130…

```
# Using the best estimator directly
best_svc_rbf_poly = grid_search.best_estimator_
best_svc_rbf_poly.fit(X_train, y_train)

# Predict on test data
y_pred = best_svc_rbf_poly.predict(X_test)

# Print performance metrics
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

Out[130]:

	0	1
0	32	10
1	0	37

- We still have 10 mistakes though, so Idk why it is showing our accuracy went up. Our single false neg transferred to a false pos.

In [131…

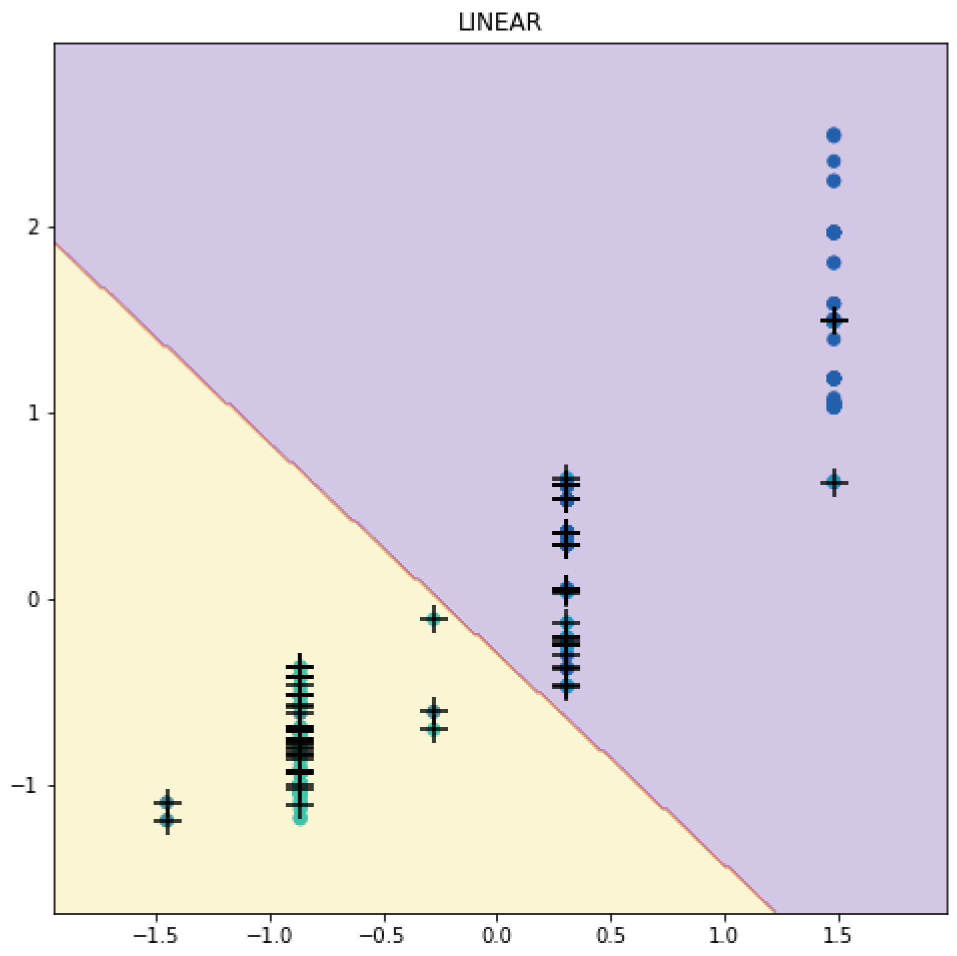
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.76      0.86        42
           1       0.79      1.00      0.88        37

    accuracy                           0.87        79
   macro avg       0.89      0.88      0.87        79
weighted avg       0.90      0.87      0.87        79
```

(d) Make some plots to back up your assertions in (b) and (c). Hint: In the lab, we used the plot_svm() function for fitted SVMs. When p > 2, you can use the keyword argument features to create plots displaying pairs of variables at a time.

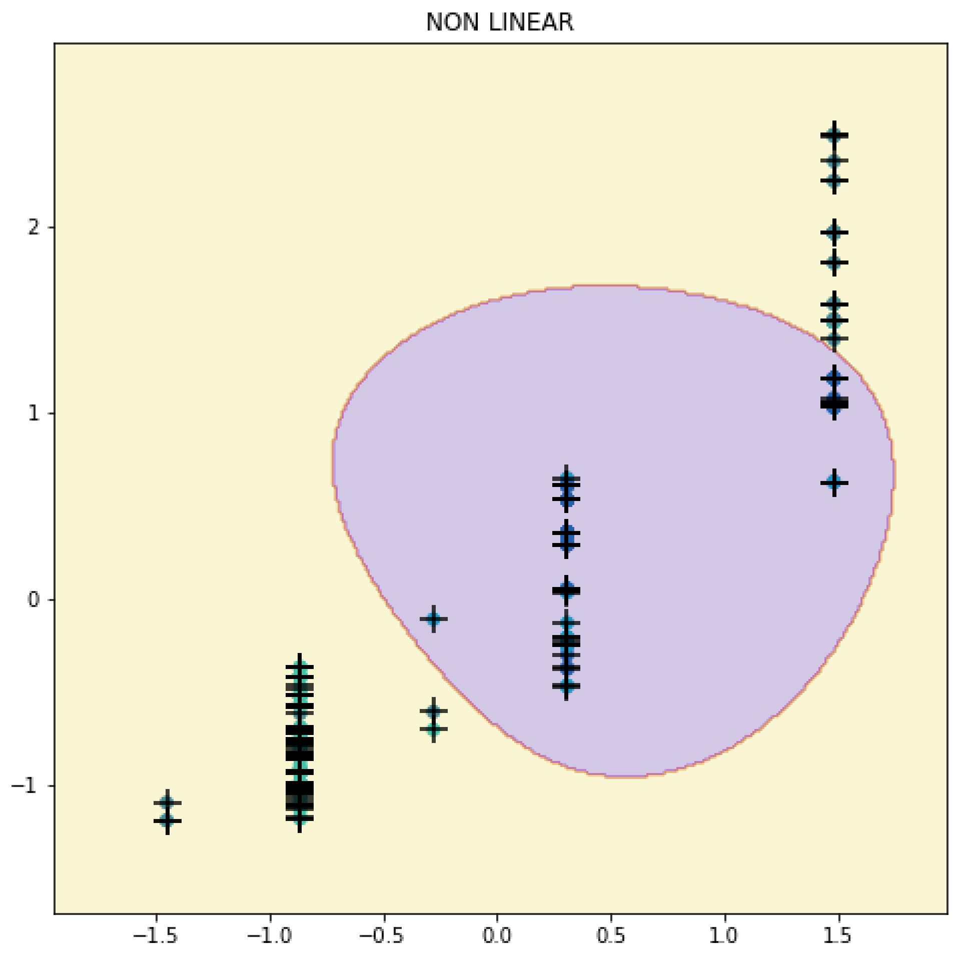
In [132…

```
# Visualization of decision boundaries - Adjust this if needed for feature dimensions
fig, ax = plt.subplots(figsize=(8, 8))

plot_svm(X_train, y_train, best_svc_linear, ax=ax)
plt.title('LINEAR')
plt.show()
```



In [133…

```
# Visualization of decision boundaries - Adjust this if needed for feature dimensions
fig, ax = plt.subplots(figsize=(8, 8))

plot_svm(X_train, y_train, best_svc_rbf_poly, ax=ax)
plt.title('NON LINEAR')
plt.show()
```



9. This problem involves the OJ data set which is part of the ISLP package.

In [134…
```
oj_df = ISLP.load_data('OJ')
```

In [135…
```
oj_df['Store7'] = np.where(oj_df['Store7'] == 'Yes',1,0)
```

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

In [136…
```
train, test = train_test_split(oj_df, test_size=len(oj_df) - 800, random_state=27)
```

In [137…
```
X_train = train.drop('Purchase', axis=1)
y_train = train['Purchase']

X_test = test.drop('Purchase', axis=1)
y_test = test['Purchase']
```

(b) Fit a support vector classifier to the training data using C = 0.01, with Purchase as the response and the other variables as predictors. How many support points are there?

In [138…
```
# Fit the SVC model
svc_model = SVC(C=0.01, kernel='linear')
```

```
svc_model.fit(X_train, y_train)

# Extract support vectors
support_vectors = svc_model.support_vectors_
print(f"Number of support vectors: {len(svc_model.support_)}")

# Detailed breakdown of support vectors per class
print(f"Support vectors per class: {svc_model.n_support_}")
```

```
Number of support vectors: 634
Support vectors per class: [318 316]
```

(c) What are the training and test error rates?

In [139…

```
from sklearn.metrics import accuracy_score

# Predict on the training data
y_train_pred = svc_model.predict(X_train)

# Calculate training accuracy and error rate
train_accuracy = accuracy_score(y_train, y_train_pred)
train_error_rate = 1 - train_accuracy

# Predict on the test data
y_test_pred = svc_model.predict(X_test)  # Assuming X_test is already defined and incl

# Calculate test accuracy and error rate
test_accuracy = accuracy_score(y_test, y_test_pred)
test_error_rate = 1 - test_accuracy

print(f"Training Error Rate: {train_error_rate:.2f}")
print(f"Test Error Rate: {test_error_rate:.2f}")
```

```
Training Error Rate: 0.22
Test Error Rate: 0.19
```

(d) Use cross-validation to select an optimal C. Consider values in the range 0.01 to 10.

In [140…

```
param_grid = {'C': [0.01, 0.1, 1, 5, 10]}

# Setup the grid search with cross-validation
optim_c_svc = GridSearchCV(estimator=SVC(kernel='linear'), param_grid=param_grid, cv=5

# Fit the grid search to the data
optim_c_svc.fit(X_train, y_train)  # Changed from X_train_features to X_train

# Print out the best C value and cross-validation score
print(f"Best C value: {optim_c_svc.best_params_['C']}")
print(f"Best cross-validation score: {optim_c_svc.best_score_:.2f}")
```

```
Best C value: 10
Best cross-validation score: 0.82
```

(e) Compute the training and test error rates using this new value for C.

In [141…

```
# Predict on the test data
y_test_pred = optim_c_svc.predict(X_test)  # Assuming X_test is already defined and in
```

```
# Calculate test accuracy and error rate
test_accuracy = accuracy_score(y_test, y_test_pred)
test_error_rate = 1 - test_accuracy

print(f"Test Error Rate: {test_error_rate:.2f}")
```

```
Test Error Rate: 0.16
```

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

In [142…

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score

# Assuming all necessary imports are done and the data is already loaded and split int

# Define the parameter range for C with an RBF kernel
param_grid = [
    {'kernel': ['rbf'], 'C': [0.1, 1, 10, 100]}
]

# Set up the grid search with cross-validation for an SVC with a radial kernel
dif_kernels = GridSearchCV(estimator=SVC(), param_grid=param_grid, cv=5, scoring='accu

# Fit the grid search to the data
dif_kernels.fit(X_train, y_train)

# Print out the best parameters and cross-validation score
print(f"Best parameters: {dif_kernels.best_params_}")
print(f"Best cross-validation score: {dif_kernels.best_score_:.2f}")

# Using the best estimator to predict on test data
y_test_pred = dif_kernels.predict(X_test)

# Calculate test accuracy and error rate
test_accuracy = accuracy_score(y_test, y_test_pred)
test_error_rate = 1 - test_accuracy

print(f"Test Error Rate: {test_error_rate:.2f}")
```

```
Best parameters: {'C': 0.1, 'kernel': 'rbf'}
Best cross-validation score: 0.60
Test Error Rate: 0.37
```

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree = 2.

In [143…

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score

# Assuming the data has already been preprocessed and split into X_train, y_train, X_t

# Define the parameter range for C with a polynomial kernel of degree 2
param_grid = {
    'kernel': ['poly'],
    'degree': [2],  # Polynomial degree
    'C': [0.1, 1, 10, 100]  # Range of C values to try
```

```python
}

# Setup the grid search with cross-validation
poly_kernel_svc = GridSearchCV(estimator=SVC(), param_grid=param_grid, cv=5, scoring='

# Fit the grid search to the data
poly_kernel_svc.fit(X_train, y_train)

# Print out the best parameters and cross-validation score
print(f"Best parameters: {poly_kernel_svc.best_params_}")
print(f"Best cross-validation score: {poly_kernel_svc.best_score_:.2f}")

# Using the best estimator to predict on test data
y_test_pred = poly_kernel_svc.predict(X_test)

# Calculate test accuracy and error rate
test_accuracy = accuracy_score(y_test, y_test_pred)
test_error_rate = 1 - test_accuracy

print(f"Test Error Rate: {test_error_rate:.2f}")
```

```
Best parameters: {'C': 0.1, 'degree': 2, 'kernel': 'poly'}
Best cross-validation score: 0.60
Test Error Rate: 0.37
```

(h) Overall, which approach seems to give the best results on this data

- Linear kernal with a C of 10 gave an 84% acc on the test data!.