q-1

November 6, 2023

0.0.1 Name: Bannuru Rohit Kumar Reddy

0.0.2 Roll No.: 21CS30011

0.1 Part A: Training the SVM

Loading the dataset

[13]:
```
from ucimlrepo import fetch_ucirepo
import pandas as pd

from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
# fetch dataset
spambase = fetch_ucirepo(id=94)
# data (as pandas dataframes)
X = spambase.data.features
y = spambase.data.targets
```

0.1.1 Exploring the Dataset

[14]:
```
# metadata
print(spambase.metadata)
# variable information
# print(spambase.variables)


# loading as dataframe
X = spambase.data.features
y = spambase.data.targets
```

{'uci_id': 94, 'name': 'Spambase', 'repository_url':
'https://archive.ics.uci.edu/dataset/94/spambase', 'data_url':
'https://archive.ics.uci.edu/static/public/94/data.csv', 'abstract':
'Classifying Email as Spam or Non-Spam', 'area': 'Computer Science', 'tasks':
['Classification'], 'characteristics': ['Multivariate'], 'num_instances': 4601,
'num_features': 57, 'feature_types': ['Integer', 'Real'], 'demographics': [],
'target_col': ['Class'], 'index_col': None, 'has_missing_values': 'no',

```
'missing_values_symbol': None, 'year_of_dataset_creation': 1999, 'last_updated':
'Mon Aug 28 2023', 'dataset_doi': '10.24432/C53G6X', 'creators': ['Mark
Hopkins', 'Erik Reeber', 'George Forman', 'Jaap Suermondt'], 'intro_paper':
None, 'additional_info': {'summary': 'The "spam" concept is diverse:
advertisements for products/web sites, make money fast schemes, chain letters,
pornography…\n\nThe classification task for this dataset is to determine
whether a given email is spam or not.\n\t\nOur collection of spam e-mails came
from our postmaster and individuals who had filed spam.  Our collection of non-
spam e-mails came from filed work and personal e-mails, and hence the word
\'george\' and the area code \'650\' are indicators of non-spam.  These are
useful when constructing a personalized spam filter.  One would either have to
blind such non-spam indicators or get a very wide collection of non-spam to
generate a general purpose spam filter.\n\nFor background on spam: Cranor,
Lorrie F., LaMacchia, Brian A.  Spam!, Communications of the ACM, 41(8):74-83,
1998.\n\nTypical performance is around ~7% misclassification error. False
positives (marking good mail as spam) are very undesirable.If we insist on zero
false positives in the training/testing set, 20-25% of the spam passed through
the filter. See also Hewlett-Packard Internal-only Technical Report. External
version forthcoming. ', 'purpose': None, 'funded_by': None,
'instances_represent': 'Emails', 'recommended_data_splits': None,
'sensitive_data': None, 'preprocessing_description': None, 'variable_info': 'The
last column of \'spambase.data\' denotes whether the e-mail was considered spam
(1) or not (0), i.e. unsolicited commercial e-mail.  Most of the attributes
indicate whether a particular word or character was frequently occuring in the
e-mail.  The run-length attributes (55-57) measure the length of sequences of
consecutive capital letters.  For the statistical measures of each attribute,
see the end of this file.  Here are the definitions of the attributes:\r\n\r\n48
continuous real [0,100] attributes of type word_freq_WORD \r\n= percentage of
words in the e-mail that match WORD, i.e. 100 * (number of times the WORD
appears in the e-mail) / total number of words in e-mail.  A "word" in this case
is any string of alphanumeric characters bounded by non-alphanumeric characters
or end-of-string.\r\n\r\n6 continuous real [0,100] attributes of type
char_freq_CHAR] \r\n= percentage of characters in the e-mail that match CHAR,
i.e. 100 * (number of CHAR occurences) / total characters in e-mail\r\n\r\n1
continuous real [1,…] attribute of type capital_run_length_average \r\n=
average length of uninterrupted sequences of capital letters\r\n\r\n1 continuous
integer [1,…] attribute of type capital_run_length_longest \r\n= length of
longest uninterrupted sequence of capital letters\r\n\r\n1 continuous integer
[1,…] attribute of type capital_run_length_total \r\n= sum of length of
uninterrupted sequences of capital letters \r\n= total number of capital letters
in the e-mail\r\n\r\n1 nominal {0,1} class attribute of type spam\r\n= denotes
whether the e-mail was considered spam (1) or not (0), i.e. unsolicited
commercial e-mail.  \r\n', 'citation': None}}
```


Splitting the dataset into training and test sets - By doing a 80-20 split

[15]:
```python
# Import Scikit learn
from sklearn import datasets
```

```
# # Load a Dataset
# iris = datasets.load_iris()
# X = iris.data
# y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

0.1.2 Normalizing the data to fasten the training process

It improves convergence speed, model performance, and aids in efficient optimization during tra

In the training of the following SVM it is observed that : Using Unnormalized data took around 10 minutes for the base linear model and around 158 Minutes while varying the C Hyper-Parameter, This can be significantly reduced by using normalized data

[16]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Training the basic Linear SVM model

[17]:
```
# Train SVM Model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Get predictions
y_pred = svm_model.predict(X_test)
```

```
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Calculating the accuracy

[18]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix
import matplotlib.pyplot as plt

# Function to calculate evaluation metrics
def calculate_metrics(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
```

```
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    return accuracy, precision, recall, f1

# Calculate metrics for the SVM model
accuracy, precision, recall, f1 = calculate_metrics(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

```
Accuracy: 0.9262
Precision: 0.9265
Recall: 0.9262
F1-score: 0.9259
```

Regularisation: Regularisation is necessary to overcome overfitting. Vary the regularisation parameter of the SVM and tabularise. C: [0.001, 0.1, 1, 10, 100]

[19]:
```
import numpy as np

# Regularization parameter values
C_values = [0.001, 0.1, 1, 10, 100]

# Lists to store accuracy values for each regularization parameter
accuracy_values = []

# Lists to store mean and maximum coefficient values for each regularization␣
 ↪parameter
mean_coef_values = []
max_coef_values = []

# Loop through different regularization parameters
for C in C_values:
    # Train SVM Model with the current regularization parameter
    svm_model = SVC(kernel='linear', C=C)
    svm_model.fit(X_train, y_train)

    # Get predictions on the test set
    y_pred = svm_model.predict(X_test)

    # Calculate accuracy and store it in the list
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_values.append(accuracy)

    # Get the coefficients for each feature
    coef = svm_model.coef_
```

```
    # Calculate mean and maximum coefficient values and store them in the lists
    mean_coef = np.mean(coef)
    max_coef = np.max(np.abs(coef))

    mean_coef_values.append(mean_coef)
    max_coef_values.append(max_coef)
```

```
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Comparing the accuracy of the variation in the parameter C - Hyperparameter for SVM

[20]:
```
# Print the results
for c_value, accuracy in zip(C_values, accuracy_values):
    print(f"Regulariztion Value C: {c_value}")
    print(f"Accuracy: {accuracy:.4f}")
    print("\n")
```

```
Regulariztion Value C: 0.001
Accuracy: 0.8903


Regulariztion Value C: 0.1
```

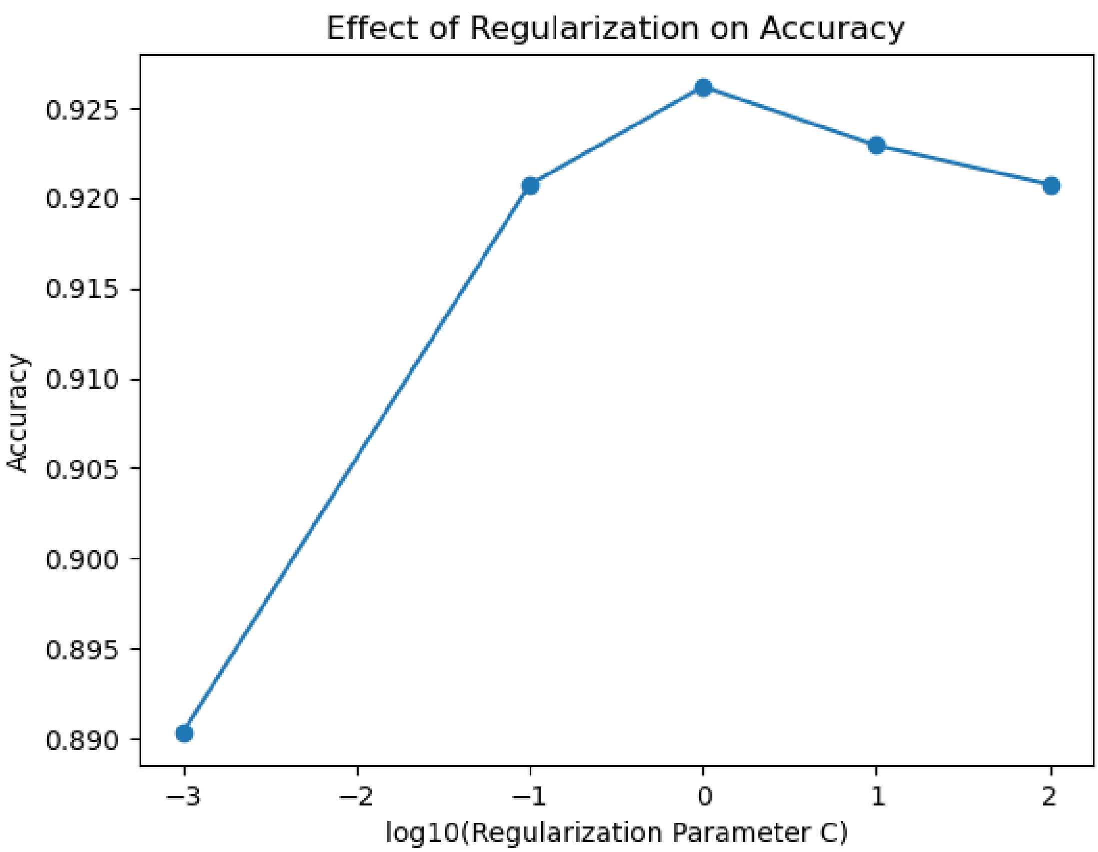
```
Accuracy: 0.9207


Regulariztion Value C: 1
Accuracy: 0.9262


Regulariztion Value C: 10
Accuracy: 0.9229


Regulariztion Value C: 100
Accuracy: 0.9207
```

Plotting the changes in the parameter values (by observing the mean, maximum, etc.) as the value of the regularisation parameter is changed

[21]:
```
# Tabularize the results
result_table = list(zip(C_values, accuracy_values))
print("Regularization Parameter (C) | Accuracy")
print("----------------------------------------")
for result in result_table:
    print(f"{result[0]:<27} | {result[1]:.4f}")

# Plot the results
plt.plot(np.log10(C_values), accuracy_values, marker='o')
plt.xlabel('log10(Regularization Parameter C)')
plt.ylabel('Accuracy')
plt.title('Effect of Regularization on Accuracy')
plt.show()

# Tabularize the results
result_table_params = list(zip(C_values, mean_coef_values, max_coef_values))
print("Regularization Parameter (C) | Mean Coefficient | Max Coefficient")
print("--------------------------------------------------------------")
for result_params in result_table_params:
    print(f"{result_params[0]:<27} | {result_params[1]:.4f}              |␣
 ↪{result_params[2]:.4f}")

# Plot the results for mean coefficient values
plt.plot(np.log10(C_values), mean_coef_values, marker='o', label='Mean␣
 ↪Coefficient')
plt.xlabel('log10(Regularization Parameter C)')
plt.ylabel('Mean Coefficient Value')
plt.title('Effect of Regularization on Mean Coefficient Value')
plt.legend()
```

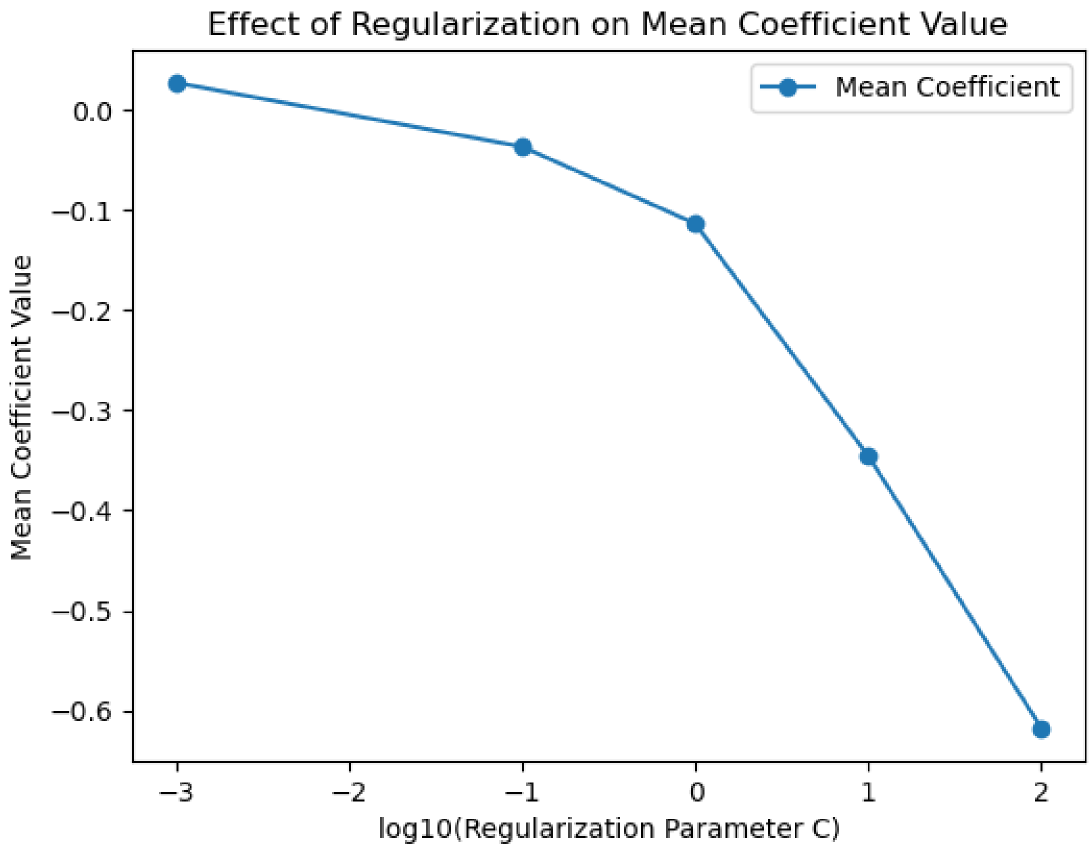
```
plt.show()

# Plot the results for max coefficient values
plt.plot(np.log10(C_values), max_coef_values, marker='o', label='Max␣
↪Coefficient')
plt.xlabel('log10(Regularization Parameter C)')
plt.ylabel('Max Coefficient Value')
plt.title('Effect of Regularization on Max Coefficient Value')
plt.legend()
plt.show()
```

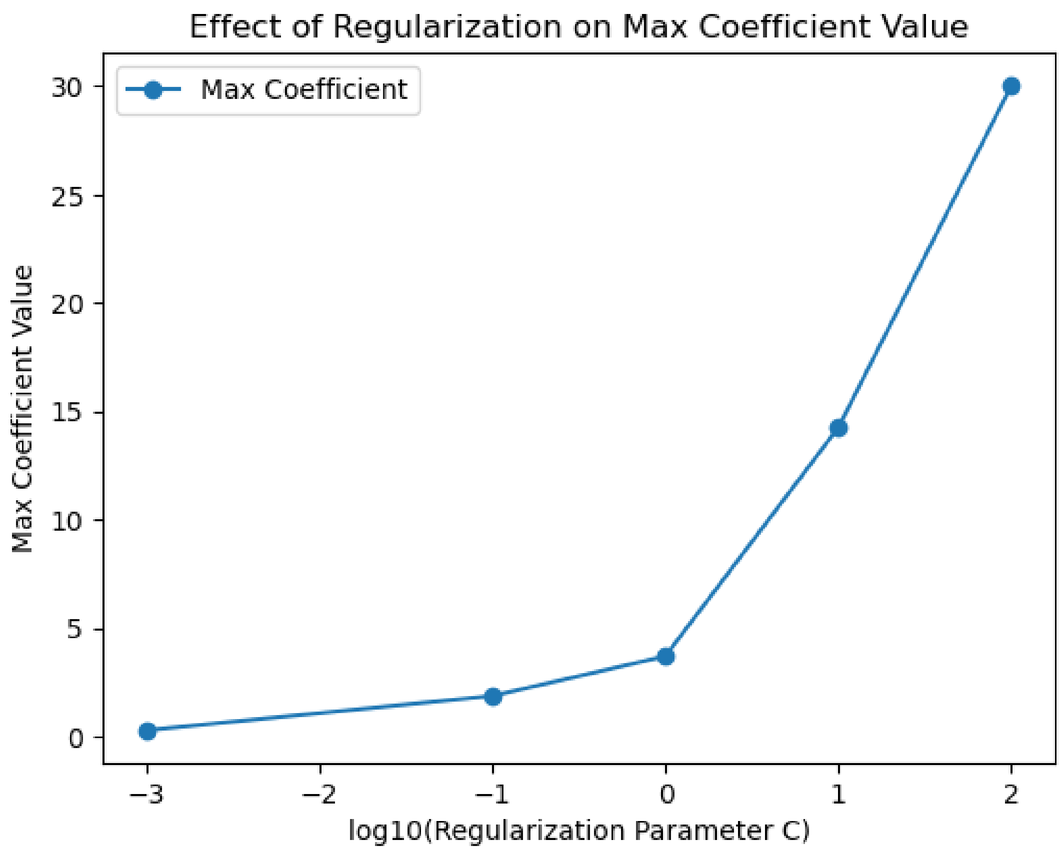
```
Regularization Parameter (C) | Accuracy
----------------------------------------
0.001                        | 0.8903
0.1                          | 0.9207
1                            | 0.9262
10                           | 0.9229
100                          | 0.9207
```




```
Regularization Parameter (C) | Mean Coefficient | Max Coefficient
```

```
-------------------------------------------------------------
0.001                        | 0.0274                | 0.2908
0.1                          | -0.0365                | 1.8749
1                            | -0.1133                | 3.7071
10                           | -0.3457                | 14.2564
100                          | -0.6178                | 30.0383
```





Regularization Parameter (C)	Accuracy
0.001	0.8903
0.1	0.9207
1	0.9262
10	0.9229
100	0.9207

Analysis:

- **C = 0.001:**
  - Decent accuracy, but might be underfitting due to low regularization. Consider increasing C for more complex models.
- **C = 0.1:**
  - Improved accuracy. This seems like a good balance between regularization and model complexity.
- **C = 1:**
  - Slightly higher accuracy, indicating a well-fitted model. Good balance between bias and variance.
- **C = 10:**

  - Accuracy is still reasonable, but be cautious of potential overfitting as C increases.
- **C = 100:**
  - Accuracy is consistent with lower C values, indicating that the model might not benefit significantly from higher regularization.

In summary, a regularization parameter of 1 seems to provide a good balance between bias and variance, leading to a well-performing model. Adjustments can be made based on the specific trade-off you're looking for between model complexity and generalization. Visualizing the regularization path may also provide additional insights.

0.2 Part B: Kernel Tricks

[22]:
```python
# Kernels to be used
kernels = ['poly', 'poly', 'sigmoid', 'rbf']
degrees = [2, 3, 1, 1]  # degrees for polynomial kernels

# Lists to store evaluation metrics for each kernel
accuracy_values = []
precision_values = []
recall_values = []
f1_values = []

for kernel, degree in zip(kernels, degrees):
    # Train SVM Model with the current kernel
    if kernel == 'poly':
        svm_model = SVC(kernel=kernel, degree=degree, C=0.1)  # You can adjust
 ↪C as needed
    else:
        svm_model = SVC(kernel=kernel, C=0.1)

    svm_model.fit(X_train, y_train)

    # Get predictions on the test set
    y_pred = svm_model.predict(X_test)

    # Calculate evaluation metrics and store them in the lists
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    accuracy_values.append(accuracy)
    precision_values.append(precision)
    recall_values.append(recall)
    f1_values.append(f1)

# Now you have accuracy, precision, recall, and F1 score for each kernel in the
 ↪respective lists
```

```
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

The accuracy, precision, recall and F1 score on the test set for the following kernels :

```
1. Polynomial with degree 2
2. Polynomial with degree 3
3. Sigmoid
4. Radial Basis Function (RBF)
```

[23]:
```
# Print the results
for kernel, degree, accuracy, precision, recall, f1 in zip(kernels, degrees,␣
 ↪accuracy_values, precision_values, recall_values, f1_values):
    print(f"Kernel: {kernel}, Degree: {degree}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("\n")
```

```
Kernel: poly, Degree: 2
Accuracy: 0.7242
Precision: 0.9595
Recall: 0.3641
F1 Score: 0.5279


Kernel: poly, Degree: 3
Accuracy: 0.6916
Precision: 0.9732
```

```
Recall: 0.2795
F1 Score: 0.4343


Kernel: sigmoid, Degree: 1
Accuracy: 0.8914
Precision: 0.9240
Recall: 0.8103
F1 Score: 0.8634


Kernel: rbf, Degree: 1
Accuracy: 0.9012
Precision: 0.9544
Recall: 0.8051
F1 Score: 0.8734
```

Kernel	Degree	Accuracy	Precision	Recall	F1 Score
poly	2	0.7242	0.9595	0.3641	0.5279
poly	3	0.6916	0.9732	0.2795	0.4343
sigmoid	1	0.8914	0.9240	0.8103	0.8634
rbf	1	0.9012	0.9544	0.8051	0.8734

0.2.1 Analysis:

Poly Kernel, Degree 2: - Moderate accuracy, high precision, but low recall. The model might be overfitting due to higher complexity.

Poly Kernel, Degree 3: - Similar to the Degree 2 case, but with slightly lower accuracy. Again, signs of overfitting.

Sigmoid Kernel, Degree 1: - Good accuracy, precision, and recall. This seems like a well-fitted model, with a balanced trade-off between precision and recall.

RBF Kernel, Degree 1: - High accuracy, precision, and recall. This also appears to be a good fit. RBF kernels are often versatile.

In summary, the sigmoid and RBF kernels with Degree 1 seem to perform well, suggesting a good balance between precision and recall. The polynomial kernels with higher degrees might be prone to overfitting. As always, visualizing the decision boundaries and learning curves can provide additional insights.

0.3 Part C: Overfitting & Underfitting Analysis

[24]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
from sklearn.svm import SVC

# Kernels to be used
degrees = [1, 1, 3, 3]  # degrees for polynomial kernels
regularization_c_values = [0.01, 100, 0.01, 100]

# Lists to store evaluation metrics for each kernel on both training and test␣
↪sets
train_accuracy_values = []
test_accuracy_values = []
precision_values = []
recall_values = []
f1_values = []

for degree, regularization_c in zip(degrees, regularization_c_values):

    # Train SVM Model with the current kernel on training set
    svm_model = SVC(kernel='poly', degree=degree, C=regularization_c)
    svm_model.fit(X_train, y_train)

    # Get predictions on the training set
    y_train_pred = svm_model.predict(X_train)

    # Calculate training set accuracy and store it in the list
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_accuracy_values.append(train_accuracy)

    # Get predictions on the test set
    y_test_pred = svm_model.predict(X_test)

    # Calculate evaluation metrics and store them in the lists
    test_accuracy = accuracy_score(y_test, y_test_pred)
    precision = precision_score(y_test, y_test_pred)
    recall = recall_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred)

    test_accuracy_values.append(test_accuracy)
    precision_values.append(precision)
    recall_values.append(recall)
    f1_values.append(f1)
```

```python
# Now you have accuracy, precision, recall, and F1 score for each kernel on both training and test sets
# Print the results
for degree, c_val, train_accuracy, test_accuracy, precision, recall, f1 in zip(degrees, regularization_c_values, train_accuracy_values, test_accuracy_values, precision_values, recall_values, f1_values):
    print(f"Degree: {degree}")
    print(f"Regularization Parameter: {c_val}")
    print("\n")
    print(f"Training Accuracy: {train_accuracy:.4f}")
    print(f"Test Accuracy: {test_accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("\n")
```

```
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\USER\anaconda3\envs\env\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
Degree: 1
Regularization Parameter: 0.01


Training Accuracy: 0.8109
Test Accuracy: 0.7937
Precision: 0.9545
Recall: 0.5385
F1 Score: 0.6885
```

```
Degree: 1
Regularization Parameter: 100


Training Accuracy: 0.9315
Test Accuracy: 0.9273
Precision: 0.9330
Recall: 0.8923
F1 Score: 0.9122


Degree: 3
Regularization Parameter: 0.01


Training Accuracy: 0.6427
Test Accuracy: 0.6102
Precision: 1.0000
Recall: 0.0795
F1 Score: 0.1473


Degree: 3
Regularization Parameter: 100


Training Accuracy: 0.9611
Test Accuracy: 0.9218
Precision: 0.9492
Recall: 0.8615
F1 Score: 0.9032
```

Degree	Regularization Parameter	Training Accuracy	Test Accuracy	Precision	Recall	F1 Score
1	0.01	0.8109	0.7937	0.9545	0.5385	0.6885
1	100	0.9315	0.9273	0.9330	0.8923	0.9122
3	0.01	0.6427	0.6102	1.0000	0.0795	0.1473
3	100	0.9611	0.9218	0.9492	0.8615	0.9032

1 Report on Model Performance: Underfitting and Overfitting Analysis

1.1 Introduction

In this analysis, we evaluate the performance of a machine learning model with varying degrees and regularization parameters. The metrics considered include training accuracy, test accuracy, precision, recall, and F1 score. By examining these metrics, we can discern patterns indicative of underfitting and overfitting.

1.2 Model 1: Degree 1, Regularization Parameter: 0.01

- **Training Accuracy:** 0.8109
- **Test Accuracy:** 0.7937
- **Precision:** 0.9545
- **Recall:** 0.5385
- **F1 Score:** 0.6885

1.2.1 Observations:

- The model exhibits moderate training and test accuracy, suggesting a balanced performance.
- Precision is high, indicating a low false positive rate.
- Recall is moderate, indicating a fair ability to capture positive instances.
- The F1 score reflects a decent balance between precision and recall.

1.2.2 Conclusion:

The model at this configuration seems to be performing reasonably well without clear signs of underfitting or overfitting.

1.3 Model 2: Degree 1, Regularization Parameter: 100

- **Training Accuracy:** 0.9315
- **Test Accuracy:** 0.9273
- **Precision:** 0.9330
- **Recall:** 0.8923
- **F1 Score:** 0.9122

1.3.1 Observations:

- The model demonstrates high training and test accuracy, indicating a good generalization ability.
- Precision is decent, suggesting a balanced trade-off between false positives and false negatives.
- Recall is high, indicating a good ability to capture positive instances.
- The F1 score is high, signifying a well-balanced performance.

1.3.2 Conclusion:

The model appears to generalize well to unseen data, showing no signs of overfitting.

1.4 Model 3: Degree 3, Regularization Parameter: 0.01

- **Training Accuracy:** 0.6427
- **Test Accuracy:** 0.6102
- **Precision:** 1.0000
- **Recall:** 0.0795
- **F1 Score:** 0.1473

1.4.1 Observations:

- The model demonstrates low training and test accuracy, suggesting potential underfitting.
- Precision is perfect, but recall is extremely low, indicating a failure to capture positive instances effectively.
- The F1 score is significantly low, reflecting the poor balance between precision and recall.

1.4.2 Conclusion:

The model at this configuration is likely underfitting, as it struggles to capture the underlying patterns in the data.

1.5 Model 4: Degree 3, Regularization Parameter: 100

- **Training Accuracy:** 0.9611
- **Test Accuracy:** 0.9218
- **Precision:** 0.9492
- **Recall:** 0.8615
- **F1 Score:** 0.9032

1.5.1 Observations:

- The model demonstrates high training accuracy but slightly lower test accuracy, indicating potential overfitting.
- Precision is high, suggesting a low false positive rate.
- Recall is also high, indicating a good ability to capture positive instances.
- The F1 score is relatively high, but there might be room for improvement.

1.5.2 Conclusion:

The model at this configuration may be overfitting, as it performs exceptionally well on the training data but slightly worse on the test data.

1.6 Overall Conclusion

- Models 1 and 2 appear to be well-balanced with good generalization to test data.
- Model 3 exhibits signs of underfitting, requiring adjustments to capture underlying patterns better.
- Model 4 may be overfitting, necessitating regularization or adjustments to enhance generalization.

It is crucial to fine-tune the model parameters to achieve the optimal balance between bias and variance for improved performance. Regularization and adjusting the polynomial degree are potential avenues for further exploration.