q1-svm

November 6, 2023

0.1 Assignment 3

0.2 Support Vector Machine

0.2.1 Name: Gautam Kumar

0.2.2 Roll Number: 21CS30020

[24]:
```
from ucimlrepo import fetch_ucirepo
import pandas as pd

spambase = fetch_ucirepo(id=94)
# data (as pandas dataframes)
X = spambase.data.features
y = spambase.data.targets
# 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}}

```
                          name     role        type demographic  \
0               word_freq_make  Feature  Continuous        None   
1            word_freq_address  Feature  Continuous        None   
2                word_freq_all  Feature  Continuous        None   
3                 word_freq_3d  Feature  Continuous        None   
4                word_freq_our  Feature  Continuous        None   
5               word_freq_over  Feature  Continuous        None   
6             word_freq_remove  Feature  Continuous        None   
7           word_freq_internet  Feature  Continuous        None   
8              word_freq_order  Feature  Continuous        None   
9               word_freq_mail  Feature  Continuous        None   
10           word_freq_receive  Feature  Continuous        None   
11              word_freq_will  Feature  Continuous        None   
12            word_freq_people  Feature  Continuous        None   
13            word_freq_report  Feature  Continuous        None   
```

```
14          word_freq_addresses  Feature  Continuous        None
15               word_freq_free  Feature  Continuous        None
16           word_freq_business  Feature  Continuous        None
17              word_freq_email  Feature  Continuous        None
18                word_freq_you  Feature  Continuous        None
19             word_freq_credit  Feature  Continuous        None
20               word_freq_your  Feature  Continuous        None
21               word_freq_font  Feature  Continuous        None
22                word_freq_000  Feature  Continuous        None
23              word_freq_money  Feature  Continuous        None
24                 word_freq_hp  Feature  Continuous        None
25                word_freq_hpl  Feature  Continuous        None
26             word_freq_george  Feature  Continuous        None
27                word_freq_650  Feature  Continuous        None
28                word_freq_lab  Feature  Continuous        None
29               word_freq_labs  Feature  Continuous        None
30             word_freq_telnet  Feature  Continuous        None
31                word_freq_857  Feature  Continuous        None
32               word_freq_data  Feature  Continuous        None
33                word_freq_415  Feature  Continuous        None
34                 word_freq_85  Feature  Continuous        None
35         word_freq_technology  Feature  Continuous        None
36               word_freq_1999  Feature  Continuous        None
37              word_freq_parts  Feature  Continuous        None
38                 word_freq_pm  Feature  Continuous        None
39             word_freq_direct  Feature  Continuous        None
40                 word_freq_cs  Feature  Continuous        None
41            word_freq_meeting  Feature  Continuous        None
42           word_freq_original  Feature  Continuous        None
43            word_freq_project  Feature  Continuous        None
44                 word_freq_re  Feature  Continuous        None
45                word_freq_edu  Feature  Continuous        None
46              word_freq_table  Feature  Continuous        None
47         word_freq_conference  Feature  Continuous        None
48                  char_freq_;  Feature  Continuous        None
49                  char_freq_(  Feature  Continuous        None
50                  char_freq_[  Feature  Continuous        None
51                  char_freq_!  Feature  Continuous        None
52                  char_freq_$  Feature  Continuous        None
53                  char_freq_#  Feature  Continuous        None
54   capital_run_length_average  Feature  Continuous        None
55   capital_run_length_longest  Feature  Continuous        None
56     capital_run_length_total  Feature  Continuous        None
57                        Class   Target      Binary        None

               description units missing_values
0                     None  None             no
1                     None  None             no
```

2	None	None	no
3	None	None	no
4	None	None	no
5	None	None	no
6	None	None	no
7	None	None	no
8	None	None	no
9	None	None	no
10	None	None	no
11	None	None	no
12	None	None	no
13	None	None	no
14	None	None	no
15	None	None	no
16	None	None	no
17	None	None	no
18	None	None	no
19	None	None	no
20	None	None	no
21	None	None	no
22	None	None	no
23	None	None	no
24	None	None	no
25	None	None	no
26	None	None	no
27	None	None	no
28	None	None	no
29	None	None	no
30	None	None	no
31	None	None	no
32	None	None	no
33	None	None	no
34	None	None	no
35	None	None	no
36	None	None	no
37	None	None	no
38	None	None	no
39	None	None	no
40	None	None	no
41	None	None	no
42	None	None	no
43	None	None	no
44	None	None	no
45	None	None	no
46	None	None	no
47	None	None	no
48	None	None	no
49	None	None	no

```
50                          None  None             no
51                          None  None             no
52                          None  None             no
53                          None  None             no
54                          None  None             no
55                          None  None             no
56                          None  None             no
57  spam (1) or not spam (0)  None             no
```

[25]:
```
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
```

[26]:
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

[27]:
```
print(len(X))
```

```
4601
```

[28]:
```
y = y.iloc[:, 0].ravel()
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

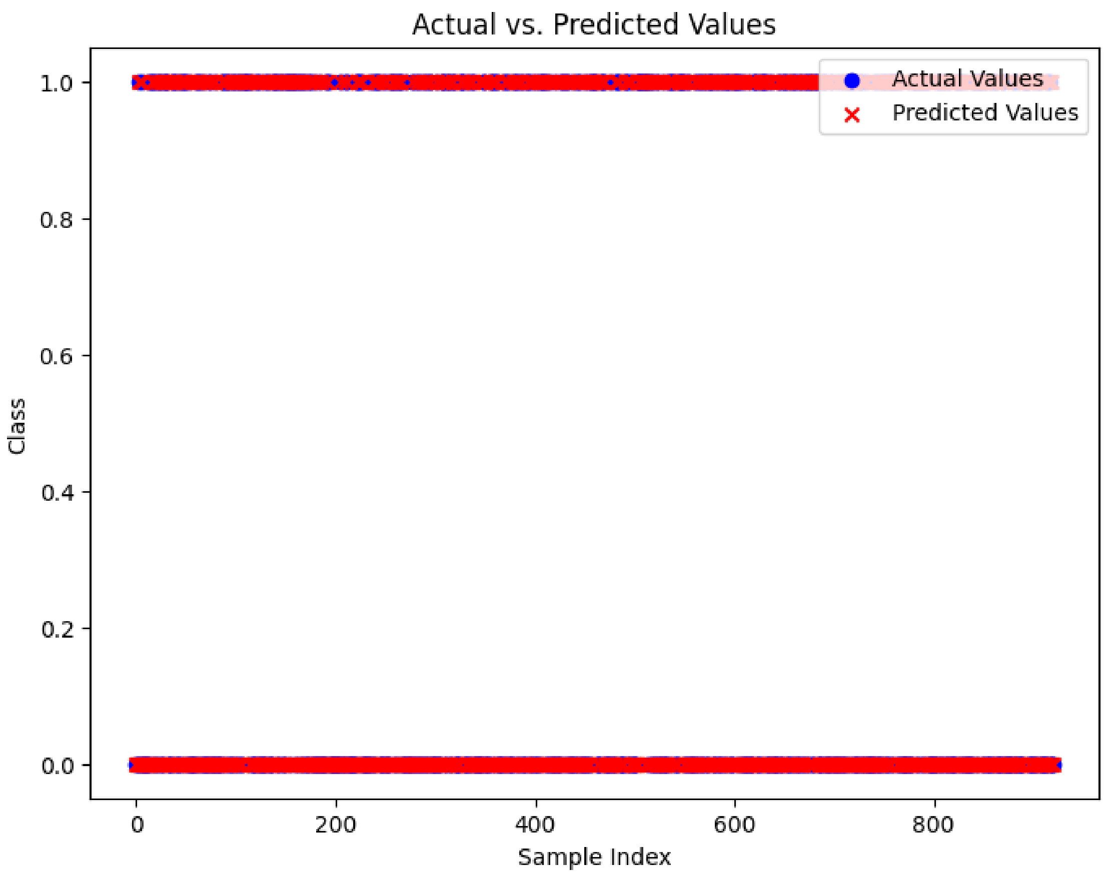
[29]:
```
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
```

[29]:
```
SVC(kernel='linear')
```

[30]:
```
y_pred = svm_model.predict(X_test)
```

[31]:
```
##Acuracy without applying Regularisation
accuracy = accuracy_score(y_test, y_pred)
```

[32]:
```
plt.figure(figsize=(8, 6))
plt.scatter(range(len(y_test)), y_test, color='blue', label='Actual Values',␣
 ↪marker='o')
plt.scatter(range(len(y_test)), y_pred, color='red', label='Predicted Values',␣
 ↪marker='x')
plt.title('Actual vs. Predicted Values')
plt.xlabel('Sample Index')
plt.ylabel('Class')
plt.legend(loc='upper right')
plt.show()
```



[33]:
```
C_values = [0.001, 0.1, 1, 10, 100]
accuracy_scores = []
y_preds = []
```

[34]:
```
for C in C_values:
    svm_model = SVC(kernel='linear', C=C)
    svm_model.fit(X_train, y_train)
    y_pred = svm_model.predict(X_test)
    y_preds.append(y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)
```

[35]:
```
accuracy_table = pd.DataFrame({'C': C_values, 'Accuracy': accuracy_scores})
```

[36]:
```
accuracy_table
```

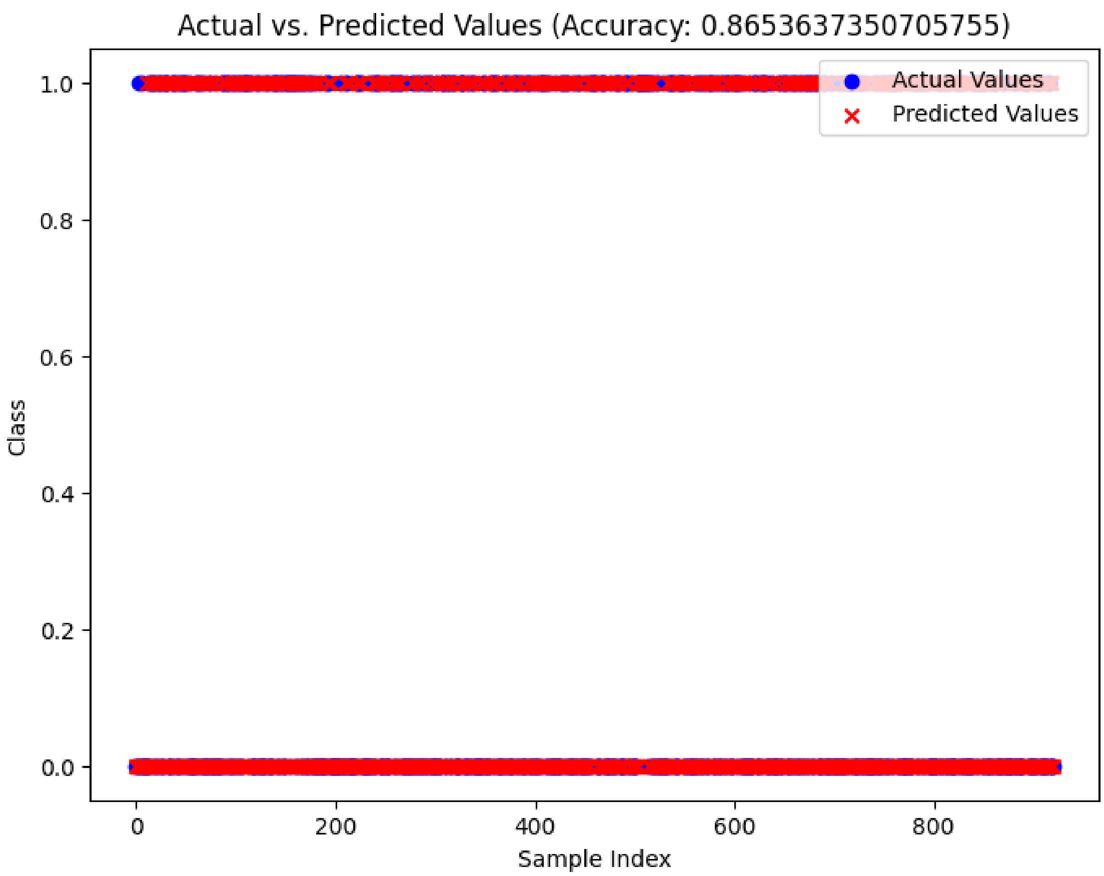
[36]:

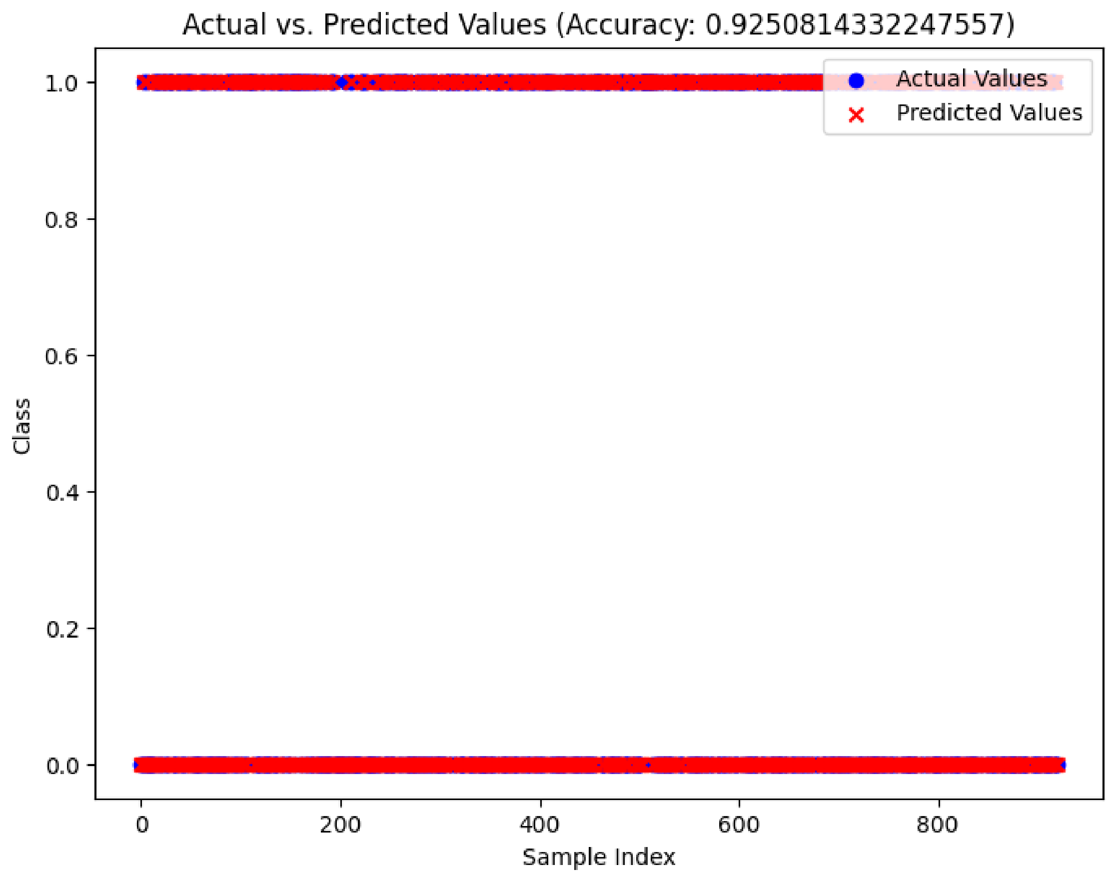
	C	Accuracy
0	0.001	0.865364
1	0.100	0.925081

```
2     1.000   0.922910
3    10.000   0.918567
4   100.000   0.913138
```

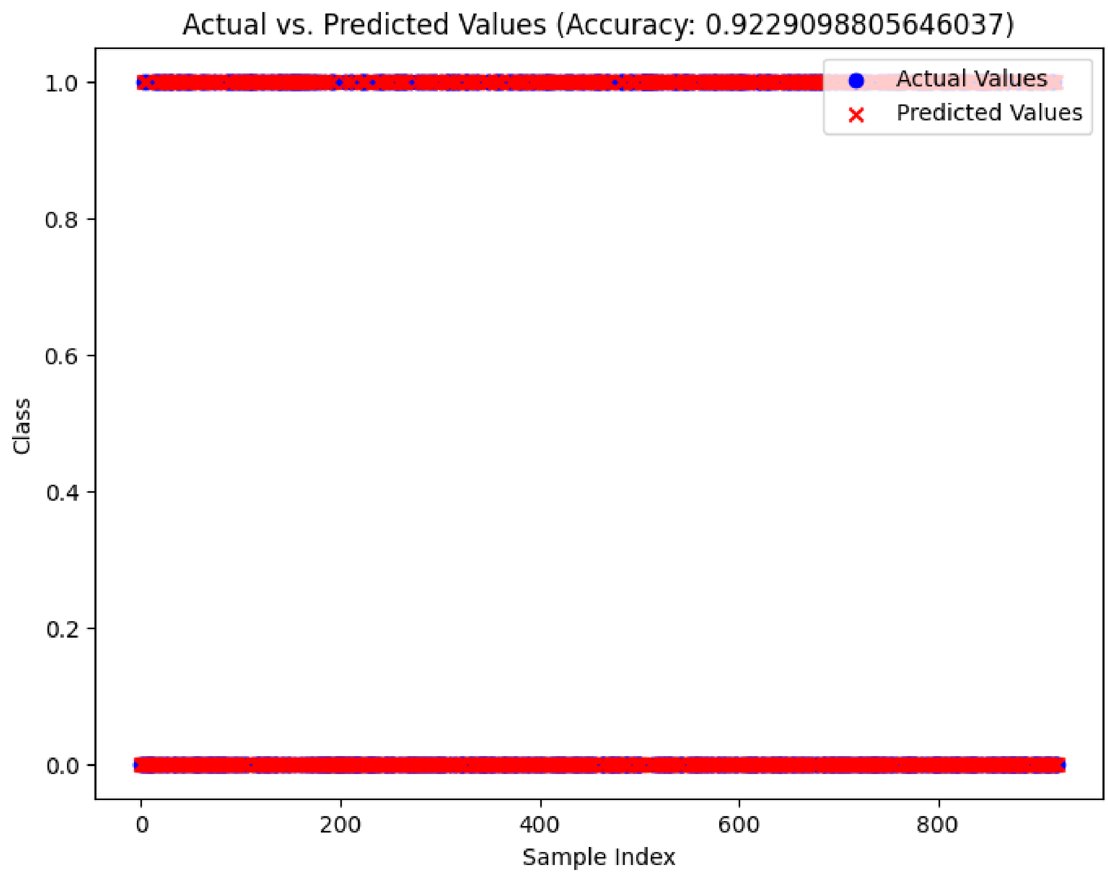
[37]:

```
i = 0
for y_pred in y_preds:
    plt.figure(figsize=(8, 6))
    plt.scatter(range(len(y_test)), y_test, color='blue', label='Actual␣
↪Values', marker='o')
    plt.scatter(range(len(y_test)), y_pred, color='red', label='Predicted␣
↪Values', marker='x')
    plt.title(f'Actual vs. Predicted Values (Accuracy: {accuracy_scores[i]})')
    plt.xlabel('Sample Index')
    plt.ylabel('Class')
    plt.legend(loc='upper right')
    plt.show()
    i += 1
```

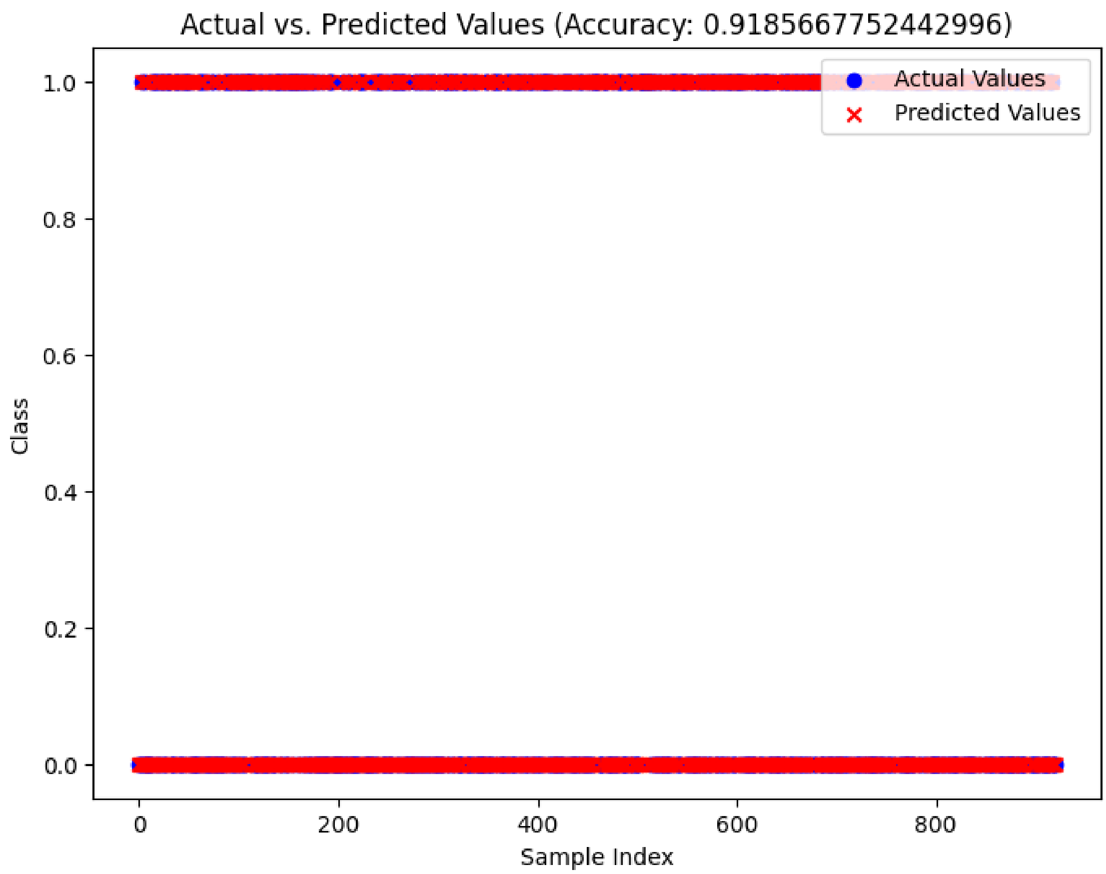


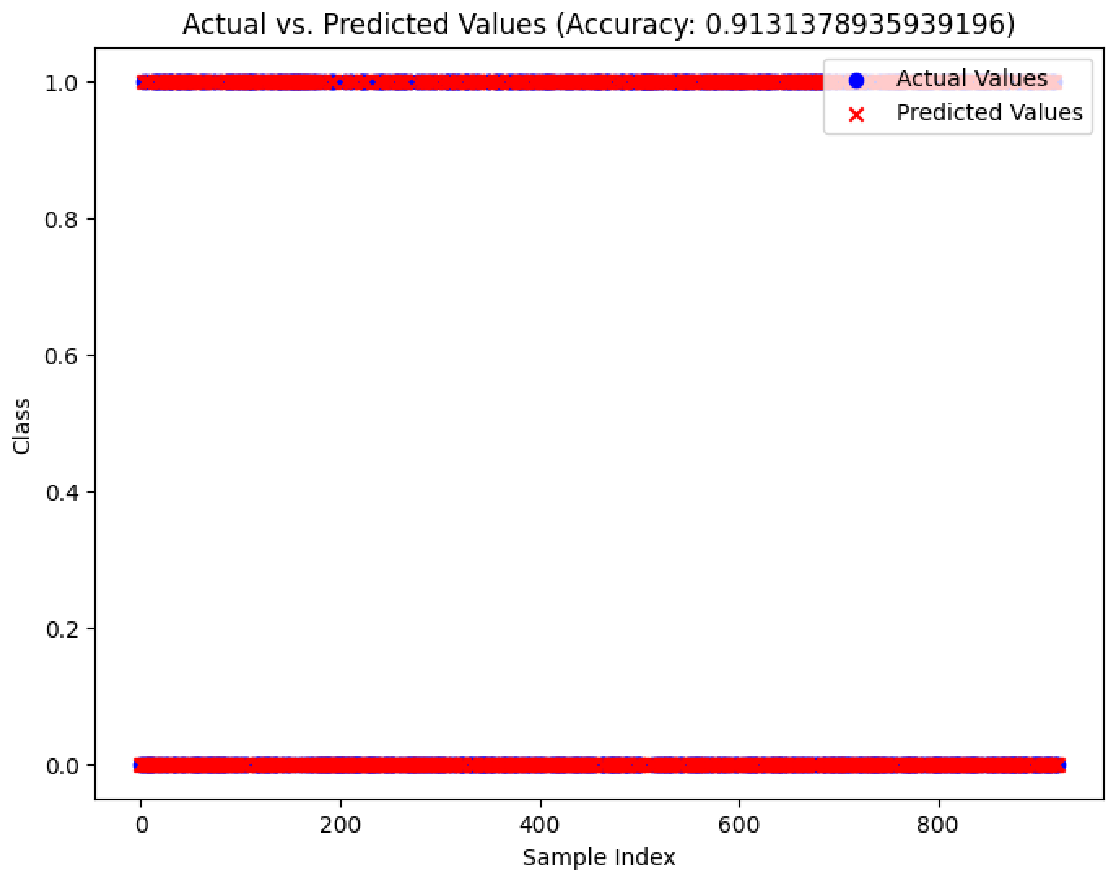


Actual vs. Predicted Values (Accuracy: 0.9250814332247557)
Actual Values
Predicted Values
1.0
0.8
0.6
0.4
0.2
0.0
Class
0
200
400
600
800
Sample Index



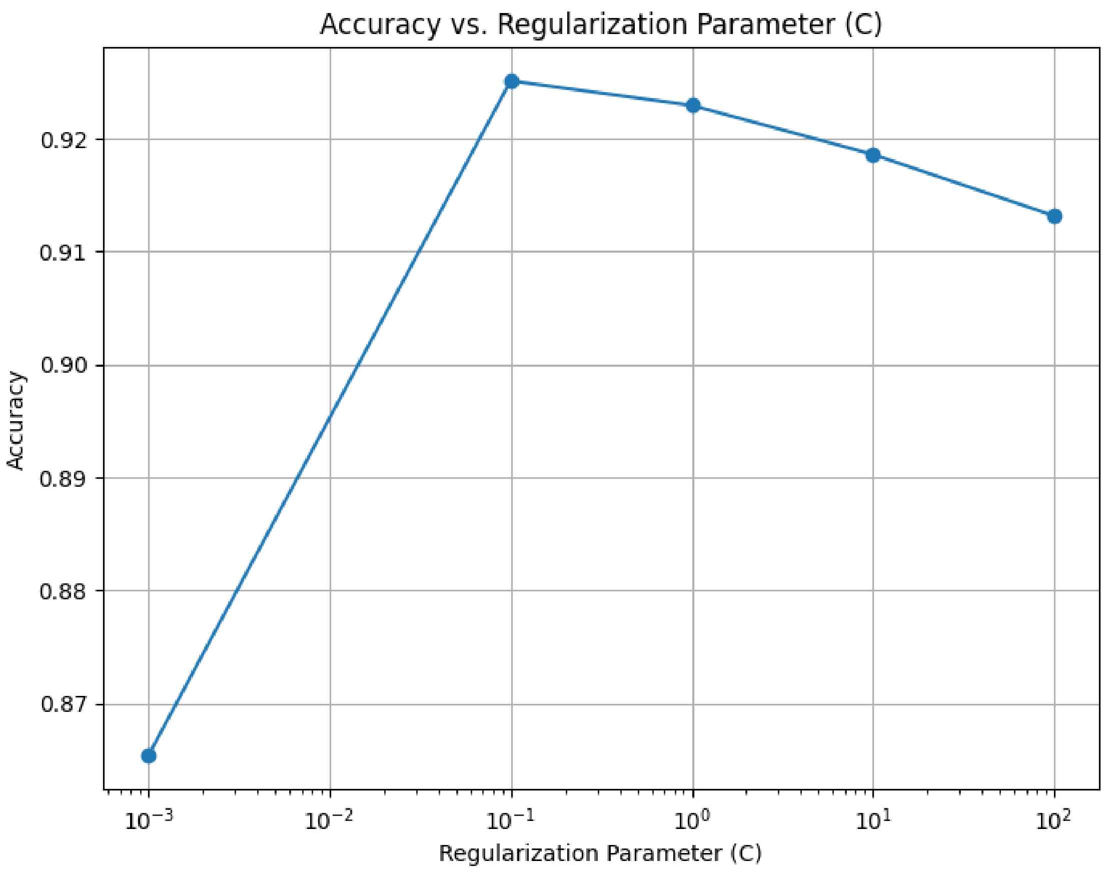
Actual vs. Predicted Values (Accuracy: 0.9229098805646037)
Actual Values
Predicted Values
1.0
0.8
0.6
0.4
0.2
0.0
Class
0
200
400
600
800
Sample Index


Actual vs. Predicted Values (Accuracy: 0.9185667752442996)
Actual Values
Predicted Values
1.0
0.8
0.6
0.4
0.2
0.0
Class
0
200
400
600
800
Sample Index



[38]:

```
plt.figure(figsize=(8, 6))
plt.plot(C_values, accuracy_scores, marker='o')
plt.title('Accuracy vs. Regularization Parameter (C)')
plt.xlabel('Regularization Parameter (C)')
plt.xscale('log')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



[39]:
```
# Define a list of kernels to test
kernels = ['poly', 'poly', 'sigmoid', 'rbf']
degrees = [2, 3, 0, 0]  # Specify the degrees for the polynomial kernels
```

[40]:
```
results = []
for kernel, degree in zip(kernels, degrees):
    if kernel == 'poly':
        # Polynomial kernel
        svm_model = SVC(kernel=kernel, degree=degree)
    else:
        # Sigmoid or RBF kernel
        svm_model = SVC(kernel=kernel)

    svm_model.fit(X_train, y_train)
    y_pred = svm_model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted',␣
↪zero_division=0)
```

```python
    recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

    results.append({
        'Kernel': kernel,
        'Degree': degree if kernel == 'poly' else None,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })
results_df = pd.DataFrame(results)
```

[41]:

```python
results_df
```

[41]:

	Kernel	Degree	Accuracy	Precision	Recall	F1 Score
0	poly	2.0	0.649294	0.731458	0.649294	0.577828
1	poly	3.0	0.625407	0.718539	0.625407	0.533076
2	sigmoid	NaN	0.635179	0.632524	0.635179	0.633420
3	rbf	NaN	0.662324	0.662088	0.662324	0.644053

[42]:

```python
degrees = [1, 1, 3, 3]
C_values = [0.01, 100, 0.01, 100]
train_accuracies = []
test_accuracies = []
```

[43]:

```python
for degree, C in zip(degrees, C_values):
    svm_model = SVC(kernel='poly', degree=degree, C=C)
    svm_model.fit(X_train, y_train)

    # Training accuracy
    y_train_pred = svm_model.predict(X_train)
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_accuracies.append(train_accuracy)

    # Test accuracy
    y_test_pred = svm_model.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_accuracies.append(test_accuracy)
```

[44]:

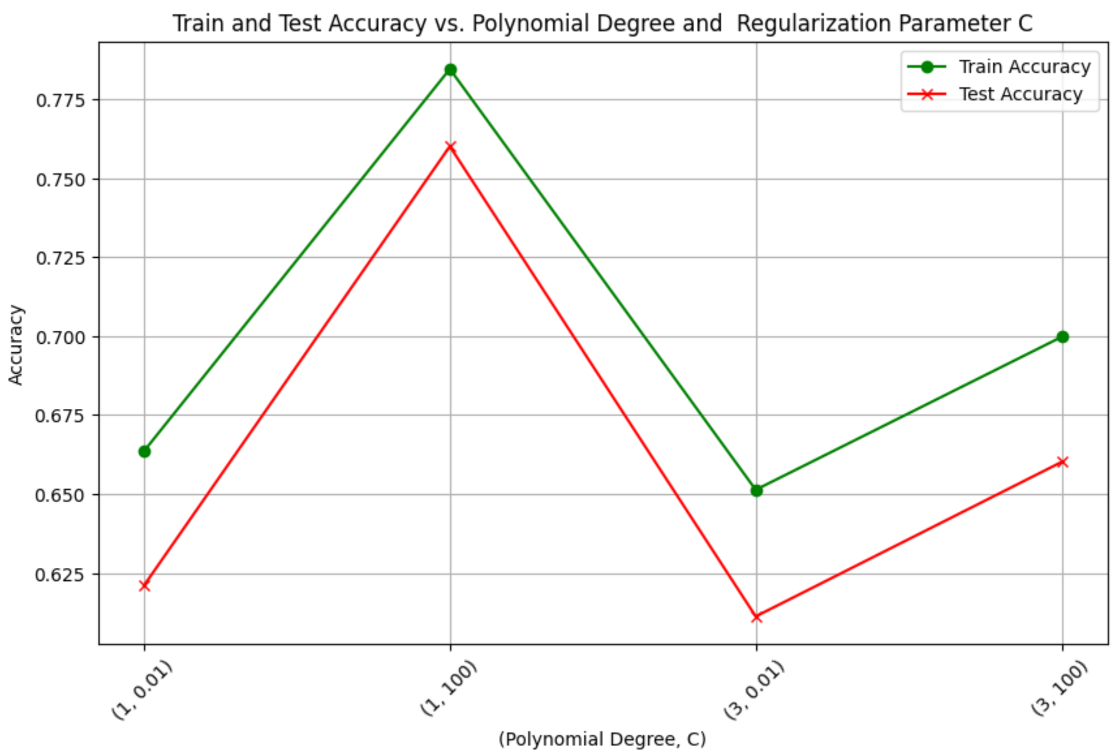
```python
experiment_results = pd.DataFrame({
    'Polynomial Degree': degrees,
    'Regularization Parameter (C)': C_values,
    'Train Accuracy': train_accuracies,
    'Test Accuracy': test_accuracies
})
```

[45]:
```
experiment_results
```

[45]:
```
   Polynomial Degree  Regularization Parameter (C)  Train Accuracy  \
0                  1                          0.01        0.663587
1                  1                        100.00        0.784511
2                  3                          0.01        0.651359
3                  3                        100.00        0.699728

   Test Accuracy
0       0.621064
1       0.760043
2       0.611292
3       0.660152
```

[46]:
```
plt.figure(figsize=(10, 6))
plt.plot(range(len(experiment_results)), train_accuracies,color='green',␣
  ↪marker='o', label='Train Accuracy')
plt.plot(range(len(experiment_results)), test_accuracies,color='red',␣
  ↪marker='x', label='Test Accuracy')
plt.xticks(range(len(experiment_results)), [f'({degree}, {C})' for degree, C in␣
  ↪zip(degrees, C_values)], rotation=45)
plt.xlabel('(Polynomial Degree, C)')
plt.ylabel('Accuracy')
plt.title('Train and Test Accuracy vs. Polynomial Degree and  Regularization␣
  ↪Parameter C')
plt.legend()
plt.grid(True)
plt.show()
```


Train and Test Accuracy vs. Polynomial Degree and Regularization Parameter C
Train Accuracy
Test Accuracy
Accuracy
0.775
0.750
0.725
0.700
0.675
0.650
0.625
(1, 0.01)
(1, 100)
(3, 0.01)
(3, 100)
(Polynomial Degree, C)

[ ]:

[ ]: