In [59]:
```
# Importing all the necessary libraries and check their versions
import sys
import numpy
import sklearn
import pandas
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier

# Print versions of the imported libraries
print('Python: {}'.format(sys.version))
print('Numpy: {}'.format(numpy.__version__))
print('Sklearn: {}'.format(sklearn.__version__))
print('Pandas: {}'.format(pandas.__version__))
```

```
Python: 3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 15:03:56) [MSC v.1
929 64 bit (AMD64)]
Numpy: 1.26.4
Sklearn: 1.4.2
Pandas: 2.2.2
```

In [60]:
```
# Import, change module names
import numpy as np
import pandas as pd
# Import the UCI Molecular Biology (Promoter Gene Sequences) Data Set
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/molecular-biology/
names = ['Class', 'id', 'Sequence']
data = pd.read_csv(url, names=names)
```

In [61]:
```
# Display a sample from the dataset
print(data.iloc[0])
```

```
Class                                                     +
id                                                      S10
Sequence    \t\ttactagcaatacgcttgcgttcggtggttaagtatgtataat...
Name: 0, dtype: object
```

In [62]:
```
# Display another sample
print(data.iloc[5])
```

```
Class                                                     +
id                                                   MALEFG
Sequence    \taggggcaaggaggatggaaagaggttgccgtataaagaaactag...
Name: 5, dtype: object
```

In [63]:
```
# Building our dataset by creating a custom Pandas DataFrame
# Each column in a DataFrame is called a Series
classes = data.loc[:, 'Class']
print(classes[:5])
```

```
0    +
1    +
2    +
3    +
4    +
Name: Class, dtype: object
```

In [64]:
```
# Generate a list of DNA sequences
sequences = list(data.loc[:, 'Sequence'])
dataset = {}

# Loop through sequences and split into individual nucleotides
for i, seq in enumerate(sequences):
    nucleotides = list(seq)
    nucleotides = [x for x in nucleotides if x != '\t']
    nucleotides.append(classes[i])
    dataset[i] = nucleotides

# Display the first dataset entry
print(dataset[0])
```

```
['t', 'a', 'c', 't', 'a', 'g', 'c', 'a', 'a', 't', 'a', 'c', 'g', 'c', 't', 't',
'g', 'c', 'g', 't', 't', 'c', 'g', 'g', 't', 'g', 'g', 't', 't', 'a', 'a', 'g', 't',
'a', 't', 'g', 't', 'a', 't', 'a', 'a', 't', 'g', 'c', 'g', 'c', 'g', 'g', 'g', 'c',
't', 't', 'g', 't', 'c', 'g', 't', '+']
```

In [65]:
```
# Turn dataset into pandas DataFrame
dframe = pd.DataFrame(dataset)
print(dframe)
```

	0	1	2	3	4	5	6	7	8	9	...	96	97	98	99	100	101	102	\
0	t	t	g	a	t	a	c	t	c	t	...	c	c	t	a	g	c	g	
1	a	g	t	a	c	g	a	t	g	t	...	c	g	a	g	a	c	t	
2	c	c	a	t	g	g	g	t	a	t	...	g	c	t	a	g	t	a	
3	t	t	c	t	a	g	g	c	c	t	...	a	t	g	g	a	c	t	
4	a	a	t	g	t	g	g	t	t	a	...	g	a	a	g	g	a	t	
5	g	t	a	t	a	c	g	a	t	a	...	t	g	c	g	c	a	c	
6	c	c	g	g	a	a	g	c	a	a	...	a	g	c	t	a	t	t	
7	a	c	a	a	t	a	t	a	a	t	...	g	a	g	g	t	g	c	
8	a	t	g	t	t	g	g	a	t	t	...	a	c	a	t	g	g	a	
9	t	g	a	g	a	g	g	a	a	t	...	c	t	a	a	t	c	a	
10	a	a	a	t	a	a	a	a	t	c	...	c	t	c	c	c	c	c	
11	c	c	c	g	c	g	g	c	a	c	...	c	t	g	t	a	t	a	
12	g	a	t	t	t	g	g	a	c	t	...	t	c	a	c	g	c	a	
13	c	g	a	a	a	a	a	c	t	c	...	t	t	g	c	c	t	g	
14	t	t	g	t	t	t	t	t	g	t	...	a	t	t	a	c	a	a	
15	t	t	t	c	t	g	t	t	c	t	...	g	g	c	a	t	a	t	
16	g	g	g	g	g	g	t	g	g	g	...	a	t	a	g	c	a	t	
17	c	t	c	a	a	a	a	a	a	t	...	g	t	a	a	g	c	a	
18	g	c	a	a	c	a	a	t	c	c	...	a	g	t	a	a	g	a	
19	t	a	t	g	g	a	g	a	a	a	...	g	a	c	g	c	g	c	
20	t	c	t	t	a	g	c	c	g	g	...	c	t	a	a	a	g	c	
21	c	g	a	g	a	a	c	t	g	g	...	a	t	g	g	a	t	g	
22	g	c	g	t	a	g	a	g	a	c	...	t	t	a	g	c	c	a	
23	g	t	c	g	a	g	t	t	c	c	...	g	t	c	a	t	t	c	
24	t	g	t	t	g	t	c	a	g	g	...	t	c	c	a	t	t	a	
25	g	a	t	t	c	t	t	t	t	g	...	c	c	g	g	g	g	g	
26	g	t	a	g	t	g	c	g	c	a	...	a	a	c	a	c	a	a	
27	t	t	t	c	g	c	c	a	c	a	...	g	t	t	t	a	g	t	
28	t	g	t	g	a	c	t	g	g	t	...	c	g	t	g	t	g	t	
29	a	g	t	g	a	g	g	c	t	a	...	c	c	t	a	a	g	c	
30	a	t	t	a	a	t	a	a	t	a	...	t	g	g	g	a	g	a	
31	g	g	t	g	a	a	t	t	c	c	...	c	g	a	g	a	t	a	
32	t	t	t	t	c	t	g	a	t	t	...	g	t	c	c	t	t	t	
33	a	c	t	a	c	a	a	c	g	c	...	a	g	t	t	g	t	c	
34	t	g	g	g	a	a	c	a	t	c	...	c	t	c	a	c	t	t	
35	g	t	t	a	c	a	g	g	g	c	...	a	t	t	g	t	t	c	
36	t	t	t	t	t	g	c	t	t	t	...	a	t	g	a	t	t	g	
37	a	a	a	g	a	a	a	a	a	a	...	c	t	g	c	t	g	t	
38	t	c	t	t	g	a	t	t	a	t	...	t	g	t	g	c	c	g	
39	a	a	c	t	a	a	a	a	a	a	...	t	c	a	t	t	t	g	
40	a	a	a	a	a	c	g	a	t	a	...	g	g	t	c	t	g	a	
41	t	t	t	g	t	t	t	t	c	t	...	c	c	t	t	g	a	t	
42	g	c	g	a	g	a	c	t	g	g	...	a	a	a	c	t	a	g	
43	c	t	c	a	c	g	a	g	c	c	...	t	a	c	t	a	a	g	
44	g	a	t	t	g	a	g	c	a	g	...	a	t	t	g	g	g	a	
45	c	a	a	a	c	g	c	t	a	c	...	a	g	g	c	a	g	c	
46	g	c	a	c	c	t	c	t	t	c	...	a	t	t	a	c	a	g	
47	g	g	c	t	t	c	c	c	g	a	...	t	t	g	t	g	g	t	
48	g	c	c	a	c	c	a	a	a	c	...	g	a	a	g	t	g	t	
49	c	a	a	a	c	g	t	a	a	c	...	c	a	a	g	g	a	c	
50	t	t	c	c	g	t	c	c	a	a	...	t	t	c	a	c	a	a	
51	t	c	c	a	t	t	a	a	t	c	...	t	c	a	g	c	c	a	
52	g	g	c	a	g	t	t	g	g	t	...	t	g	t	t	c	t	c	
53	t	c	g	a	g	a	g	a	g	g	...	c	c	t	a	t	a	a	
54	c	c	g	c	t	g	a	a	t	a	...	t	t	a	t	a	t	t	

											...							
55	g	a	c	t	a	g	a	c	t	c	...	t	t	t	g	c	a	t
56	t	a	g	c	g	t	t	a	t	a	...	g	t	t	a	g	t	g
57	+	+	+	+	+	+	+	+	+	+	...	-	-	-	-	-	-	-

	103	104	105
0	c	c	t
1	g	t	a
2	c	c	a
3	g	g	c
4	a	t	a
5	c	c	t
6	t	c	t
7	a	t	a
8	c	c	a
9	g	a	t
10	a	a	a
11	t	t	a
12	g	g	a
13	a	g	t
14	g	c	a
15	a	c	a
16	t	t	g
17	g	c	g
18	c	t	a
19	c	a	g
20	t	a	g
21	g	a	c
22	a	c	t
23	g	g	c
24	t	g	t
25	g	g	a
26	c	t	a
27	t	c	t
28	t	t	g
29	c	t	g
30	c	g	c
31	g	a	a
32	t	g	c
33	t	g	t
34	a	g	c
35	c	g	a
36	t	t	t
37	g	t	t
38	g	t	a
39	a	t	g
40	t	t	c
41	t	t	c
42	g	g	a
43	t	c	a
44	c	t	t
45	a	g	c
46	c	a	a
47	c	a	a
48	a	a	t
49	a	g	c
50	g	g	a

```
51   g   a   a
52   c   g   g
53   t   g   a
54   t   a   a
55   c   a   c
56   c   c   t
57   -   -   -

[58 rows x 106 columns]
```

In [66]:
```
# Transpose the DataFrame
df = dframe.transpose()
print(df.iloc[:5])
```

```
  0  1  2  3  4  5  6  7  8  9   ... 48 49 50 51 52 53 54 55 56 57
0  t  a  c  t  a  g  c  a  a  t  ...  g  c  t  t  g  t  c  g  t  +
1  t  g  c  t  a  t  c  c  t  g  ...  c  a  t  c  g  c  c  a  a  +
2  g  t  a  c  t  a  g  a  g  a  ...  c  a  c  c  c  g  g  c  g  +
3  a  a  t  t  g  t  g  a  t  g  ...  a  a  c  a  a  a  c  t  c  +
4  t  c  g  a  t  a  a  t  t  a  ...  c  c  g  t  g  g  t  a  g  +

[5 rows x 58 columns]
```

In [67]:
```
# For clarity, rename the last DataFrame column to 'Class'
df.rename(columns={57: 'Class'}, inplace=True)
print(df.iloc[:5])
```

```
  0  1  2  3  4  5  6  7  8  9  ... 48 49 50 51 52 53 54 55 56 Class
0  t  a  c  t  a  g  c  a  a  t  ...  g  c  t  t  g  t  c  g  t     +
1  t  g  c  t  a  t  c  c  t  g  ...  c  a  t  c  g  c  c  a  a     +
2  g  t  a  c  t  a  g  a  g  a  ...  c  a  c  c  c  g  g  c  g     +
3  a  a  t  t  g  t  g  a  t  g  ...  a  a  c  a  a  a  c  t  c     +
4  t  c  g  a  t  a  a  t  t  a  ...  c  c  g  t  g  g  t  a  g     +

[5 rows x 58 columns]
```

In [68]:
```
# Use describe() to summarize the dataset
df.describe()
```

Out[68]:

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52
count	106	106	106	106	106	106	106	106	106	106	...	106	106	106	106	106
unique	4	4	4	4	4	4	4	4	4	4	...	4	4	4	4	4
top	t	a	a	c	a	a	a	a	a	a	...	c	c	c	t	t
freq	38	34	30	30	36	42	38	34	33	36	...	36	42	31	33	35

4 rows × 58 columns

In [69]:
```
#Generating counts for each column
series = []
for name in df.columns:
    series.append(df[name].value_counts())
```

```
info = pd.DataFrame(series)
details = info.transpose()
print(details)
```

```
   count  count  count  count  count  count  count  count  count  count  ...  \
t   38.0   26.0   27.0   26.0   22.0   24.0   30.0   32.0   32.0   28.0  ...
c   27.0   22.0   21.0   30.0   19.0   18.0   21.0   20.0   22.0   22.0  ...
a   26.0   34.0   30.0   22.0   36.0   42.0   38.0   34.0   33.0   36.0  ...
g   15.0   24.0   28.0   28.0   29.0   22.0   17.0   20.0   19.0   20.0  ...
+    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN  ...
-    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN  ...

   count  count  count  count  count  count  count  count  count  count
t   21.0   22.0   23.0   33.0   35.0   30.0   23.0   29.0   34.0    NaN
c   36.0   42.0   31.0   32.0   21.0   32.0   29.0   29.0   17.0    NaN
a   23.0   24.0   28.0   27.0   25.0   22.0   26.0   24.0   27.0    NaN
g   26.0   18.0   24.0   14.0   25.0   22.0   28.0   24.0   28.0    NaN
+    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN   53.0
-    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN   53.0

[6 rows x 58 columns]
```

In [70]:
```
# Convert categorical sequence data to numerical using one-hot encoding
numerical_df = pd.get_dummies(df)
numerical_df.iloc[:5]
```

Out[70]:

	0_a	0_c	0_g	0_t	1_a	1_c	1_g	1_t	2_a	2_c	...	55_a	55_c	55_g	55
0	False	False	False	True	True	False	False	False	False	True	...	False	False	True	Fal
1	False	False	False	True	False	False	True	False	False	True	...	True	False	False	Fal
2	False	False	True	False	False	False	False	True	True	False	...	False	True	False	Fal
3	True	False	False	False	True	False	False	False	False	False	...	False	False	False	Tr
4	False	False	False	True	False	True	False	False	False	False	...	True	False	False	Fal

5 rows × 230 columns

In [71]:
```
#display 60 rows of the data
numerical_df.iloc[:60]
```

Out[71]:

	0_a	0_c	0_g	0_t	1_a	1_c	1_g	1_t	2_a	2_c	...	55_a	55_c	55_g	5
0	False	False	False	True	True	False	False	False	False	True	...	False	False	True	F
1	False	False	False	True	False	False	True	False	False	True	...	True	False	False	F
2	False	False	True	False	False	False	False	True	True	False	...	False	True	False	F
3	True	False	False	False	True	False	False	False	False	False	...	False	False	False	T
4	False	False	False	True	False	True	False	False	False	False	...	True	False	False	F
5	True	False	False	False	False	False	True	False	False	False	...	False	False	True	F
6	False	True	False	False	True	False	False	False	False	False	...	True	False	False	F
7	False	False	False	True	False	False	False	True	False	False	...	False	True	False	F
8	False	True	False	False	False	False	True	False	True	False	...	False	False	False	T
9	False	False	False	True	False	False	False	True	False	False	...	False	True	False	F
10	False	False	True	False	False	True	False	False	True	False	...	False	True	False	F
11	False	True	False	False	False	True	False	False	False	False	...	True	False	False	F
12	False	False	True	False	True	False	False	False	False	False	...	False	False	True	F
13	False	True	False	False	False	False	False	True	False	False	...	False	True	False	F
14	False	False	False	True	False	False	False	True	False	False	...	False	True	False	F
15	True	False	False	False	True	False	False	False	False	False	...	False	True	False	F
16	True	False	False	False	False	False	False	True	False	False	...	False	True	False	F
17	True	False	False	False	True	False	False	False	True	False	...	False	False	True	F
18	False	False	False	True	False	True	False	False	False	False	...	False	True	False	F
19	False	False	True	False	False	True	False	False	True	False	...	False	False	False	T
20	False	False	True	False	True	False	False	False	False	True	...	True	False	False	F
21	True	False	False	False	True	False	False	False	True	False	...	False	False	False	T
22	False	False	False	True	False	True	False	False	False	False	...	False	True	False	F
23	True	False	False	False	False	True	False	False	False	True	...	False	False	True	F
24	True	False	False	False	True	False	False	False	True	False	...	False	True	False	F
25	False	False	False	True	False	False	False	True	False	False	...	False	True	False	F
26	False	True	False	False	True	False	False	False	False	False	...	False	False	False	T
27	False	False	False	True	False	True	False	False	False	True	...	False	True	False	F
28	True	False	False	False	False	True	False	False	True	False	...	False	False	True	F
29	False	False	False	True	False	False	True	False	False	False	...	False	False	True	F

	0_a	0_c	0_g	0_t	1_a	1_c	1_g	1_t	2_a	2_c	...	55_a	55_c	55_g	5
30	False	True	False	False	False	False	False	True	False	False	...	False	False	True	F
31	True	False	False	False	False	False	False	True	False	False	...	False	False	True	F
32	True	False	False	False	False	False	False	True	False	False	...	False	False	False	T
33	False	False	False	True	True	False	False	False	True	False	...	False	False	True	F
34	True	False	False	False	False	False	False	True	False	False	...	True	False	False	F
35	False	True	False	False	False	True	False	False	False	False	...	False	False	True	F
36	False	False	False	True	False	True	False	False	False	False	...	False	False	False	T
37	False	True	False	False	False	True	False	False	False	False	...	True	False	False	F
38	False	False	False	True	False	False	False	True	False	True	...	False	False	False	T
39	False	False	False	True	False	False	True	False	False	False	...	False	False	False	T
40	False	False	True	False	True	False	False	False	False	False	...	False	False	False	T
41	True	False	False	False	True	False	False	False	False	True	...	False	False	False	T
42	False	False	False	True	False	False	False	True	True	False	...	False	True	False	F
43	False	False	True	False	False	True	False	False	False	True	...	True	False	False	F
44	False	True	False	False	True	False	False	False	False	False	...	False	False	True	F
45	False	True	False	False	True	False	False	False	False	True	...	True	False	False	F
46	True	False	False	False	False	False	False	True	True	False	...	False	False	True	F
47	False	True	False	False	True	False	False	False	True	False	...	True	False	False	F
48	False	False	True	False	False	False	True	False	False	True	...	False	True	False	F
49	False	False	False	True	True	False	False	False	False	False	...	False	False	False	T
50	False	True	False	False	False	True	False	False	True	False	...	True	False	False	F
51	False	False	False	True	False	False	True	False	False	False	...	True	False	False	F
52	False	False	False	True	False	True	False	False	True	False	...	True	False	False	F
53	True	False	False	False	False	False	False	True	True	False	...	False	False	False	T
54	False	True	False	False	False	False	True	False	True	False	...	False	False	False	T
55	False	True	False	False	True	False	False	False	True	False	...	False	True	False	F
56	False	False	False	True	False	False	False	True	False	False	...	True	False	False	F
57	False	True	False	False	False	False	True	False	False	False	...	True	False	False	F
58	False	False	True	False	False	True	False	False	False	True	...	False	False	False	T
59	False	False	False	True	False	False	True	False	False	False	...	True	False	False	F

60 rows × 230 columns

In [72]:
```
# Drop the redundant class column and rename the remaining one
df = numerical_df.drop(columns=['Class_-'])
df.rename(columns={'Class_+': 'Class'}, inplace=True)
print(df.iloc[:5])
```

```
    0_a    0_c    0_g    0_t    1_a    1_c    1_g    1_t    2_a    2_c  ...  \
0  False  False  False   True   True  False  False  False  False   True  ...
1  False  False  False   True  False  False   True  False  False   True  ...
2  False  False   True  False  False  False  False   True   True  False  ...
3   True  False  False  False   True  False  False  False  False  False  ...
4  False  False  False   True  False   True  False  False  False  False  ...

    54_t   55_a   55_c   55_g   55_t   56_a   56_c   56_g   56_t  Class
0  False  False  False   True  False  False  False  False   True   True
1  False   True  False  False  False   True  False  False  False   True
2  False  False   True  False  False  False  False   True  False   True
3  False  False  False  False   True  False   True  False  False   True
4   True   True  False  False  False  False  False   True  False   True

[5 rows x 229 columns]
```

In [73]:
```
# since data is already preprocessed  in 'df'
X = df.drop(columns=['Class'])
y = df['Class']
```

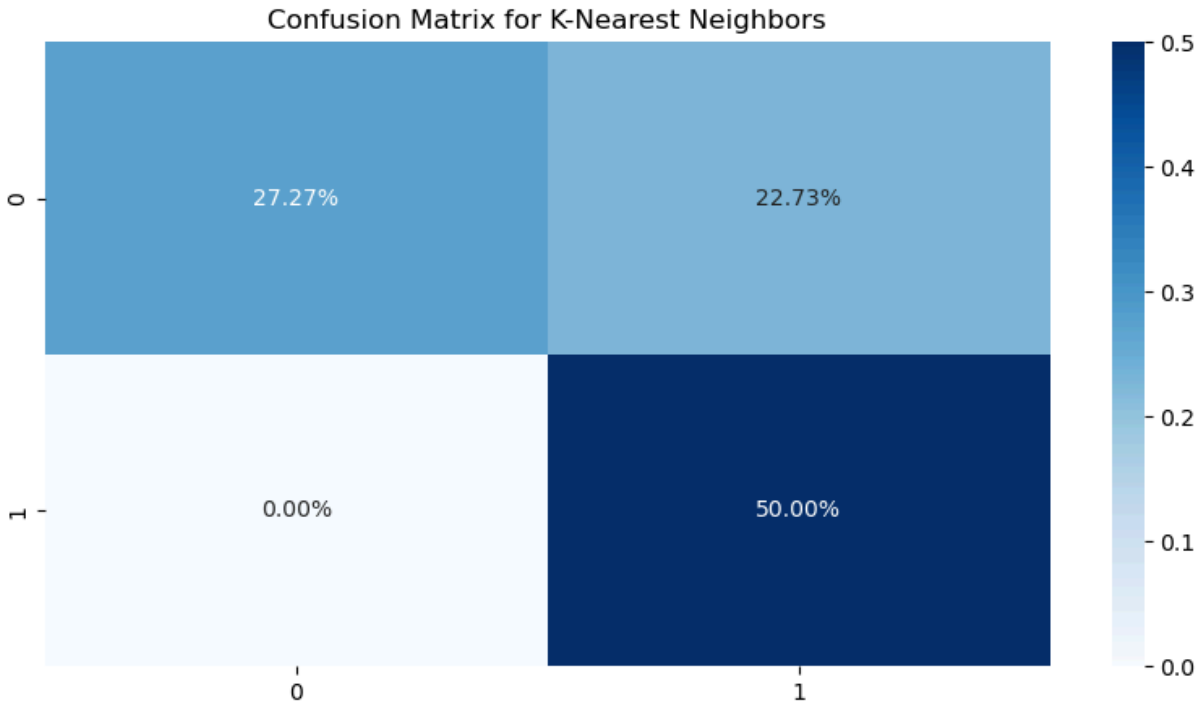
In [74]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
X.shape
```

Out[74]: (106, 228)

In [75]:
```
# 1. K-Nearest Neighbors (KNN)
knn_model = KNeighborsClassifier()

# Fit the model
knn_model.fit(X_train, y_train)

# Make predictions
knn_predictions = knn_model.predict(X_test)

# Print classification report
print("\nClassification Report for K-Nearest Neighbors:")
print(classification_report(y_test, knn_predictions))

# Print accuracy score
print(f"Accuracy Score for K-Nearest Neighbors: {accuracy_score(y_test, knn_predict

# Create and display confusion matrix
knn_cm = confusion_matrix(y_test, knn_predictions)
plt.figure(figsize=(10, 5))
sns.heatmap(knn_cm / np.sum(knn_cm), annot=True, fmt='.2%', cmap='Blues')
```

```
plt.title("Confusion Matrix for K-Nearest Neighbors")
plt.show()
```

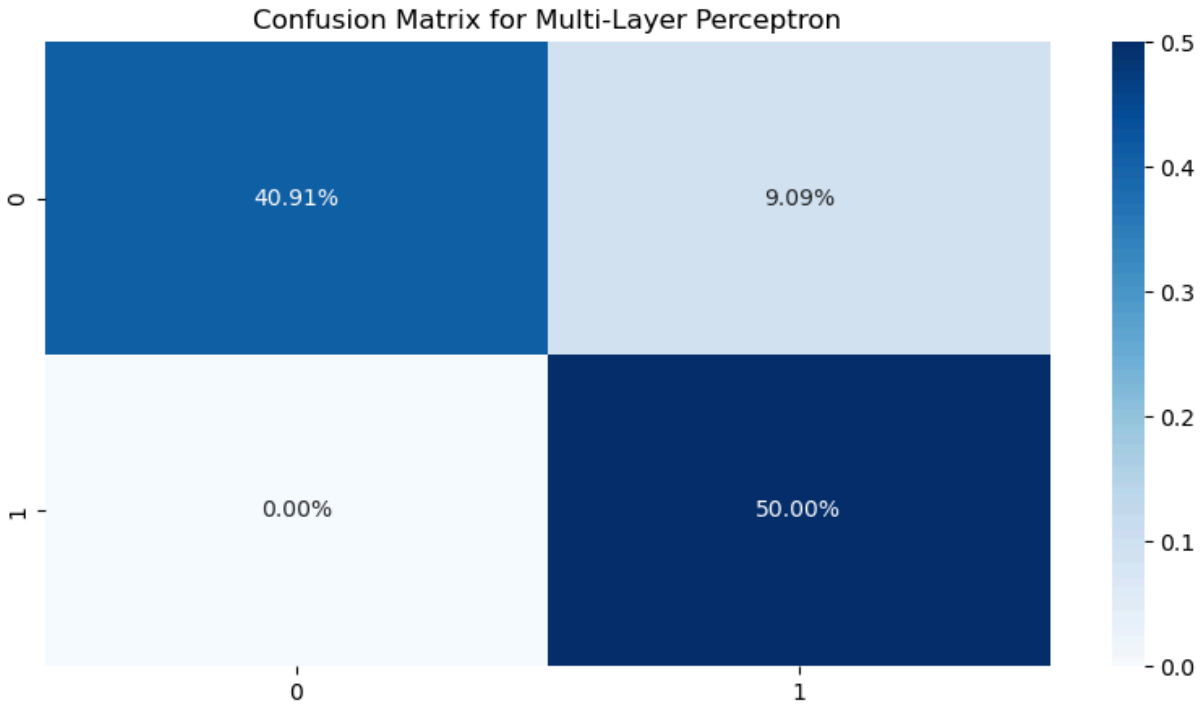
```
Classification Report for K-Nearest Neighbors:
              precision    recall  f1-score   support

       False       1.00      0.55      0.71        11
        True       0.69      1.00      0.81        11

    accuracy                           0.77        22
   macro avg       0.84      0.77      0.76        22
weighted avg       0.84      0.77      0.76        22

Accuracy Score for K-Nearest Neighbors: 0.77
```



In [76]:
```
# 2. Multi-Layer Perceptron (MLP)
mlp_model = MLPClassifier(max_iter=500, random_state=42)

# Fit the model
mlp_model.fit(X_train, y_train)

# Make predictions
mlp_predictions = mlp_model.predict(X_test)

# Print classification report
print("\nClassification Report for Multi-Layer Perceptron:")
print(classification_report(y_test, mlp_predictions))

# Print accuracy score
print(f"Accuracy Score for Multi-Layer Perceptron: {accuracy_score(y_test, mlp_pred

# Create and display confusion matrix
mlp_cm = confusion_matrix(y_test, mlp_predictions)
plt.figure(figsize=(10, 5))
```

```
sns.heatmap(mlp_cm / np.sum(mlp_cm), annot=True, fmt='.2%', cmap='Blues')
plt.title("Confusion Matrix for Multi-Layer Perceptron")
plt.show()
```

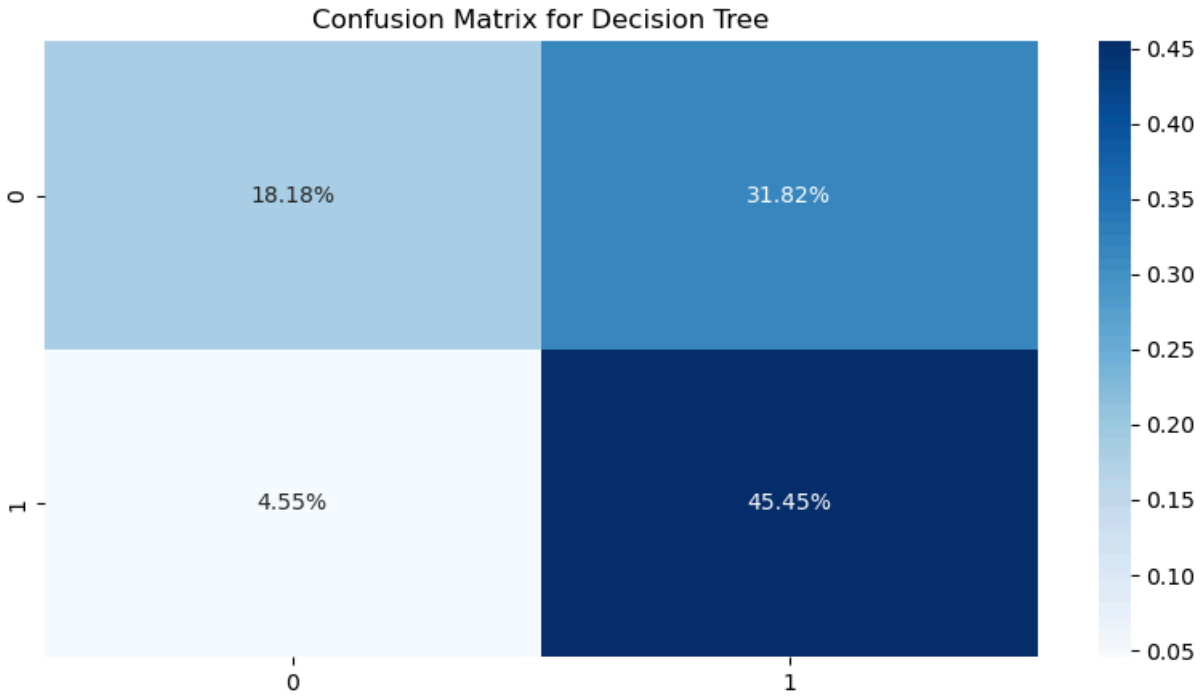
```
Classification Report for Multi-Layer Perceptron:
              precision    recall  f1-score   support

       False       1.00      0.82      0.90        11
        True       0.85      1.00      0.92        11

    accuracy                           0.91        22
   macro avg       0.92      0.91      0.91        22
weighted avg       0.92      0.91      0.91        22

Accuracy Score for Multi-Layer Perceptron: 0.91
```



In [77]:
```
# 3. Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)

# Fit the model
dt_model.fit(X_train, y_train)

# Make predictions
dt_predictions = dt_model.predict(X_test)

# Print classification report
print("\nClassification Report for Decision Tree:")
print(classification_report(y_test, dt_predictions))

# Print accuracy score
print(f"Accuracy Score for Decision Tree: {accuracy_score(y_test, dt_predictions):.

# Create and display confusion matrix
dt_cm = confusion_matrix(y_test, dt_predictions)
```

```
plt.figure(figsize=(10, 5))
sns.heatmap(dt_cm / np.sum(dt_cm), annot=True, fmt='.2%', cmap='Blues')
plt.title("Confusion Matrix for Decision Tree")
plt.show()
```

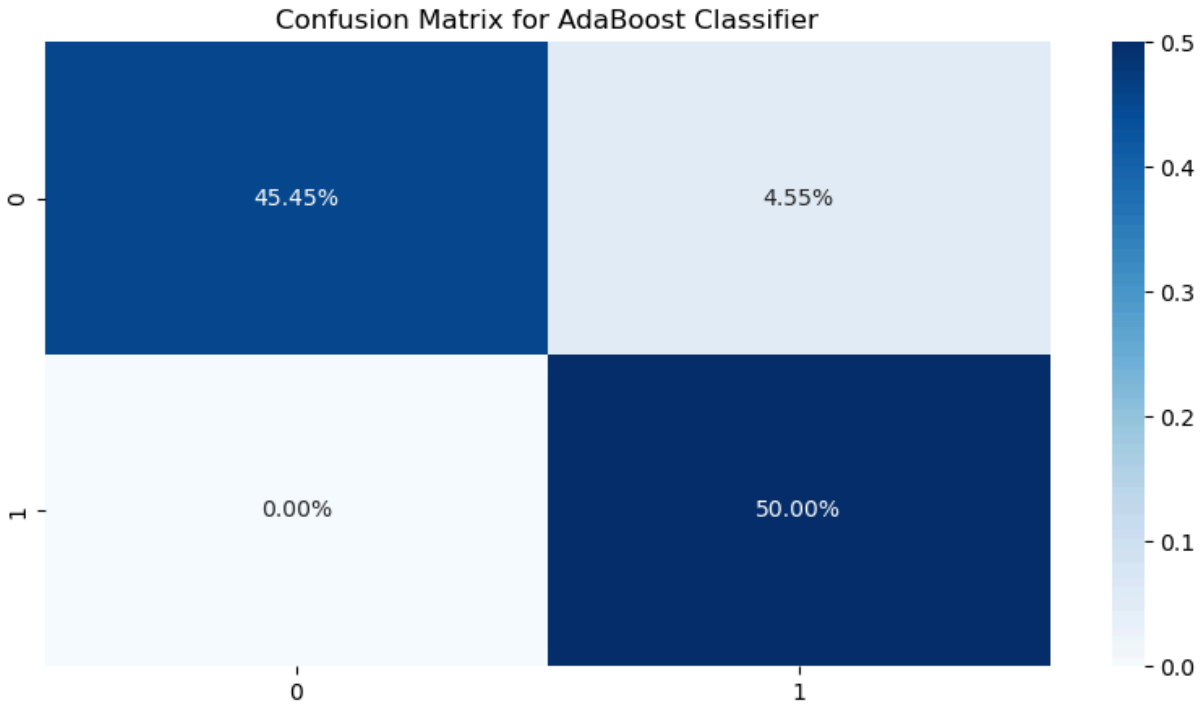
```
Classification Report for Decision Tree:
              precision    recall  f1-score   support

       False       0.80      0.36      0.50        11
        True       0.59      0.91      0.71        11

    accuracy                           0.64        22
   macro avg       0.69      0.64      0.61        22
weighted avg       0.69      0.64      0.61        22

Accuracy Score for Decision Tree: 0.64
```



In [78]:
```
# 4. AdaBoost Classifier
adaboost_model = AdaBoostClassifier(random_state=42, algorithm='SAMME')

# Fit the model
adaboost_model.fit(X_train, y_train)

# Make predictions
adaboost_predictions = adaboost_model.predict(X_test)

# Print classification report
print("\nClassification Report for AdaBoost Classifier:")
print(classification_report(y_test, adaboost_predictions))

# Print accuracy score
print(f"Accuracy Score for AdaBoost Classifier: {accuracy_score(y_test, adaboost_pr

# Create and display confusion matrix
```

```
adaboost_cm = confusion_matrix(y_test, adaboost_predictions)
plt.figure(figsize=(10, 5))
sns.heatmap(adaboost_cm / np.sum(adaboost_cm), annot=True, fmt='.2%', cmap='Blues')
plt.title("Confusion Matrix for AdaBoost Classifier")
plt.show()
```

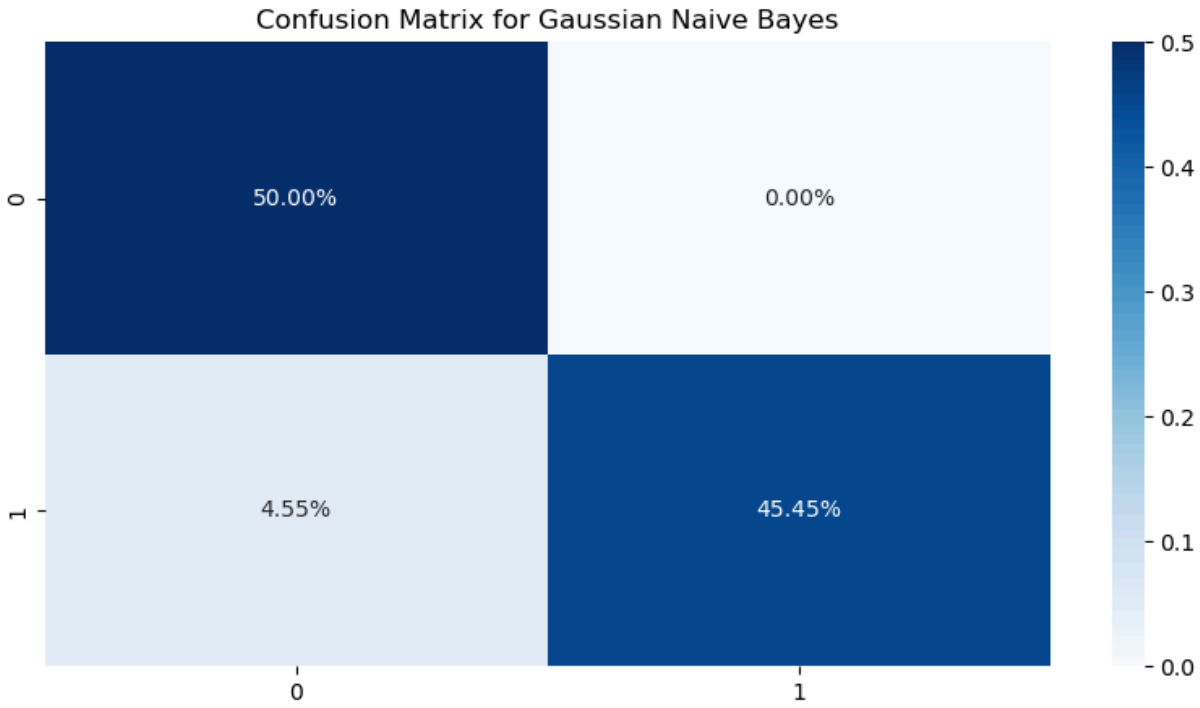
```
Classification Report for AdaBoost Classifier:
              precision    recall  f1-score   support

       False       1.00      0.91      0.95        11
        True       0.92      1.00      0.96        11

    accuracy                           0.95        22
   macro avg       0.96      0.95      0.95        22
weighted avg       0.96      0.95      0.95        22

Accuracy Score for AdaBoost Classifier: 0.95
```



In [79]:
```
# 5. Gaussian Naive Bayes
gaussian_model = GaussianNB()

# Fit the model
gaussian_model.fit(X_train, y_train)

# Make predictions
gaussian_predictions = gaussian_model.predict(X_test)

# Print classification report
print("\nClassification Report for Gaussian Naive Bayes:")
print(classification_report(y_test, gaussian_predictions))

# Print accuracy score
print(f"Accuracy Score for Gaussian Naive Bayes: {accuracy_score(y_test, gaussian_p
```

```
# Create and display confusion matrix
gaussian_cm = confusion_matrix(y_test, gaussian_predictions)
plt.figure(figsize=(10, 5))
sns.heatmap(gaussian_cm / np.sum(gaussian_cm), annot=True, fmt='.2%', cmap='Blues')
plt.title("Confusion Matrix for Gaussian Naive Bayes")
plt.show()
```

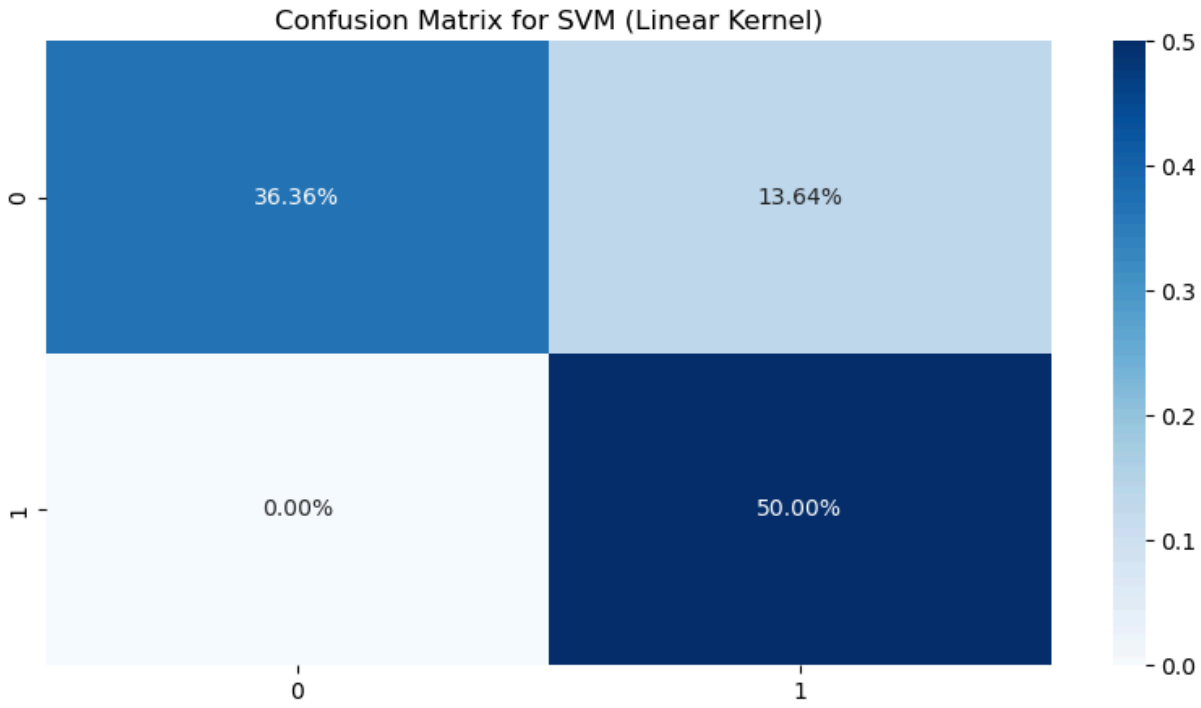
```
Classification Report for Gaussian Naive Bayes:
              precision    recall  f1-score   support

       False       0.92      1.00      0.96        11
        True       1.00      0.91      0.95        11

    accuracy                           0.95        22
   macro avg       0.96      0.95      0.95        22
weighted avg       0.96      0.95      0.95        22

Accuracy Score for Gaussian Naive Bayes: 0.95
```



In [80]:
```
# 6. Support Vector Machine (SVM)
# SVM with linear kernel
svm_linear = SVC(kernel='linear', random_state=42)

# Fit the model
svm_linear.fit(X_train, y_train)

# Make predictions
svm_linear_predictions = svm_linear.predict(X_test)

# Print classification report
print("\nClassification Report for SVM (Linear Kernel):")
print(classification_report(y_test, svm_linear_predictions))

# Print accuracy score
```

```
print(f"Accuracy Score for SVM (Linear Kernel): {accuracy_score(y_test, svm_linear_

# Create and display confusion matrix
svm_linear_cm = confusion_matrix(y_test, svm_linear_predictions)
plt.figure(figsize=(10, 5))
sns.heatmap(svm_linear_cm / np.sum(svm_linear_cm), annot=True, fmt='.2%', cmap='Blu
plt.title("Confusion Matrix for SVM (Linear Kernel)")
plt.show()
```

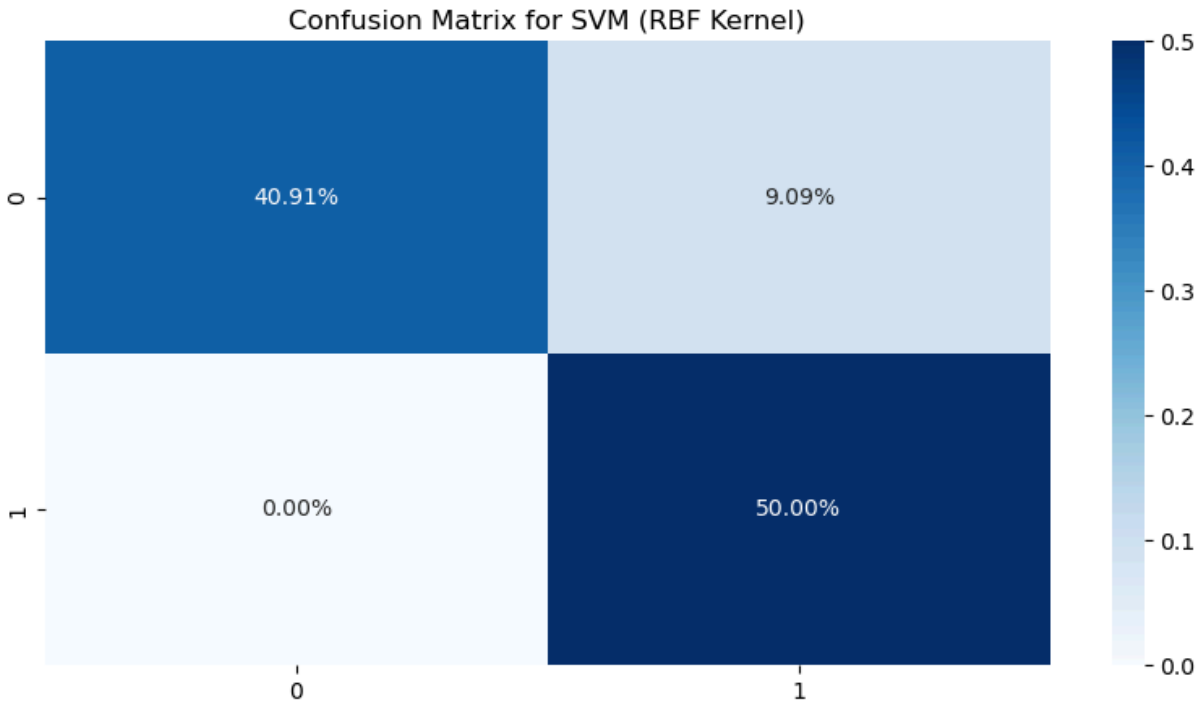
```
Classification Report for SVM (Linear Kernel):
              precision    recall  f1-score   support

       False       1.00      0.73      0.84        11
        True       0.79      1.00      0.88        11

    accuracy                           0.86        22
   macro avg       0.89      0.86      0.86        22
weighted avg       0.89      0.86      0.86        22

Accuracy Score for SVM (Linear Kernel): 0.86
```



In [81]:
```
# SVM with RBF kernel
svm_rbf = SVC(kernel='rbf', random_state=42)

# Fit the model
svm_rbf.fit(X_train, y_train)

# Make predictions
svm_rbf_predictions = svm_rbf.predict(X_test)

# Print classification report
print("\nClassification Report for SVM (RBF Kernel):")
print(classification_report(y_test, svm_rbf_predictions))
```

```
# Print accuracy score
print(f"Accuracy Score for SVM (RBF Kernel): {accuracy_score(y_test, svm_rbf_predic

# Create and display confusion matrix
svm_rbf_cm = confusion_matrix(y_test, svm_rbf_predictions)
plt.figure(figsize=(10, 5))
sns.heatmap(svm_rbf_cm / np.sum(svm_rbf_cm), annot=True, fmt='.2%', cmap='Blues')
plt.title("Confusion Matrix for SVM (RBF Kernel)")
plt.show()
```

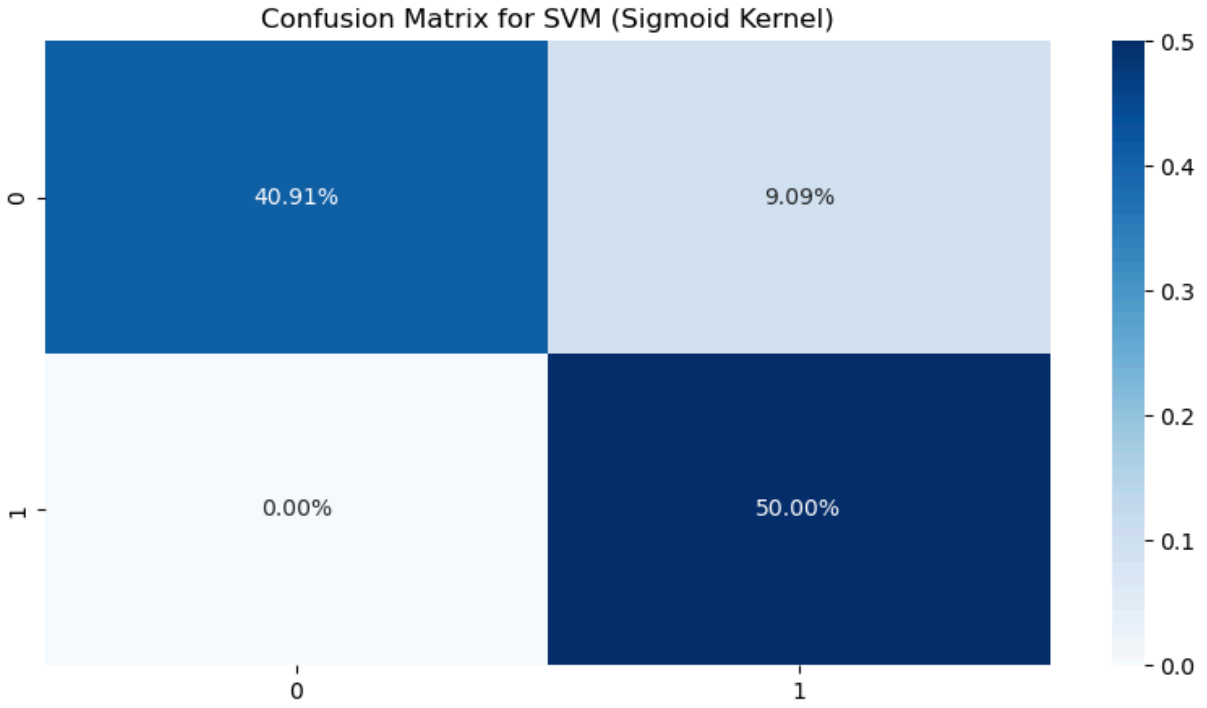
```
Classification Report for SVM (RBF Kernel):
              precision    recall  f1-score   support

       False       1.00      0.82      0.90        11
        True       0.85      1.00      0.92        11

    accuracy                           0.91        22
   macro avg       0.92      0.91      0.91        22
weighted avg       0.92      0.91      0.91        22

Accuracy Score for SVM (RBF Kernel): 0.91
```



In [82]:
```
# SVM with Sigmoid kernel
svm_sigmoid = SVC(kernel='sigmoid', random_state=42)

# Fit the model
svm_sigmoid.fit(X_train, y_train)

# Make predictions
svm_sigmoid_predictions = svm_sigmoid.predict(X_test)

# Print classification report
print("\nClassification Report for SVM (Sigmoid Kernel):")
print(classification_report(y_test, svm_sigmoid_predictions))
```

```
# Print accuracy score
print(f"Accuracy Score for SVM (Sigmoid Kernel): {accuracy_score(y_test, svm_sigmoi

# Create and display confusion matrix
svm_sigmoid_cm = confusion_matrix(y_test, svm_sigmoid_predictions)
plt.figure(figsize=(10, 5))
sns.heatmap(svm_sigmoid_cm / np.sum(svm_sigmoid_cm), annot=True, fmt='.2%', cmap='B
plt.title("Confusion Matrix for SVM (Sigmoid Kernel)")
plt.show()
```

```
Classification Report for SVM (Sigmoid Kernel):
              precision    recall  f1-score   support

       False       1.00      0.82      0.90        11
        True       0.85      1.00      0.92        11

    accuracy                           0.91        22
   macro avg       0.92      0.91      0.91        22
weighted avg       0.92      0.91      0.91        22

Accuracy Score for SVM (Sigmoid Kernel): 0.91
```



In [83]:
```
#comparison of model performance
# Define models and their accuracy scores
models = ["KNN", "MLP", "Decision Tree", "AdaBoost", "SVM (Linear)", "SVM (RBF)", "
accuracies = [
    accuracy_score(y_test, knn_predictions),
    accuracy_score(y_test, mlp_predictions),
    accuracy_score(y_test, dt_predictions),
    accuracy_score(y_test, adaboost_predictions),
    accuracy_score(y_test, svm_linear_predictions),
    accuracy_score(y_test, svm_rbf_predictions),
    accuracy_score(y_test, svm_sigmoid_predictions)
```

```
]

# Create a summary table of accuracy scores
performance_df = pd.DataFrame({"Model": models, "Accuracy": accuracies})
print("\nModel Performance Summary:")
print(performance_df)
```

```
Model Performance Summary:
           Model  Accuracy
0            KNN  0.772727
1            MLP  0.909091
2  Decision Tree  0.636364
3       AdaBoost  0.954545
4   SVM (Linear)  0.863636
5      SVM (RBF)  0.909091
6  SVM (Sigmoid)  0.909091
```