Assignment 03

Name:- Sk Fardeen Hossain

Roll:- 2021CSB023

Department:- Computer Science and Technology

G-Suite ID:- 2021csb023.sk@students.iiests.ac.in

Question 1:

Download the Forest Cover Type Dataset and preprocess the dummy variables to create training, test and development set. Reduce the train data size if the system is unable to process the whole dataset.

In [6]:
```
import pandas as pd
forest_cover_df = pd.read_csv('../ML_DRIVE/Assignment03/covtype.csv')
forest_cover_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 581012 entries, 0 to 581011
Data columns (total 55 columns):
 #   Column                              Non-Null Count   Dtype
---  ------                              --------------   -----
 0   Elevation                           581012 non-null  int64
 1   Aspect                              581012 non-null  int64
 2   Slope                               581012 non-null  int64
 3   Horizontal_Distance_To_Hydrology    581012 non-null  int64
 4   Vertical_Distance_To_Hydrology      581012 non-null  int64
 5   Horizontal_Distance_To_Roadways     581012 non-null  int64
 6   Hillshade_9am                       581012 non-null  int64
 7   Hillshade_Noon                      581012 non-null  int64
 8   Hillshade_3pm                       581012 non-null  int64
 9   Horizontal_Distance_To_Fire_Points  581012 non-null  int64
 10  Wilderness_Area1                    581012 non-null  int64
 11  Wilderness_Area2                    581012 non-null  int64
 12  Wilderness_Area3                    581012 non-null  int64
 13  Wilderness_Area4                    581012 non-null  int64
 14  Soil_Type1                          581012 non-null  int64
 15  Soil_Type2                          581012 non-null  int64
 16  Soil_Type3                          581012 non-null  int64
 17  Soil_Type4                          581012 non-null  int64
 18  Soil_Type5                          581012 non-null  int64
 19  Soil_Type6                          581012 non-null  int64
 20  Soil_Type7                          581012 non-null  int64
 21  Soil_Type8                          581012 non-null  int64
 22  Soil_Type9                          581012 non-null  int64
 23  Soil_Type10                         581012 non-null  int64
 24  Soil_Type11                         581012 non-null  int64
 25  Soil_Type12                         581012 non-null  int64
 26  Soil_Type13                         581012 non-null  int64
 27  Soil_Type14                         581012 non-null  int64
 28  Soil_Type15                         581012 non-null  int64
 29  Soil_Type16                         581012 non-null  int64
 30  Soil_Type17                         581012 non-null  int64
 31  Soil_Type18                         581012 non-null  int64
 32  Soil_Type19                         581012 non-null  int64
 33  Soil_Type20                         581012 non-null  int64
 34  Soil_Type21                         581012 non-null  int64
 35  Soil_Type22                         581012 non-null  int64
 36  Soil_Type23                         581012 non-null  int64
 37  Soil_Type24                         581012 non-null  int64
 38  Soil_Type25                         581012 non-null  int64
 39  Soil_Type26                         581012 non-null  int64
 40  Soil_Type27                         581012 non-null  int64
```

```
 41   Soil_Type28                         581012 non-null  int64
 42   Soil_Type29                         581012 non-null  int64
 43   Soil_Type30                         581012 non-null  int64
 44   Soil_Type31                         581012 non-null  int64
 45   Soil_Type32                         581012 non-null  int64
 46   Soil_Type33                         581012 non-null  int64
 47   Soil_Type34                         581012 non-null  int64
 48   Soil_Type35                         581012 non-null  int64
 49   Soil_Type36                         581012 non-null  int64
 50   Soil_Type37                         581012 non-null  int64
 51   Soil_Type38                         581012 non-null  int64
 52   Soil_Type39                         581012 non-null  int64
 53   Soil_Type40                         581012 non-null  int64
 54   Cover_Type                          581012 non-null  int64
dtypes: int64(55)
memory usage: 243.8 MB
```

As we can see above, all the columns are important since all have non-null values

In [7]:
```
forest_cover_df.head()
```

Out[7]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillsh
0	2596	51	3	258	0	510	221	
1	2590	56	2	212	-6	390	220	
2	2804	139	9	268	65	3180	234	
3	2785	155	18	242	118	3090	238	
4	2595	45	2	153	-1	391	220	

5 rows × 55 columns

Standardisation of non-binary data columns

In [8]:
```
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

'''
    @param:- [df:pd.DataFrame, col_name: string]
    @brief:- Function returns a standardised dataframe on the column `col_name`
    @return:- Standardised Dataframe
'''
```

```python
def standardise(df:"pd.DataFrame",col_name:"str")->"pd.DataFrame":
    df[[col_name]]=pd.DataFrame(
        data = scaler.fit_transform(df[[col_name]]),
        columns=[col_name],
        index=df.index
    )

    return df
```

In [9]:

```python
'''
 Elevation                           581012 non-null  int64
 1   Aspect                              581012 non-null  int64
 2   Slope                               581012 non-null  int64
 3   Horizontal_Distance_To_Hydrology    581012 non-null  int64
 4   Vertical_Distance_To_Hydrology      581012 non-null  int64
 5   Horizontal_Distance_To_Roadways     581012 non-null  int64
 6   Hillshade_9am                       581012 non-null  int64
 7   Hillshade_Noon                      581012 non-null  int64
 8   Hillshade_3pm                       581012 non-null  int64
 9   Horizontal_Distance_To_Fire_Points  581012 non-null  int64
'''
columns_to_standardise = ['Elevation','Aspect','Slope','Horizontal_Distance_To_Hydrology','Vertical_Distance_To_Hydrology',
                          'Horizontal_Distance_To_Roadways','Hillshade_9am','Hillshade_Noon','Hillshade_3pm','Horizontal_Distan


for col_name in columns_to_standardise:
    forest_cover_df=standardise(forest_cover_df,col_name)

forest_cover_df.head()
```

Out[9]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am
0	-1.297805	-0.935157	-1.482820	-0.053767	-0.796273	-1.180146	0.330743
1	-1.319235	-0.890480	-1.616363	-0.270188	-0.899197	-1.257106	0.293388
2	-0.554907	-0.148836	-0.681563	-0.006719	0.318742	0.532212	0.816364
3	-0.622768	-0.005869	0.520322	-0.129044	1.227908	0.474492	0.965786
4	-1.301377	-0.988770	-1.616363	-0.547771	-0.813427	-1.256464	0.293388

5 rows × 55 columns

In [10]:

```python
## Number of target classes
```

```python
forest_cover_df[['Cover_Type']].value_counts()
```

Out[10]:
```
Cover_Type
2             283301
1             211840
3              35754
7              20510
6              17367
5               9493
4               2747
Name: count, dtype: int64
```

Since there high number of samples especially from classes 1 , 2 , 3 we need to perform random sampling across population points

In [11]:
```python
forest_cover_df_rough = forest_cover_df.sample(frac=0.1,random_state=5)
forest_cover_df_rough[['Cover_Type']].value_counts()
```

Out[11]:
```
Cover_Type
2             28366
1             21062
3              3611
7              2068
6              1724
5               985
4               285
Name: count, dtype: int64
```

In [12]:
```python
y=forest_cover_df_rough[['Cover_Type']]
X=forest_cover_df_rough.drop(columns=['Cover_Type'])
X
```

Out[12]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade
456056	1.223763	1.566773	0.653865	1.842264	3.029086	0.569409	-1.49
456659	0.280854	1.816965	-0.280934	-0.067882	-0.213034	-0.778672	-0.37
210876	1.095185	-0.827931	-0.414477	-0.703029	-0.264496	2.261886	0.66
490911	-0.097739	-0.827931	0.653865	0.962468	1.159292	-1.083305	0.77
267218	-2.994328	1.575708	1.321579	-1.126460	-0.659040	-0.806890	-2.06
...	...	...	...	...	...	...	
509165	0.777310	-1.194285	0.386780	0.938944	0.816211	-0.506747	-0.19
261011	0.266567	1.057451	0.119694	-0.373693	0.130048	-1.256464	-1.20
278278	-2.165710	1.745482	1.989293	-0.561885	0.181510	-1.316750	-2.06
494618	0.923747	1.736546	-0.013849	-0.105520	-0.110110	-1.171167	-0.67
24166	0.373716	-0.184578	-1.082191	-1.126460	-0.727656	2.867304	0.66

58101 rows × 54 columns

In [13]:

```
# 80% -> train | 10% -> validation | 10% -> testing
from sklearn.model_selection import train_test_split

X_train,X_rest,y_train,y_rest = train_test_split(X,y,random_state=5,test_size=0.2)
X_val,X_test,y_val,y_test = train_test_split(X_rest,y_rest,random_state=5,test_size=0.5)
```

Question 02:

Consider only two features and three classes and train Logistic Regression 3- class classifier (any three class) to show the training and test area in a 2D plane, using matplotlib.

Here we are choosing the features `Elevation` and `SLope` which are to be mapped to classes `1`, `2`, `3`

In [14]:

```
X_train_subset = X_train[['Elevation','Slope']]
X_train_subset
```

Out[14]:

	Elevation	Slope
257482	0.366573	0.520322
264358	-0.597766	1.588665
1767	-0.358467	0.787408
58906	-0.447758	-0.147392
390314	0.902317	-1.215734
...	...	...
53980	0.173705	0.119694
162900	0.988036	0.520322
41957	-1.065649	0.787408
15266	-0.719202	-0.280934
474993	-0.229889	1.989293

46480 rows × 2 columns

In [15]:

```
train_subset = X_train_subset.join(y_train)
train_subset = train_subset[train_subset['Cover_Type'].isin([1,2,3])]
train_subset
```

Out[15]:

	Elevation	Slope	Cover_Type
257482	0.366573	0.520322	1
264358	-0.597766	1.588665	2
1767	-0.358467	0.787408	2
58906	-0.447758	-0.147392	2
390314	0.902317	-1.215734	1
...	...	...	...
53980	0.173705	0.119694	1
162900	0.988036	0.520322	2
41957	-1.065649	0.787408	2
15266	-0.719202	-0.280934	2
474993	-0.229889	1.989293	2

42451 rows × 3 columns

In [16]:

```
from sklearn.linear_model import LogisticRegression

LR_model = LogisticRegression(
    solver='saga',
    max_iter=10000,
)
LR_model.fit(train_subset.iloc[:,0:2],train_subset.iloc[:,2])
```

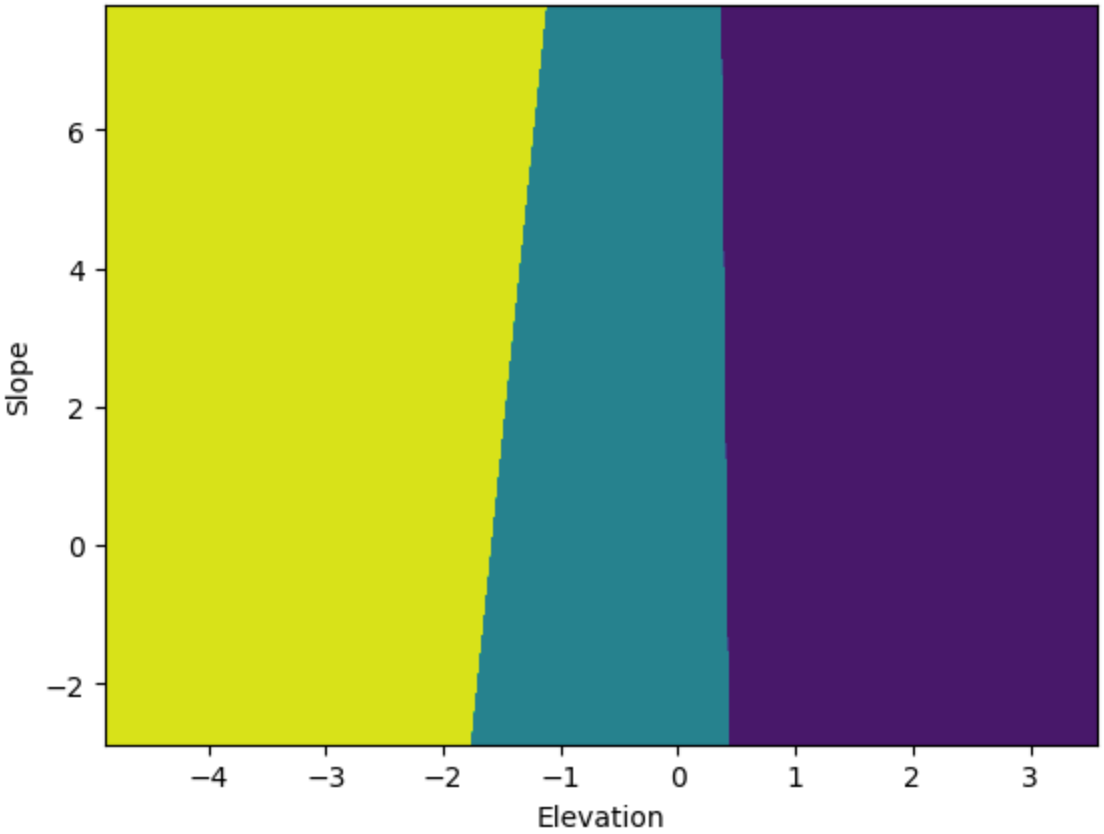
Out[16]:

```
▾ LogisticRegression                ⓘ ?

LogisticRegression(max_iter=10000, solver='saga')
```

In [17]:

```
# Training Data
from sklearn.inspection import DecisionBoundaryDisplay

display_fig = DecisionBoundaryDisplay.from_estimator(
    LR_model,
    train_subset.iloc[:,0:2],
    xlabel = 'Elevation',
    ylabel = 'Slope',
    grid_resolution = 6000,
```

```
)
```



In [18]:

```
X_test_subset = X_test[['Elevation','Slope']]

test_subset = X_test_subset.join(y_test)
test_subset = test_subset[test_subset['Cover_Type'].isin([1,2,3])]
test_subset
```

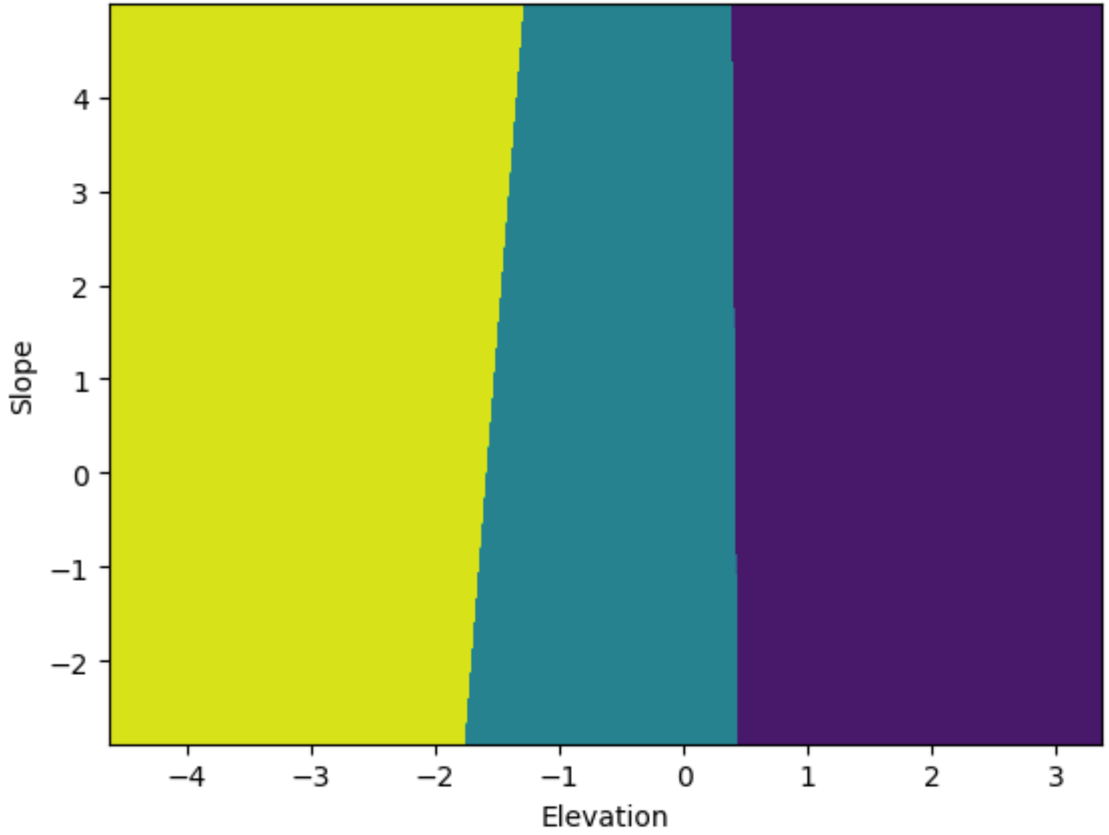
Out[18]:

	Elevation	Slope	Cover_Type
402158	0.005839	-1.349277	2
271152	-2.097849	-1.215734	3
318946	0.305855	0.119694	2
554660	-1.008503	0.787408	2
450417	1.366628	0.119694	1
...	...	...	...
521316	0.748737	0.920951	1
540121	0.327285	-0.013849	2
153065	-0.176314	0.520322	2
192414	0.927318	-1.349277	1
445729	0.852314	0.386780	2

5311 rows × 3 columns

In [19]:

```
# Test Data

display_fig = DecisionBoundaryDisplay.from_estimator(
    LR_model,
    test_subset.iloc[:,0:2],
    xlabel = 'Elevation',
    ylabel = 'Slope',
    grid_resolution = 6000
)
```



Question 03:

Analyze and control the overfitting by varying the inverse of regularization strength parameter (0.1, 0.25, 0.5, 0.75, 0.9) and plot the accuracy graph for the test set.

In [20]:
```
from sklearn.metrics import accuracy_score

'''
    @params: - X_train,y_train,X_val,y_val,X_test,y_test,solver,inv_reg_str,penalty
    @return:- list([accuracy_before_finetune,accuracy_after_finetune])
    @brief:- Trains and tests a LR model on training, testing and validation dataset
'''

def LR_model_util(
    X_train: "pd.DataFrame",
    y_train: "pd.DataFrame",
    X_val: "pd.DataFrame",
    y_val: "pd.DataFrame",
```

```python
    X_test: "pd.DataFrame",
    y_test: "pd.DataFrame",
    solver: "str",
    inv_reg_str: "double",
    penalty: "str"
)->"list":
    model = LogisticRegression(
        solver=solver,
        C=inv_reg_str,
        max_iter=10000,
        penalty=penalty
    )
    model.fit(X_train,y_train)
    y_pred_before_finetune = model.predict(X_val)
    accuracy_before_finetuning = accuracy_score(y_val,y_pred_before_finetune)
    model.fit(X_val,y_val)
    y_pred_after_finetune = model.predict(X_test)
    accuracy_after_finetuning= accuracy_score(y_test,y_pred_after_finetune)

    return [accuracy_before_finetuning,accuracy_after_finetuning]
```

In [21]:

```python
X_val_subset = X_val[['Elevation','Slope']]


val_subset = X_val_subset.join(y_val)
val_subset = val_subset[val_subset['Cover_Type'].isin([1,2,3])]
val_subset
```

Out[21]:

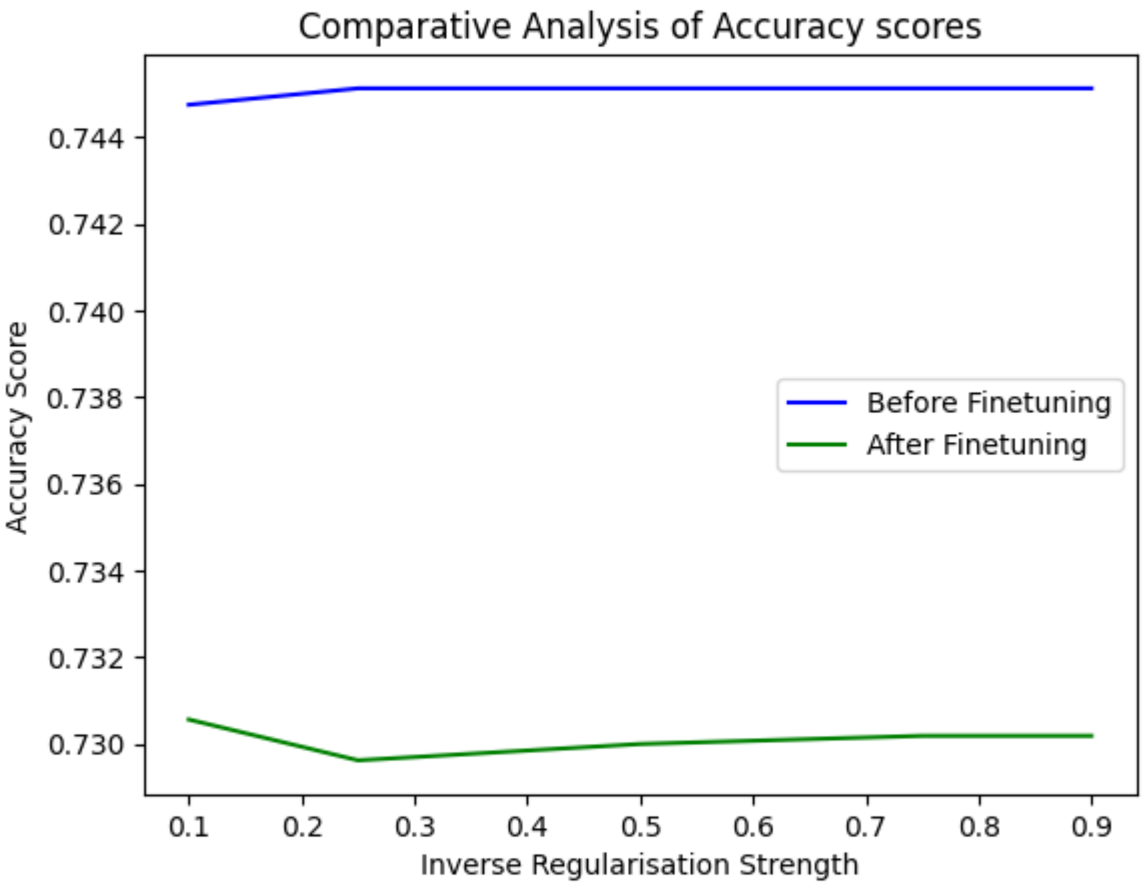
	Elevation	Slope	Cover_Type
286194	-0.072737	-0.013849	2
519090	-0.069166	-0.147392	2
429189	-0.069166	0.386780	1
175544	0.898745	-0.414477	2
320041	-1.554962	-0.414477	2
...	...	...	...
120920	-0.154885	-0.681563	2
451493	1.013037	-0.548020	1
391006	0.716592	0.386780	1
508515	0.948748	-0.147392	1
108165	-1.165655	-0.147392	2

5277 rows × 3 columns

In [22]:

```
import matplotlib.pyplot as plt

inverse_regularization_strength_list = [0.1,0.25,0.5,0.75,0.9]

train_subset_X = train_subset.iloc[:,0:2]
train_subset_y = train_subset.iloc[:,2]
val_subset_X = val_subset.iloc[:,0:2]
val_subset_y = val_subset.iloc[:,2]
test_subset_X = test_subset.iloc[:,0:2]
test_subset_y = test_subset.iloc[:,2]

accuracy_before_finetune_list=[]
accuracy_after_finetune_list=[]

for C in inverse_regularization_strength_list:
    [acc_b_f,acc_a_f] = LR_model_util(train_subset_X,train_subset_y,val_subset_X,val_subset_y,
                                test_subset_X,test_subset_y,"saga",C,"l1")
    accuracy_before_finetune_list.append(acc_b_f)
    accuracy_after_finetune_list.append(acc_a_f)

plt.plot(inverse_regularization_strength_list,accuracy_before_finetune_list,"blue",label="Before Finetuning")
plt.plot(inverse_regularization_strength_list,accuracy_after_finetune_list,"green",label="After Finetuning")
```

```
plt.title("Comparative Analysis of Accuracy scores")
plt.xlabel('Inverse Regularisation Strength')
plt.ylabel('Accuracy Score')

plt.legend()

plt.show()
```



Question 04:

Apply multiclass classification in Support Vector Machine (SVM) using Forest Cover Type dataset.

In [39]:
```
# Using a Support vector machine to perform multi-class classification
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
```

```
'''
    @params:- X_train,y_train,X_val,y_val,X_test,y_test,penalty,inverse regularisation strength,loss
    @return:- list[penalty,inv_reg_str,loss,accuracy_before_finetune,accuracy_after_finetune,precision_before_finetune,precisio
                recall_before_finetune,recall_after_finetune,f1score_before_finetune,f1score_after_finetune]
    @brief:- Function creates Linear SVC(Support Vector Classifier) based on tuned parameters and returns its accuracy metrics
'''
def SVC_util(
    X_train: "pd.DataFrame",
    y_train: "pd.DataFrame",
    X_val: "pd.DataFrame",
    y_val: "pd.DataFrame",
    X_test: "pd.DataFrame",
    y_test: "pd.DataFrame",
    penalty: 'str',
    inv_reg_str: "double",
    loss: 'str'
)->"list":
        model = LinearSVC(
            max_iter=10000,
            C = inv_reg_str,
            loss=loss
        ).fit(X_train,y_train)
        y_pred = model.predict(X_val)
        accuracy_before_finetune = accuracy_score(y_val,y_pred)
        precision_before_finetune = precision_score(y_val,y_pred,average='macro').item()
        recall_before_finetune = recall_score(y_val,y_pred,average='macro').item()
        f1score_before_finetune = f1_score(y_val,y_pred,average='macro').item()
        model.fit(X_val,y_val)
        y_pred = model.predict(X_test)
        accuracy_after_finetune = accuracy_score(y_test,y_pred)
        precision_after_finetune = precision_score(y_test,y_pred,average='macro').item()
        recall_after_finetune = recall_score(y_test,y_pred,average='macro').item()
        f1score_after_finetune = f1_score(y_test,y_pred,average='macro').item()
        return [penalty,inv_reg_str,loss,accuracy_before_finetune,accuracy_after_finetune,precision_before_finetune,precision_a
                recall_before_finetune,recall_after_finetune,f1score_before_finetune,f1score_after_finetune]
```

In [41]:

```
kernels = ['linear','rbf','poly','sigmoid','precomputed']

linear_1 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.1,'hinge')
linear_2 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.1,'squared_hinge')
linear_3 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.1,'hinge')
linear_4 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.1,'squared_hinge')

linear_5 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.25,'hinge')
linear_6 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.25,'squared_hinge')
```

```
linear_7 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.25,'hinge')
linear_8 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.25,'squared_hinge')

linear_9 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.5,'hinge')
linear_10 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.5,'squared_hinge')
linear_11 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.5,'hinge')
linear_12 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.5,'squared_hinge')

linear_13 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.75,'hinge')
linear_14 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.75,'squared_hinge')
linear_15 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.75,'hinge')
linear_16 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.75,'squared_hinge')

linear_17 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.9,'hinge')
linear_18 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l1',0.9,'squared_hinge')
linear_19 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.9,'hinge')
linear_20 = SVC_util(X_train,y_train,X_val,y_val,X_test,y_test,'l2',0.9,'squared_hinge')

SVC_report = pd.DataFrame(
    columns=['penalty','inv_reg_strength','loss','accuracy_before_finetune','accuracy_after_finetune','precision_before_finetun
             'recall_before_finetune','recall_after_finetune','f1score_before_finetune','f1score_after_finetune'],
    data = [linear_1,linear_2,linear_3,linear_4,linear_5,linear_6,linear_7,linear_8,linear_9,linear_10,
            linear_11,linear_12,linear_13,linear_14,linear_15,linear_16,linear_17,linear_18,linear_19,
            linear_20]
)

SVC_report
```

Out[41]:

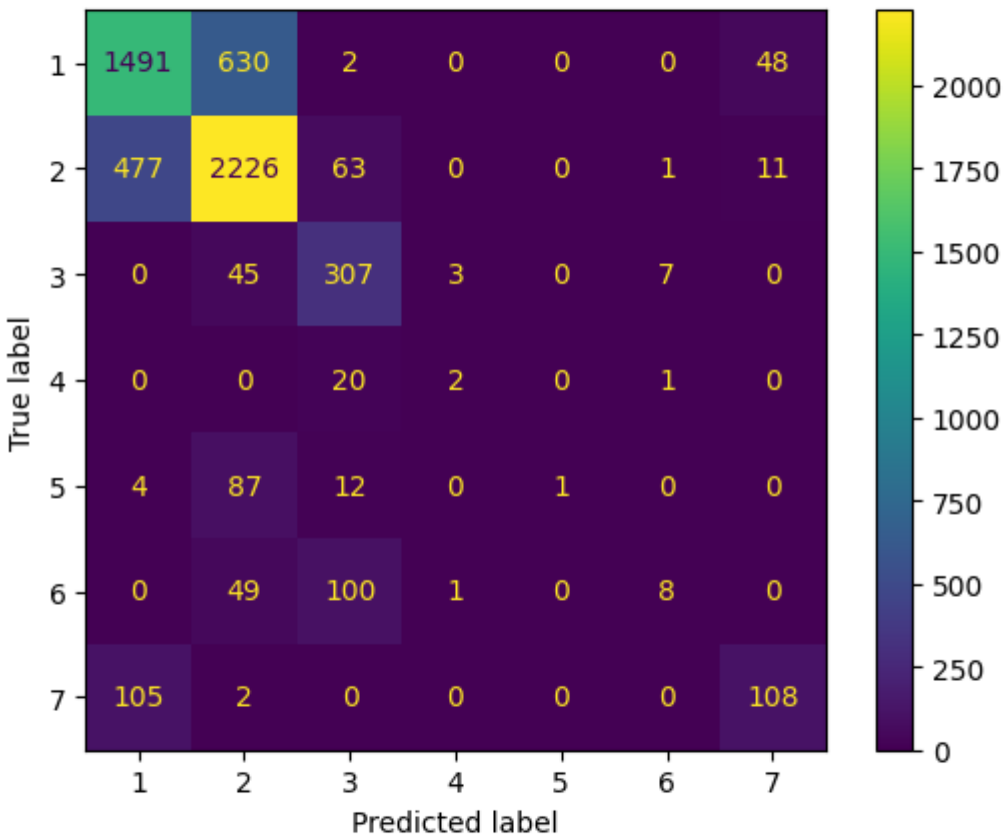
	penalty	inv_reg_strength	loss	accuracy_before_finetune	accuracy_after_finetune	precision_before_finetune	precision_after_finetune	rec
0	l1	0.10	hinge	0.713597	0.700396	0.521611	0.365776	
1	l1	0.10	squared_hinge	0.713769	0.705903	0.523590	0.425119	
2	l2	0.10	hinge	0.713597	0.700568	0.551474	0.413395	
3	l2	0.10	squared_hinge	0.713769	0.705903	0.523590	0.425119	
4	l1	0.25	hinge	0.712565	0.706763	0.461092	0.456528	
5	l1	0.25	squared_hinge	0.713941	0.708828	0.679575	0.429829	
6	l2	0.25	hinge	0.712565	0.706419	0.488530	0.406084	
7	l2	0.25	squared_hinge	0.713941	0.708828	0.679575	0.429829	
8	l1	0.50	hinge	0.713253	0.708828	0.530967	0.442193	
9	l1	0.50	squared_hinge	0.714114	0.709000	0.679629	0.576261	
10	l2	0.50	hinge	0.712392	0.707796	0.482432	0.407406	
11	l2	0.50	squared_hinge	0.714114	0.709000	0.679629	0.576261	
12	l1	0.75	hinge	0.713597	0.710377	0.548501	0.452501	
13	l1	0.75	squared_hinge	0.714286	0.707968	0.682814	0.574935	
14	l2	0.75	hinge	0.714114	0.709861	0.548739	0.429317	
15	l2	0.75	squared_hinge	0.714286	0.707968	0.682814	0.574935	
16	l1	0.90	hinge	0.713425	0.709861	0.528923	0.531003	
17	l1	0.90	squared_hinge	0.714114	0.708312	0.682751	0.574776	
18	l2	0.90	hinge	0.713425	0.708828	0.492822	0.480583	
19	l2	0.90	squared_hinge	0.714114	0.708312	0.682751	0.574776	

From the above table based on f1-score, we have seen that the LinearSVC model with l2 penalty, 0.9 inverse_reg_strength and squared_hinge_loss performs the best

Question 05:

Plot and analyze the confusion matrix for the above applied SVM method.

In [43]:
```
from sklearn.metrics import ConfusionMatrixDisplay,confusion_matrix

model = LinearSVC(
    penalty='l2',
    C=0.9,
    loss='squared_hinge',
    max_iter=10000
).fit(X_train,y_train)

y_pred = model.predict(X_test)
matrix = confusion_matrix(y_test,y_pred,labels=model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=matrix, display_labels= model.classes_)

disp.plot()

plt.show()
```



From the above confusion matrix we can conclude that the major misclassfication occur among groups:-

1. (1,2)

2. (5,2)
3. (6,3)
4. (7,1)

Label 2 is most accuractely classfied from the given model among all other labels

Question 06:

Download Titanic Dataset and do the initial preprocessing and train a Decision Tree classifier and vary the maximum depth of the tree to train at least 5 classifiers to analyze the effectiveness.

In [23]:
```
TRAIN_DATA_PATH = "../ML_DRIVE/Assignment03/train.csv"
TEST_DATA_PATH = "../ML_DRIVE/Assignment03/test.csv"

train_df = pd.read_csv(TRAIN_DATA_PATH)

test_df = pd.read_csv(TEST_DATA_PATH) # only for prediction since final label 'Survived' is not given

train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [24]:
```
train_df.columns
```

Out[24]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [25]:
```
# Dropping passengerId, name, cabin, Ticket
train_df = train_df.drop("PassengerId",axis=1)
train_df = train_df.drop("Name",axis=1)
train_df = train_df.drop("Cabin",axis=1)
train_df = train_df.drop("Ticket",axis=1)
```

In [26]:
```
# Dropping null age values
train_df = train_df[train_df['Age'].notna()]
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 714 entries, 0 to 890
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  714 non-null    int64  
 1   Pclass    714 non-null    int64  
 2   Sex       714 non-null    object 
 3   Age       714 non-null    float64
 4   SibSp     714 non-null    int64  
 5   Parch     714 non-null    int64  
 6   Fare      714 non-null    float64
 7   Embarked  712 non-null    object 
dtypes: float64(2), int64(4), object(2)
memory usage: 50.2+ KB
```

In [27]:
```
train_df.head()
```

Out[27]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

Preprocessing the dataset is done !!!!

In [28]:
```
## One-hot encode `Embarked`, `Pclass` and `Sex`
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()

encoded_df = pd.DataFrame(
    encoder.fit_transform(train_df[['Embarked']]).toarray(),
    index = train_df.index,
    columns = encoder.get_feature_names_out()
)

train_df = train_df.join(encoded_df)
train_df = train_df.drop('Embarked',axis=1)

encoded_df = pd.DataFrame(
    encoder.fit_transform(train_df[['Pclass']]).toarray(),
    index = train_df.index,
    columns = encoder.get_feature_names_out()
)

train_df = train_df.join(encoded_df)
train_df = train_df.drop('Pclass',axis=1)

encoded_df = pd.DataFrame(
    encoder.fit_transform(train_df[['Sex']]).toarray(),
    index = train_df.index,
    columns = encoder.get_feature_names_out()
)

train_df = train_df.join(encoded_df)
train_df = train_df.drop('Sex',axis=1)
```

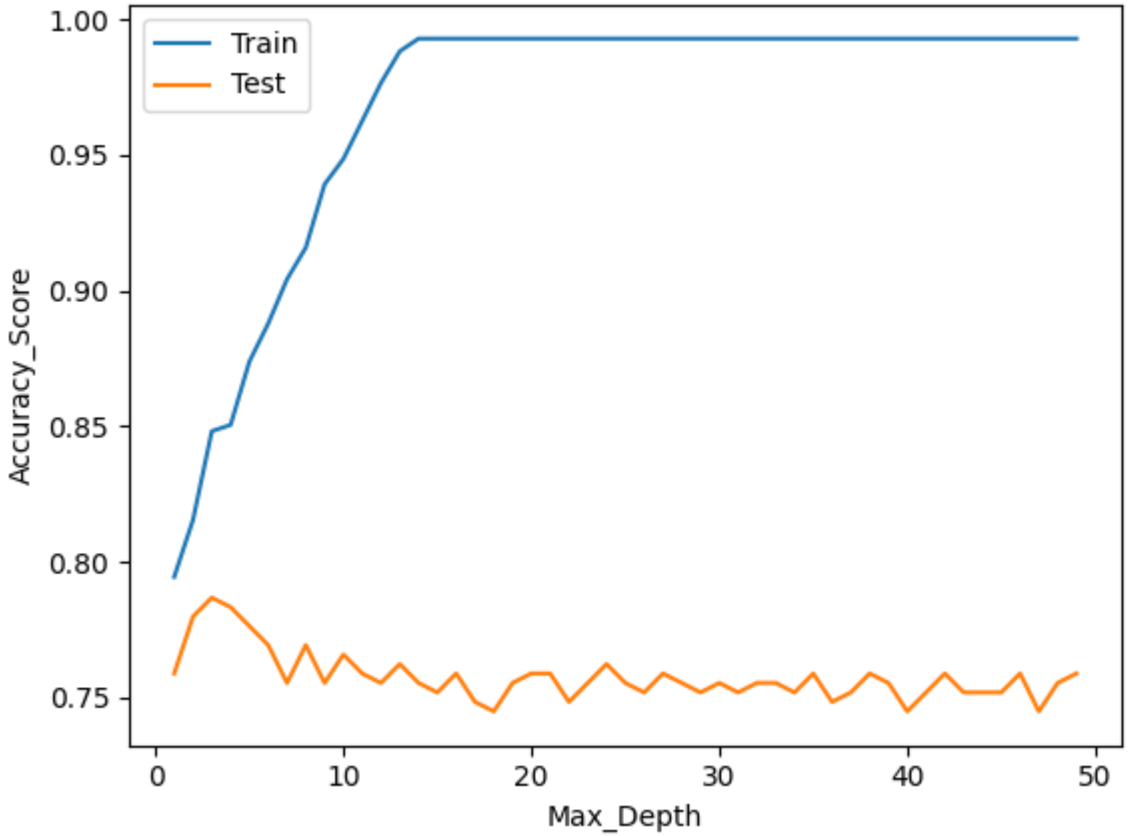
In [29]:
```
X = train_df.drop(columns=['Survived'])

y = train_df[['Survived']]
```

In [30]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.4)

from sklearn.tree import DecisionTreeClassifier

'''
 @params:- X_train,y_train,X_test,y_test,max_depth
 @return:-  Accuracy score of the model
 @brief:- Trains a decision tree classifier and then returns its accuracy score
'''
```

```python
def DT_util(
    X_train: "pd.DataFrame",
    y_train: "pd.DataFrame",
    X_test: "pd.DataFrame",
    y_test: "pd.DataFrame",
    max_depth: "integer"
)-> "float":
    model = DecisionTreeClassifier(max_depth=max_depth).fit(X_train,y_train)
    y_pred = model.predict(X_test)
    return accuracy_score(y_test,y_pred)
```

In [31]:

```python
max_depths = range(1,50)

training_accuracies = [DT_util(X_train,y_train,X_train,y_train,depth) for depth in max_depths]
testing_accuracies = [DT_util(X_train,y_train,X_test,y_test,depth) for depth in max_depths]

plt.plot(max_depths,training_accuracies,'-',label='Train')
plt.plot(max_depths,testing_accuracies,'-',label='Test')

plt.xlabel('Max_Depth')
plt.ylabel('Accuracy_Score')

plt.legend()
plt.show()
```

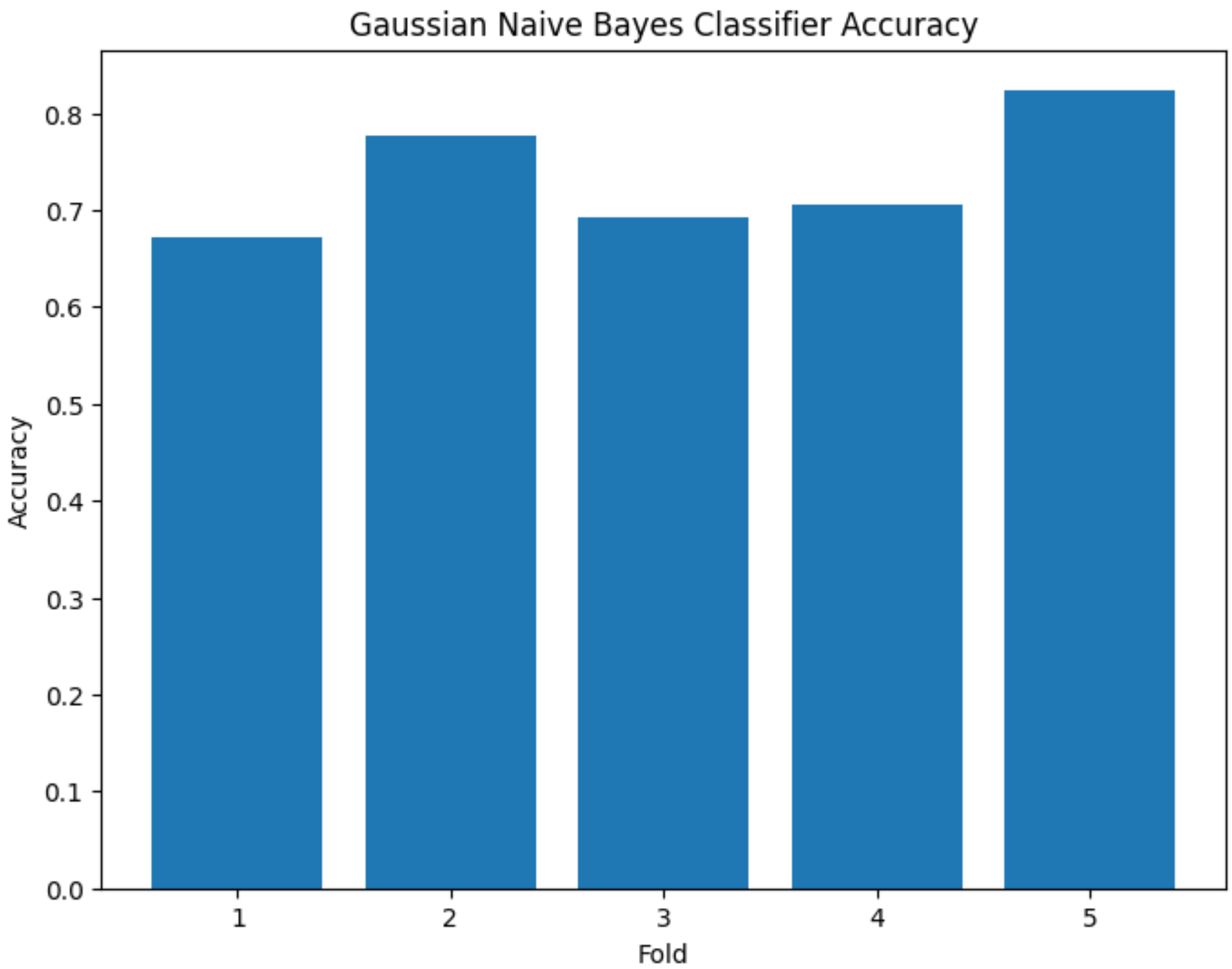


Question 07:

Estimate the average accuracy of the Naïve Bayes Classifier using 5-fold cross- validation using a scikit-learn package in python. Plot the bar graph using matplotlib.

In [48]:
```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB,ComplementNB

gnb_clf = GaussianNB()
scores = cross_val_score(gnb_clf, X, y, cv=5)
gnb_avg_accuracy = scores.mean()

# Plot the bar graph for Naive Bayes Classifier accuracy
plt.figure(figsize=(8, 6))
plt.bar(range(1, 6), scores)
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title('Gaussian Naive Bayes Classifier Accuracy')
```

```
plt.show()

print(f'Average Accuracy of Gaussian Naive Bayes Classifier: {gnb_avg_accuracy:.2f}')
```

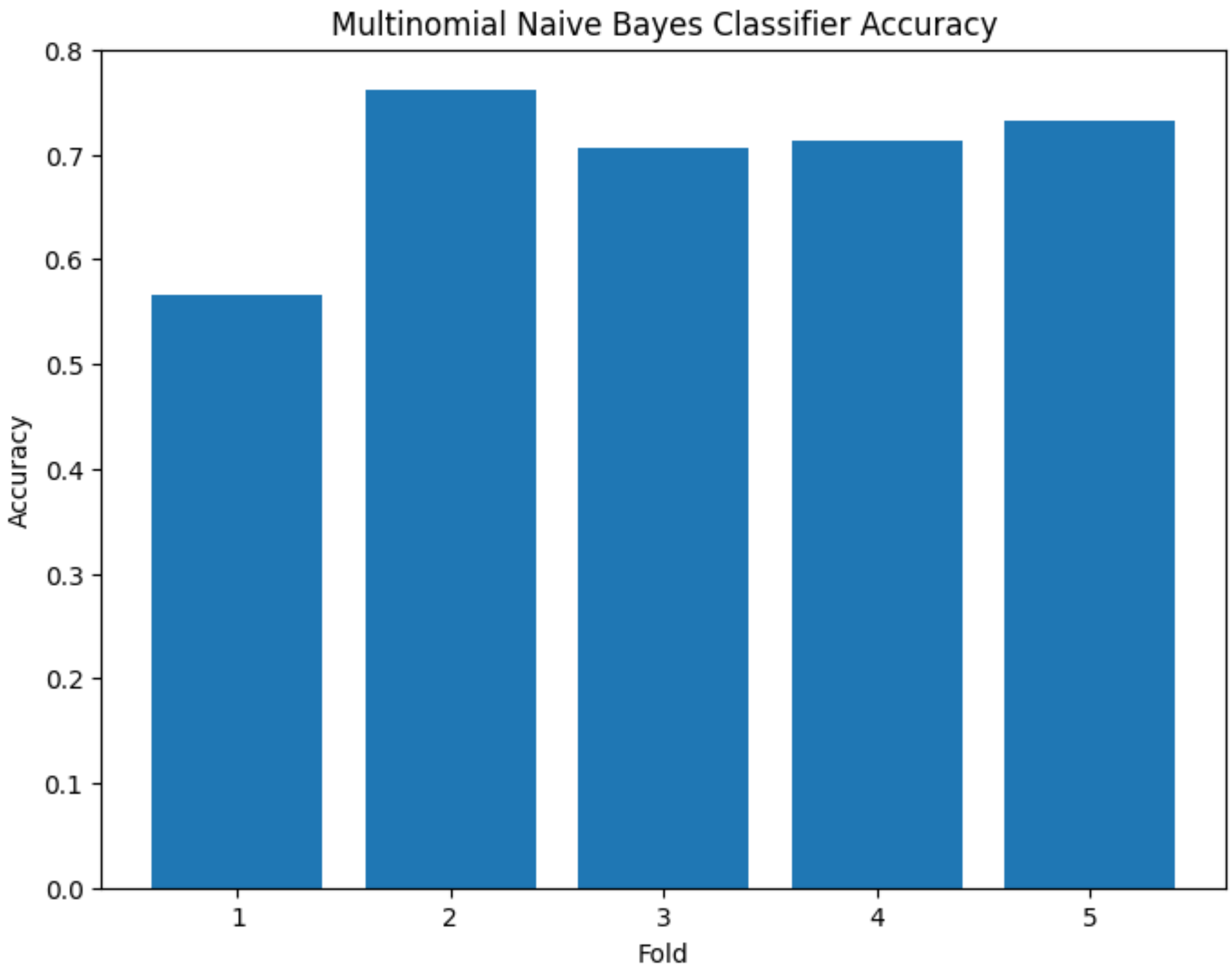



```
Average Accuracy of Gaussian Naive Bayes Classifier: 0.73
```

In [49]:

```
mnb_clf = MultinomialNB()
scores = cross_val_score(mnb_clf, X, y, cv=5)
mnb_avg_accuracy = scores.mean()

# Plot the bar graph for Naive Bayes Classifier accuracy
plt.figure(figsize=(8, 6))
plt.bar(range(1, 6), scores)
plt.xlabel('Fold')
```

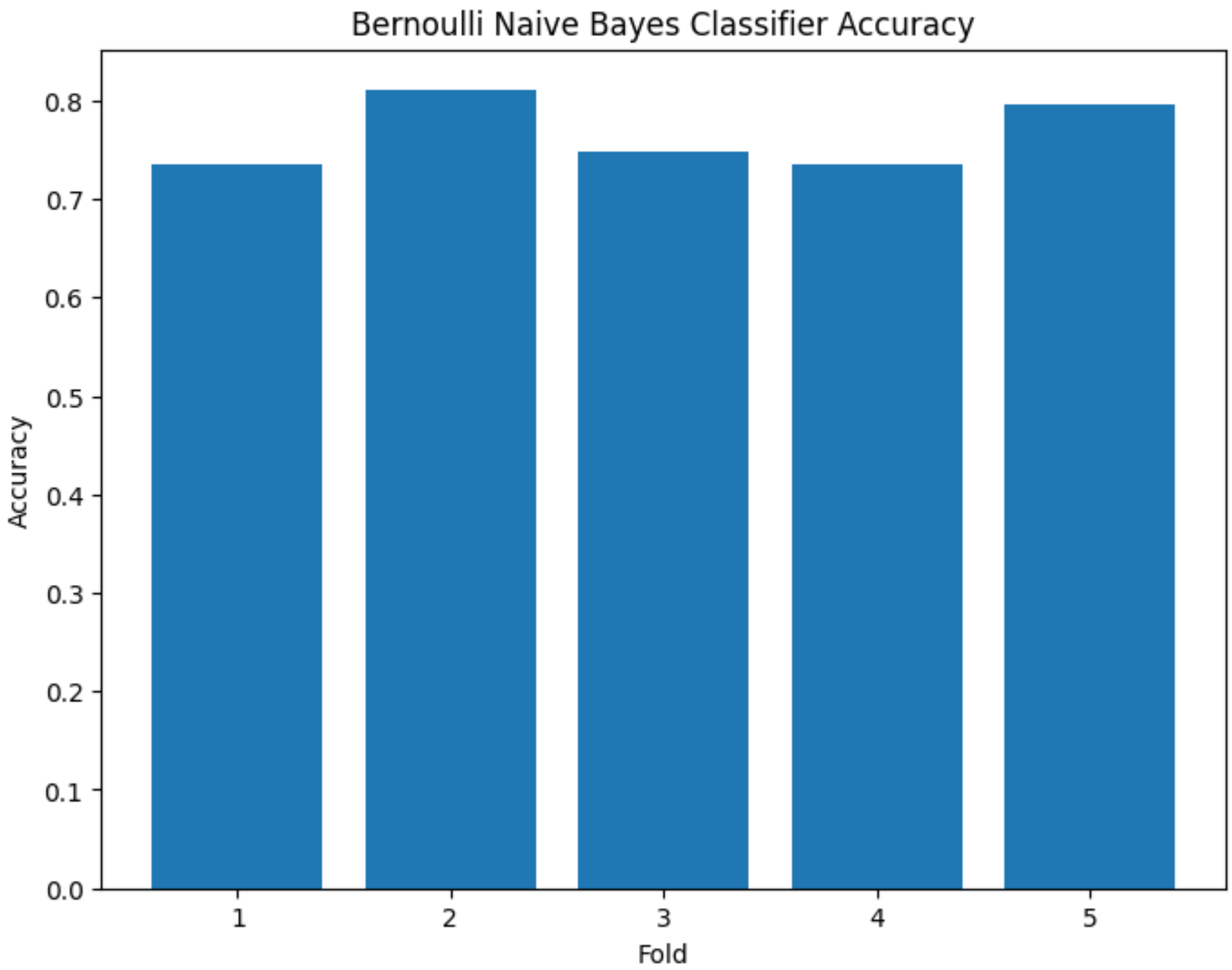
```
plt.ylabel('Accuracy')
plt.title('Multinomial Naive Bayes Classifier Accuracy')
plt.show()

print(f'Average Accuracy of Multinomial Naive Bayes Classifier: {mnb_avg_accuracy:.2f}')
```




```
Average Accuracy of Multinomial Naive Bayes Classifier: 0.70
```

In [50]:
```
bnb_clf = BernoulliNB()
scores = cross_val_score(bnb_clf, X, y, cv=5)
bnb_avg_accuracy = scores.mean()

# Plot the bar graph for Naive Bayes Classifier accuracy
plt.figure(figsize=(8, 6))
```

```
plt.bar(range(1, 6), scores)
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title('Bernoulli Naive Bayes Classifier Accuracy')
plt.show()

print(f'Average Accuracy of Bernoulli Naive Bayes Classifier: {bnb_avg_accuracy:.2f}')
```

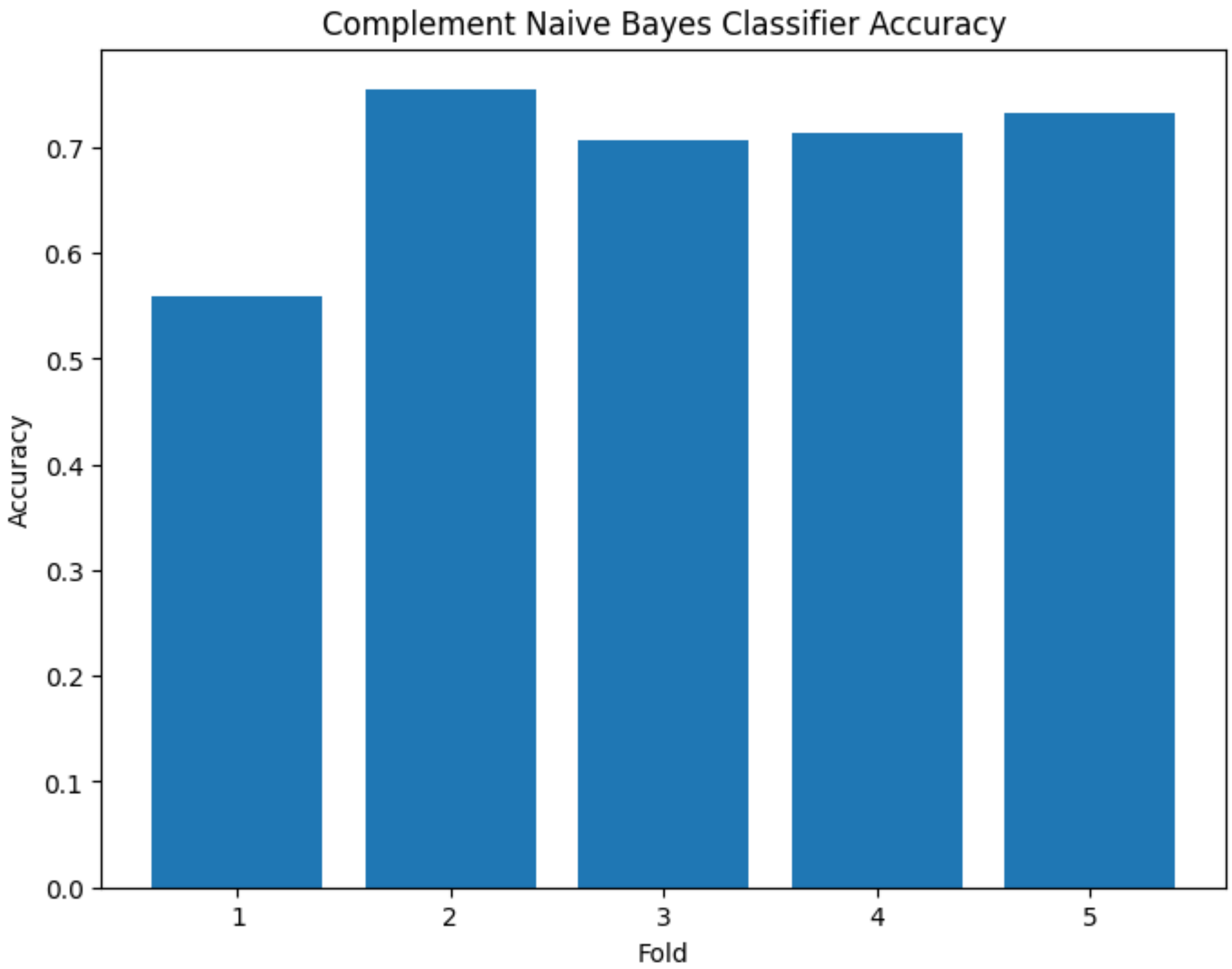



```
Average Accuracy of Bernoulli Naive Bayes Classifier: 0.76
```

In [51]:

```
cnb_clf = ComplementNB()
scores = cross_val_score(cnb_clf, X, y, cv=5)
cnb_avg_accuracy = scores.mean()
```

```python
# Plot the bar graph for Naive Bayes Classifier accuracy
plt.figure(figsize=(8, 6))
plt.bar(range(1, 6), scores)
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title('Complement Naive Bayes Classifier Accuracy')
plt.show()

print(f'Average Accuracy of Complement Naive Bayes Classifier: {cnb_avg_accuracy:.2f}')
```




```
Average Accuracy of Complement Naive Bayes Classifier: 0.69
```