In [159]:
```
from google.colab import drive
drive.mount('/content/drive')

import sys
sys.path.append('drive/MyDrive/USC Spring 2024/Machine Learning/Pro
ject/')
```

```
Drive already mounted at /content/drive; to attempt to forcibly rem
ount, call drive.mount("/content/drive", force_remount=True).
```

In [156]:
```
import numpy as np
import pandas as pd
import pickle
import matplotlib.pyplot as plt
```

In [157]:
```
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.linear_model import Perceptron
from sklearn.neighbors import NeighborhoodComponentsAnalysis, KNeig
hborsClassifier
from sklearn.metrics import f1_score, ConfusionMatrixDisplay
```

In [4]:
```
def load_data(filename):
    #Loading the file into data
    data = np.array(pd.read_csv(filename))
    return data
```

In [5]:
```
drybeans_dataset_train_filename = 'drive/MyDrive/USC Spring 2024/Ma
chine Learning/Project/dry_bean_classification_train.csv'
drybeans_dataset_test_filename = 'drive/MyDrive/USC Spring 2024/Mac
hine Learning/Project/dry_bean_classification_test.csv'
```

In [6]:
```
drybeans_dataset_train = load_data(drybeans_dataset_train_filename)
#Splitting into X and y dataset
drybeans_dataset_train_X = drybeans_dataset_train[:, :-1]
drybeans_dataset_train_y = drybeans_dataset_train[:, -1]

drybeans_dataset_test = load_data(drybeans_dataset_test_filename)
drybeans_dataset_test_X = drybeans_dataset_test[:, :-1]
drybeans_dataset_test_y = drybeans_dataset_test[:, -1]
```

In [8]:
```
def standardize_training_dataset(dataset):
    dataset_copy = dataset.copy()
    means = []
    std_devs = []

    for i in range(dataset.shape[1]):
        column = dataset[:, i]
        mean = np.mean(column)
        std_dev = np.std(column)

        means.append(mean)
        std_devs.append(std_dev)

        dataset_copy[:, i] = (column-mean)/std_dev

    return means, std_devs, dataset_copy
```

In [9]:
```
def standardize_test_dataset(dataset, means, std_devs, ):
    dataset_test_copy = dataset.copy()

    for i in range(dataset.shape[1]):
        column = dataset[:, i]
        dataset_test_copy[:, i] = (column-means[i])/std_devs[i]

    return dataset_test_copy
```

In [10]:
```
def compute_test_accuracy(predicted, actual):
    return np.sum(predicted == actual)/len(actual)
```

In [11]:
```
def compute_F1_macro(actual, predicted):
    return f1_score(y_true=actual, y_pred=predicted, average='macr
o')
```

In [12]:
```
def compute_F1_weighted(actual, predicted):
    return f1_score(y_true=actual, y_pred=predicted, average='weigh
ted')
```

In [13]:
```
def create_confusion_matrix(model, dataset, labels):

    disp = ConfusionMatrixDisplay.from_estimator(
    model,
    dataset,
    drybeans_dataset_test_y,
    cmap=plt.cm.Blues,
    normalize=None)

    plt.figure(figsize=(20, 20))

    disp.ax_.set_title('Confusion Matrix')

    plt.show()
```

In [16]:
```
training_means, training_std_devs, standardized_drybeans_dataset_tr
ain_X = standardize_training_dataset(drybeans_dataset_train_X)
standardized_drybeans_dataset_test_X = standardize_test_dataset(dry
beans_dataset_test_X, training_means, training_std_devs)
```

2. Support Vector Classifier (SVC)

In [46]:
```
def model_selection_svc(C_range, gamma_range, dataset, labels, k):
    average_validation_classification_accuracy_C = {}

    for C in C_range:
        average_validation_classification_accuracy_C[C] = []

        for gamma in gamma_range:
            KF = KFold(n_splits=k)
            gamma_valid_accuracy = []

            for i, (train, valid) in enumerate(KF.split(dataset, la
bels)):
                svc_model = SVC(C=C, gamma=gamma)
                svc_model.fit(dataset[train], labels[train])
                valid_accuracy = svc_model.score(dataset[valid], la
bels[valid])

                gamma_valid_accuracy.append(valid_accuracy)

            average_validation_classification_accuracy_C[C].append
(np.mean(gamma_valid_accuracy))

    return average_validation_classification_accuracy_C
```

In [147]:
```
def find_best_parameter_svc(C_range, gamma_range, average_validatio
n_classification_accuracy):
    best_gamma = 0
    best_C = 0
    best_avg_valid_accuracy = float('-inf')

    for C in C_range:
        for i in range(len(average_validation_classification_accura
cy[C])):
            print('C =', C, ', Gamma =', gamma_range[i], ', Average 
Validation Accuracy =', average_validation_classification_accuracy
[C][i])
            if average_validation_classification_accuracy[C][i] >=
best_avg_valid_accuracy:
                best_avg_valid_accuracy = average_validation_classi
fication_accuracy[C][i]
                best_C = C
                best_gamma = gamma_range[i]

    return best_C, best_avg_valid_accuracy, best_gamma
```

In [48]:
```
def find_best_validation_accuracy(mean_validation_accuracy, C_val
s):
    best_mean_validation_accuracy_C = []

    for C in C_vals:
        best_mean_validation_accuracy_C.append(max(mean_validation_
accuracy[C]))
        best_index = np.argmin(mean_validation_accuracy[C])

    return best_mean_validation_accuracy_C
```

In [49]:
```
def plot_ave_valid_accuracy_vs_C(C_range, validation_accuracy):
    plt.plot(C_range, validation_accuracy)
    plt.title('Average Validation Accuracy vs C')
    plt.xlabel('C')
    plt.xscale('log')
    plt.ylabel('Average Validation Accuracy')
    plt.show()
```

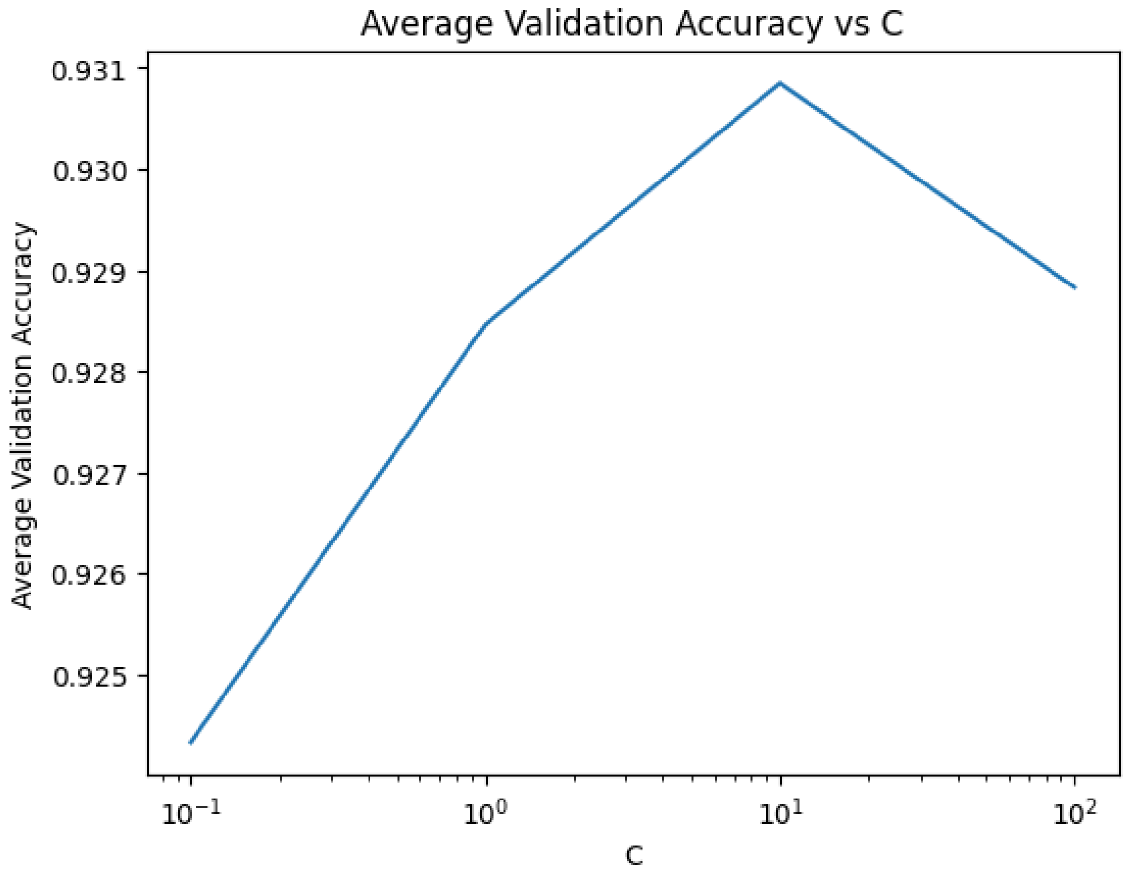
In [50]:
```
def plot_ave_valid_accuracy_vs_gamma(gamma_range, validation_accura
cy):
    plt.plot(gamma_range, validation_accuracy)
    plt.title('Average Validation Accuracy vs Gamma')
    plt.xlabel('Gamma')
    plt.xscale('log')
    plt.ylabel('Average Validation Accuracy')
    plt.show()
```

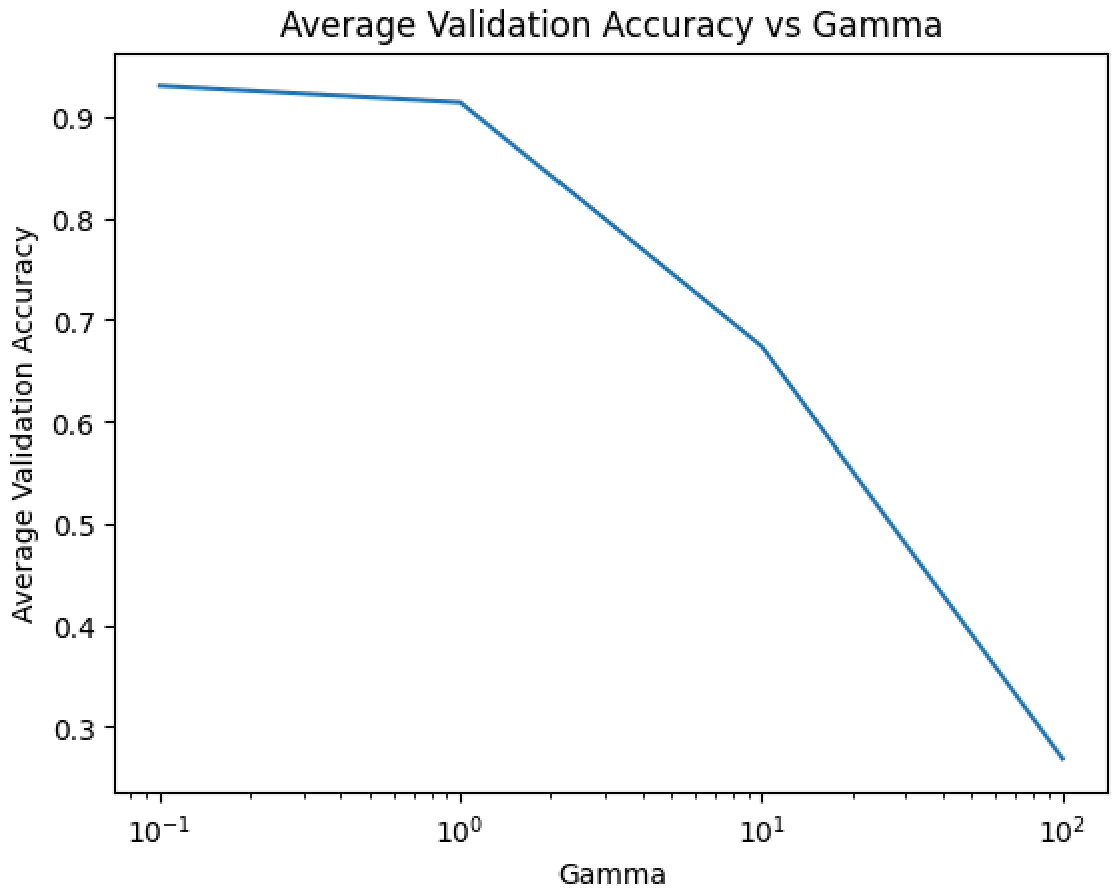
In [51]:
```
C_range = [0.1, 1, 10, 100]
gamma_range = [0.1, 1, 10, 100]
K = 5

average_validation_classification_accuracy_C = model_selection_svc
(C_range, gamma_range, standardized_drybeans_dataset_train_X, drybe
ans_dataset_train_y, K)
```

In [148]:
```
best_C, best_avg_valid_accuracy, best_gamma = find_best_parameter_s
vc(C_range, gamma_range, average_validation_classification_accuracy
_C)

print('Best gamma:', best_gamma)
print('Best C:', best_C)
print('Best Average Validation Accuracy:', best_avg_valid_accuracy)
```

```
C = 0.1 , Gamma = 0.1 , Average Validation Accuracy = 0.92432758705
7783
C = 0.1 , Gamma = 1 , Average Validation Accuracy = 0.8798788823635
36
C = 0.1 , Gamma = 10 , Average Validation Accuracy = 0.285518525126
8268
C = 0.1 , Gamma = 100 , Average Validation Accuracy = 0.25851891782
906106
C = 1 , Gamma = 0.1 , Average Validation Accuracy = 0.9284602402696
528
C = 1 , Gamma = 1 , Average Validation Accuracy = 0.919736092287977
6
C = 1 , Gamma = 10 , Average Validation Accuracy = 0.65460670090895
18
C = 1 , Gamma = 100 , Average Validation Accuracy = 0.2669679211626
011
C = 10 , Gamma = 0.1 , Average Validation Accuracy = 0.930847962651
5288
C = 10 , Gamma = 1 , Average Validation Accuracy = 0.91450062490693
87
C = 10 , Gamma = 10 , Average Validation Accuracy = 0.6739833293472
579
C = 10 , Gamma = 100 , Average Validation Accuracy = 0.268804637176
45825
C = 100 , Gamma = 0.1 , Average Validation Accuracy = 0.92882746536
64891
C = 100 , Gamma = 1 , Average Validation Accuracy = 0.8996229889828
253
C = 100 , Gamma = 10 , Average Validation Accuracy = 0.673983329347
258
C = 100 , Gamma = 100 , Average Validation Accuracy = 0.26880463717
645825
Best gamma: 0.1
Best C: 10
Best Average Validation Accuracy: 0.9308479626515288
```

In [53]:
```
best_mean_validation_accuracy_C = find_best_validation_accuracy(average_validation_classification_accuracy_C, C_range)
plot_ave_valid_accuracy_vs_C(C_range, best_mean_validation_accuracy_C)
plot_ave_valid_accuracy_vs_gamma(gamma_range, average_validation_classification_accuracy_C[best_C])
```


Average Validation Accuracy vs C
0.931
0.930
0.929
0.928
0.927
0.926
0.925
Average Validation Accuracy
10^{-1}
10^{0}
10^{1}
10^{2}
C


Average Validation Accuracy vs Gamma
0.9
0.8
0.7
0.6
0.5
0.4
0.3
Average Validation Accuracy
10^{-1}
10^{0}
10^{1}
10^{2}
Gamma

In [54]:
```
SVC_model_final = SVC(C=best_C, gamma=best_gamma)
SVC_model_final.fit(standardized_drybeans_dataset_train_X, drybeans
_dataset_train_y)
```

Out[54]:
```
▾          SVC
SVC(C=10, gamma=0.1)
```

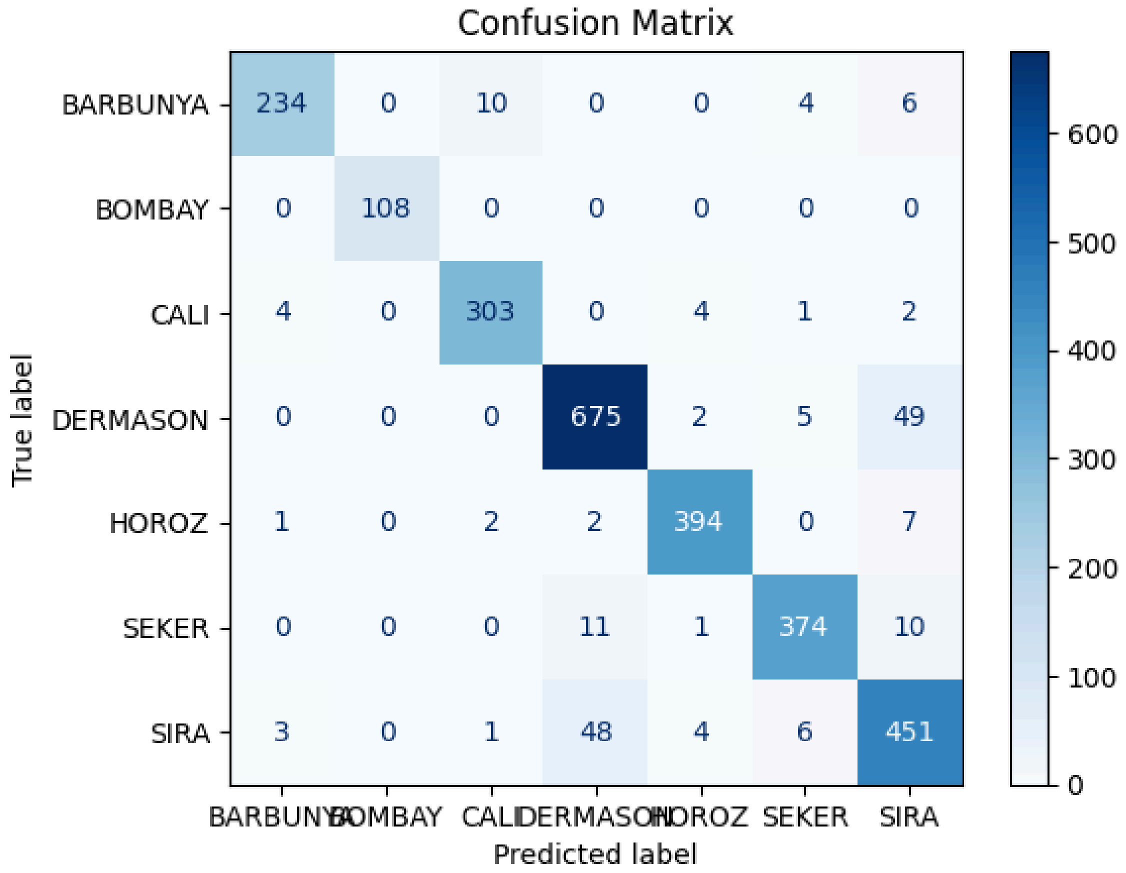
In [55]:
```
predictions = SVC_model_final.predict(standardized_drybeans_dataset
_test_X)
test_accuracy_final_model = compute_test_accuracy(predictions, dryb
eans_dataset_test_y)
print('Classification Test Accuracy:', test_accuracy_final_model*10
0, '%')
```

```
Classification Test Accuracy: 93.27700220426158 %
```

In [56]:
```
F1_macro_SVC = compute_F1_macro(drybeans_dataset_test_y, prediction
s)
F1_weighted_SVC = compute_F1_weighted(drybeans_dataset_test_y, pred
ictions)

print('Macro-Averaged F1 Score =', F1_macro_SVC)
print('Weighted-Averaged F1 Score =', F1_weighted_SVC)
```

```
Macro-Averaged F1 Score = 0.945424463981152
Weighted-Averaged F1 Score = 0.9329631451281314
```

In [57]:
```
create_confusion_matrix(SVC_model_final, standardized_drybeans_data
set_test_X, drybeans_dataset_test_y)
```




```
<Figure size 2000x2000 with 0 Axes>
```

3. Random Forest Classifier

In [149]:
```
def model_selection_rfc(max_depth_range, tree_range, dataset, label
s, k):
    average_validation_classification_accuracy_tree = {}

    for tree in tree_range:
        average_validation_classification_accuracy_tree[tree] = []

        for max_depth in max_depth_range:
            KF = KFold(n_splits=k)
            depth_valid_accuracy = []

            for i, (train, valid) in enumerate(KF.split(dataset, la
bels)):
                rfc_model = RandomForestClassifier(n_estimators=tre
e, max_depth=max_depth)
                rfc_model.fit(dataset[train], labels[train])
                valid_accuracy = rfc_model.score(dataset[valid], la
bels[valid])

                depth_valid_accuracy.append(valid_accuracy)

            average_validation_classification_accuracy_tree[tree].a
ppend(np.mean(depth_valid_accuracy))

    return average_validation_classification_accuracy_tree
```

In [163]:
```
def find_best_parameter_rf(max_depth_range, tree_range, average_val
idation_classification_accuracy):
    best_max_depth = 0
    best_tree = 0
    best_avg_valid_accuracy = float('-inf')

    for tree in tree_range:
        for i in range(len(average_validation_classification_accura
cy[tree])):
            print('Tree =', tree, ', Max Depth =', max_depth_range
[i], ', Average Validation Accuracy =', average_validation_classifi
cation_accuracy[tree][i])
            if average_validation_classification_accuracy[tree][i]
>= best_avg_valid_accuracy:
                best_avg_valid_accuracy = average_validation_classi
fication_accuracy[tree][i]
                best_tree = tree
                best_max_depth = max_depth_range[i]

    return best_tree, best_avg_valid_accuracy, best_max_depth
```

In [151]:
```
def find_best_validation_accuracy(mean_validation_accuracy, tree_va
ls):
    best_mean_validation_accuracy_tree = []

    for tree in tree_vals:
        best_mean_validation_accuracy_tree.append(max(mean_validati
on_accuracy[tree]))
        best_index = np.argmin(mean_validation_accuracy[tree])

    return best_mean_validation_accuracy_tree
```

In [152]:
```
def plot_ave_valid_accuracy_vs_tree(tree_range, validation_accurac
y):
    plt.plot(tree_range, validation_accuracy)
    plt.title('Average Validation Accuracy vs Number of Trees')
    plt.xlabel('Number of Trees')
    plt.xscale('log')
    plt.ylabel('Average Validation Accuracy')
    plt.show()
```

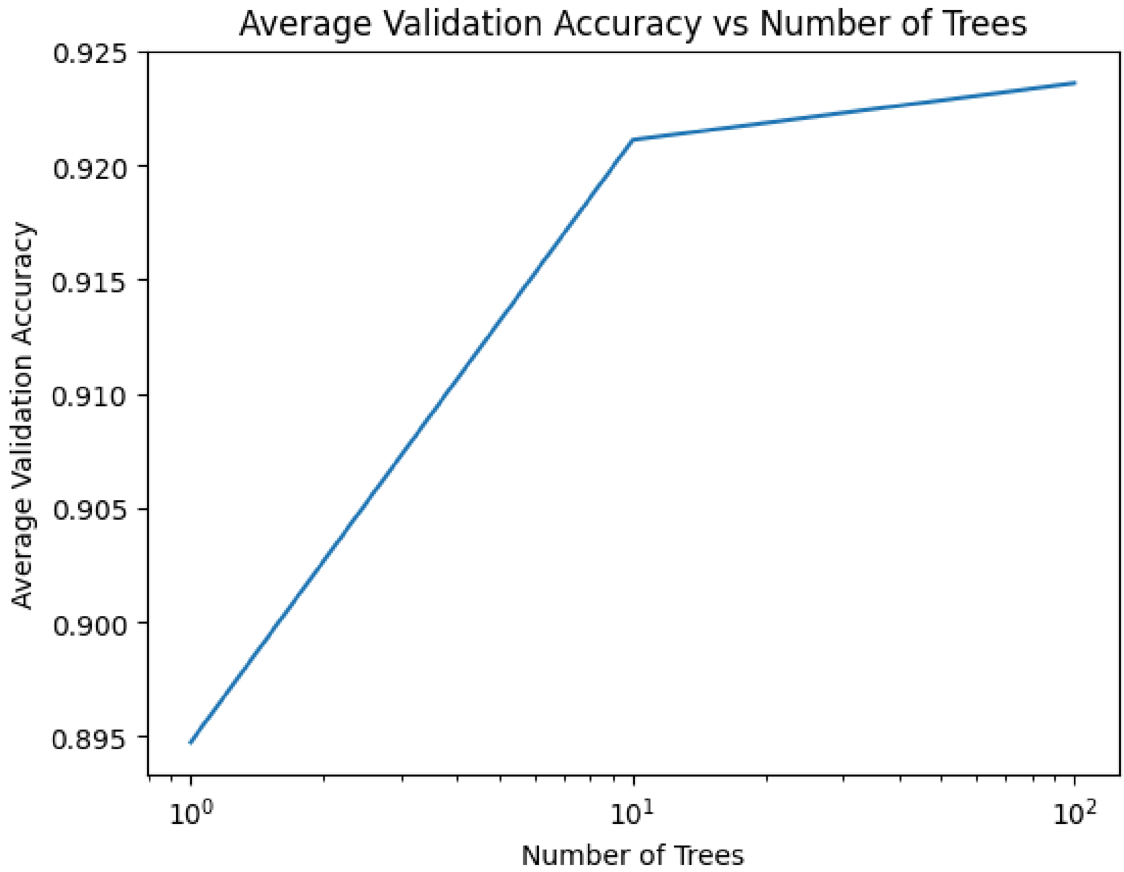
In [153]:
```
def plot_ave_valid_accuracy_vs_max_depth(max_depth_range, validatio
n_accuracy):
    plt.plot(max_depth_range, validation_accuracy)
    plt.title('Average Validation Accuracy vs Max Depth')
    plt.xlabel('Max Depth')
    plt.xscale('log')
    plt.ylabel('Average Validation Accuracy')
    plt.show()
```

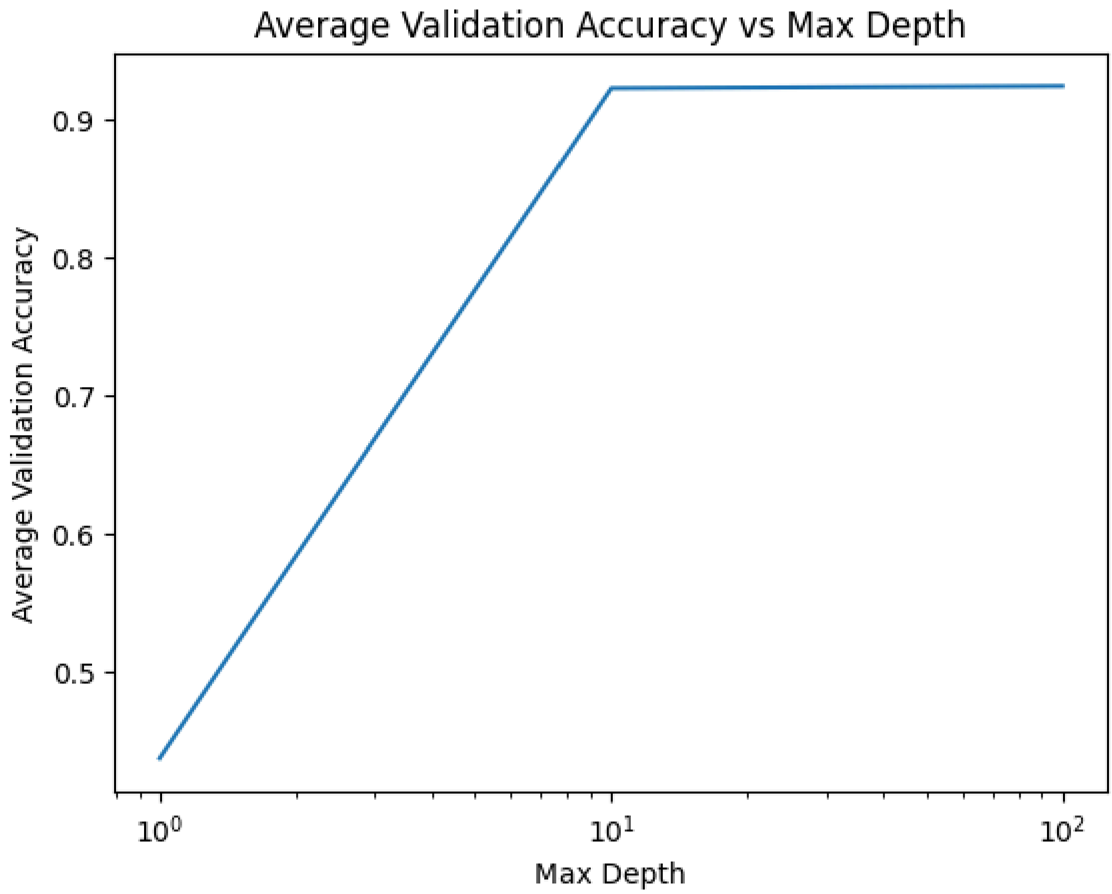
In [161]:
```
tree_range = [1, 10, 100]
max_depth_range = [1, 10, 100]
K = 5

average_validation_classification_accuracy_tree = model_selection_r
fc(max_depth_range, tree_range, standardized_drybeans_dataset_train
_X, drybeans_dataset_train_y, K)
```

In [164]:
```
best_tree, best_avg_valid_accuracy, best_max_depth = find_best_para
meter_rf(max_depth_range, tree_range, average_validation_classifica
tion_accuracy_tree)

print('Best Number of Trees:', best_tree)
print('Best Max Depth:', best_max_depth)
print('Best Average Validation Accuracy', best_avg_valid_accuracy)
```

```
Tree = 1 , Max Depth = 1 , Average Validation Accuracy = 0.40554697
17849139
Tree = 1 , Max Depth = 10 , Average Validation Accuracy = 0.8947570
66636634
Tree = 1 , Max Depth = 100 , Average Validation Accuracy = 0.884103
1098558139
Tree = 10 , Max Depth = 1 , Average Validation Accuracy = 0.4992138
362790219
Tree = 10 , Max Depth = 10 , Average Validation Accuracy = 0.921113
418394915
Tree = 10 , Max Depth = 100 , Average Validation Accuracy = 0.91560
35656181812
Tree = 100 , Max Depth = 1 , Average Validation Accuracy = 0.437136
302263458
Tree = 100 , Max Depth = 10 , Average Validation Accuracy = 0.92203
17764018435
Tree = 100 , Max Depth = 100 , Average Validation Accuracy = 0.9235
932634061836
Best Number of Trees: 100
Best Max Depth: 100
Best Average Validation Accuracy 0.9235932634061836
```

In [165]:
```
best_mean_validation_accuracy_tree = find_best_validation_accuracy
(average_validation_classification_accuracy_tree, tree_range)
plot_ave_valid_accuracy_vs_tree(tree_range, best_mean_validation_ac
curacy_tree)
plot_ave_valid_accuracy_vs_max_depth(max_depth_range, average_valid
ation_classification_accuracy_tree[best_tree])
```


Average Validation Accuracy vs Number of Trees
Average Validation Accuracy
0.925
0.920
0.915
0.910
0.905
0.900
0.895
10^0
10^1
10^2
Number of Trees


Average Validation Accuracy vs Max Depth
Average Validation Accuracy
0.9
0.8
0.7
0.6
0.5
10^0
10^1
10^2
Max Depth

In [166]:
```
rf_model_final = RandomForestClassifier(n_estimators=best_tree, max
_depth=best_max_depth)
rf_model_final.fit(standardized_drybeans_dataset_train_X, drybeans_
dataset_train_y)
```

Out[166]:
```
▾       RandomForestClassifier

RandomForestClassifier(max_depth=100)
```

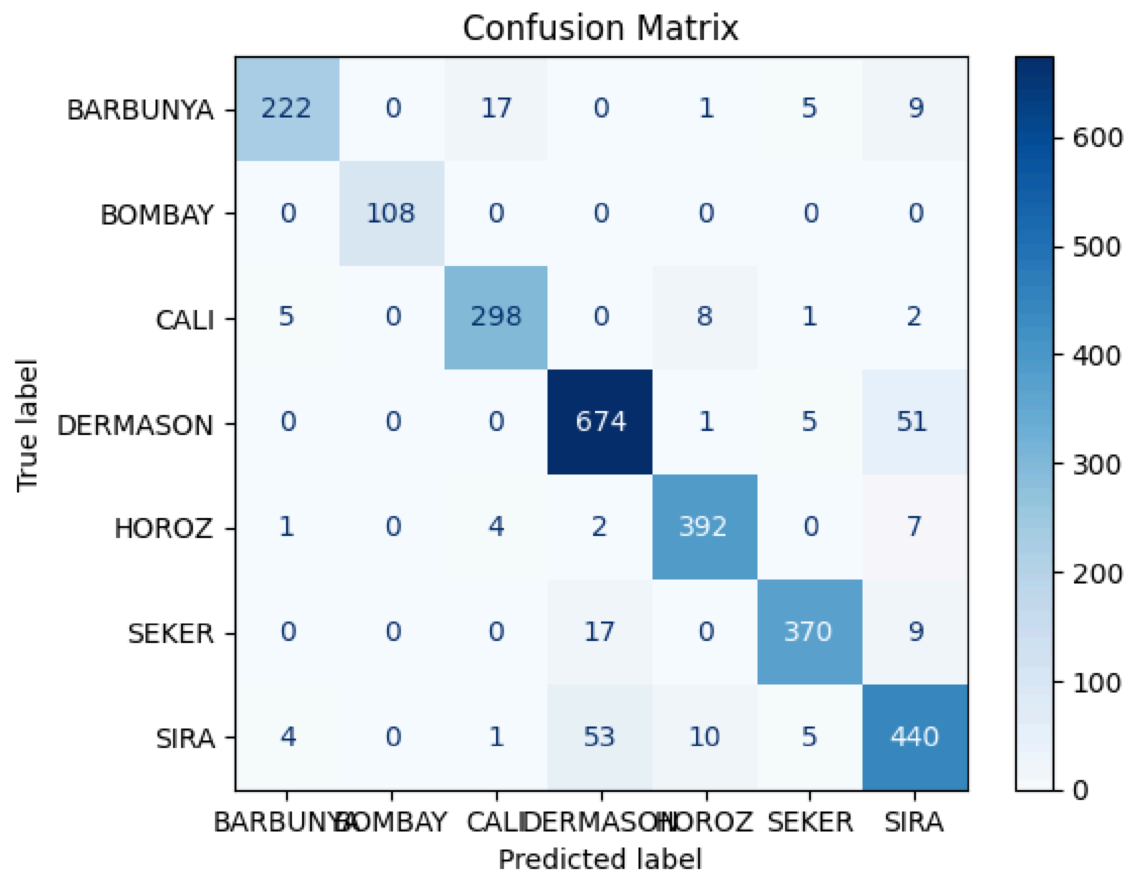
In [167]:
```
predictions = rf_model_final.predict(standardized_drybeans_dataset_
test_X)
test_accuracy_rf = compute_test_accuracy(drybeans_dataset_test_y, p
redictions)
print('Classification Test Accuracy:', test_accuracy_rf*100, '%')
```

```
Classification Test Accuracy: 91.9911829537105 %
```

In [168]:
```
F1_macro_RF = compute_F1_macro(drybeans_dataset_test_y, prediction
s)
F1_weighted_RF = compute_F1_weighted(drybeans_dataset_test_y, predi
ctions)

print('Macro-Averaged F1 Score =', F1_macro_RF)
print('Weighted-Averaged F1 Score =', F1_weighted_RF)
```

```
Macro-Averaged F1 Score = 0.9320815939173662
Weighted-Averaged F1 Score = 0.9199509695063591
```

In [169]:
```
create_confusion_matrix(rf_model_final, standardized_drybeans_datas
et_test_X, drybeans_dataset_test_y)
```




```
<Figure size 2000x2000 with 0 Axes>
```

4. KNN Classifier

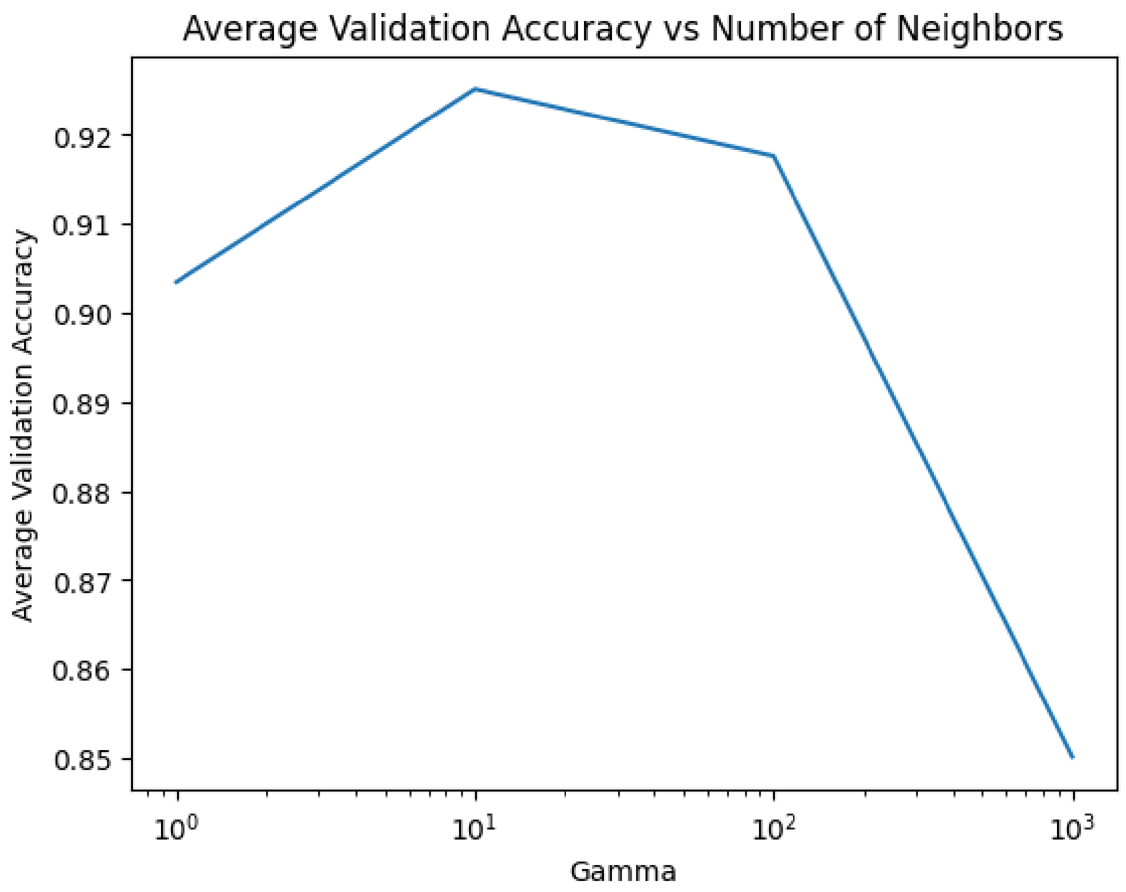
In [34]:
```
def model_selection_knn(neighbor_range, dataset, labels, k):
    best_neighbor = 0
    average_validation_classification_accuracy = []

    for neighbor in neighbor_range:
        KF = KFold(n_splits=k)
        neighbor_valid_accuracy = []

        for i, (train, valid) in enumerate(KF.split(dataset, label
s)):

            knn_model = KNeighborsClassifier(n_neighbors=neighbor)

            knn_model.fit(dataset[train], labels[train])

            valid_accuracy = knn_model.score(dataset[valid], labels
[valid])

            neighbor_valid_accuracy.append(valid_accuracy)

        average_validation_classification_accuracy.append(np.mean(n
eighbor_valid_accuracy))

    for i in range(len(average_validation_classification_accurac
y)):
        print('Number of Neighbors =', neighbor_range[i], ', Averag
e Validation RMSE =', average_validation_classification_accuracy
[i])

    best_neighbor_index = np.argmax(average_validation_classificati
on_accuracy)
    best_neighbor = neighbor_range[best_neighbor_index]
    best_average_validation_accuracy = average_validation_classific
ation_accuracy[best_neighbor_index]

    return best_neighbor, average_validation_classification_accurac
y, best_average_validation_accuracy
```

In [15]:
```
def plot_ave_valid_accuracy_vs_neighbors(neighbor_range, validation
_accuracy):
    plt.plot(neighbor_range, validation_accuracy)
    plt.title('Average Validation Accuracy vs Number of Neighbors')
    plt.xlabel('Gamma')
    plt.xscale('log')
    plt.ylabel('Average Validation Accuracy')
    plt.show()
```

In [35]:
```
neighbor_range = [1, 10, 100, 1000]
K = 5

best_neighbor, average_validation_classification_accuracy, best_ave
rage_validation_accuracy = model_selection_knn(neighbor_range, stan
dardized_drybeans_dataset_train_X, drybeans_dataset_train_y, K)
print('Best number of neighbors:', best_neighbor)
print('Best average validation accuracy:', best_average_validation_
accuracy)
```

```
Number of Neighbors = 1 , Average Validation RMSE = 0.9034802444624
134
Number of Neighbors = 10 , Average Validation RMSE = 0.925154328603
6125
Number of Neighbors = 100 , Average Validation RMSE = 0.91762372545
76922
Number of Neighbors = 1000 , Average Validation RMSE = 0.8501239057
801466
Best number of neighbors: 10
Best average validation accuracy: 0.9251543286036125
```

In [36]:
```
plot_ave_valid_accuracy_vs_neighbors(neighbor_range, average_valida
tion_classification_accuracy)
```



In [58]:
```
knn_model_final = KNeighborsClassifier(n_neighbors=best_neighbor)
knn_model_final.fit(standardized_drybeans_dataset_train_X, drybeans
_dataset_train_y)
```

Out[58]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10)
```

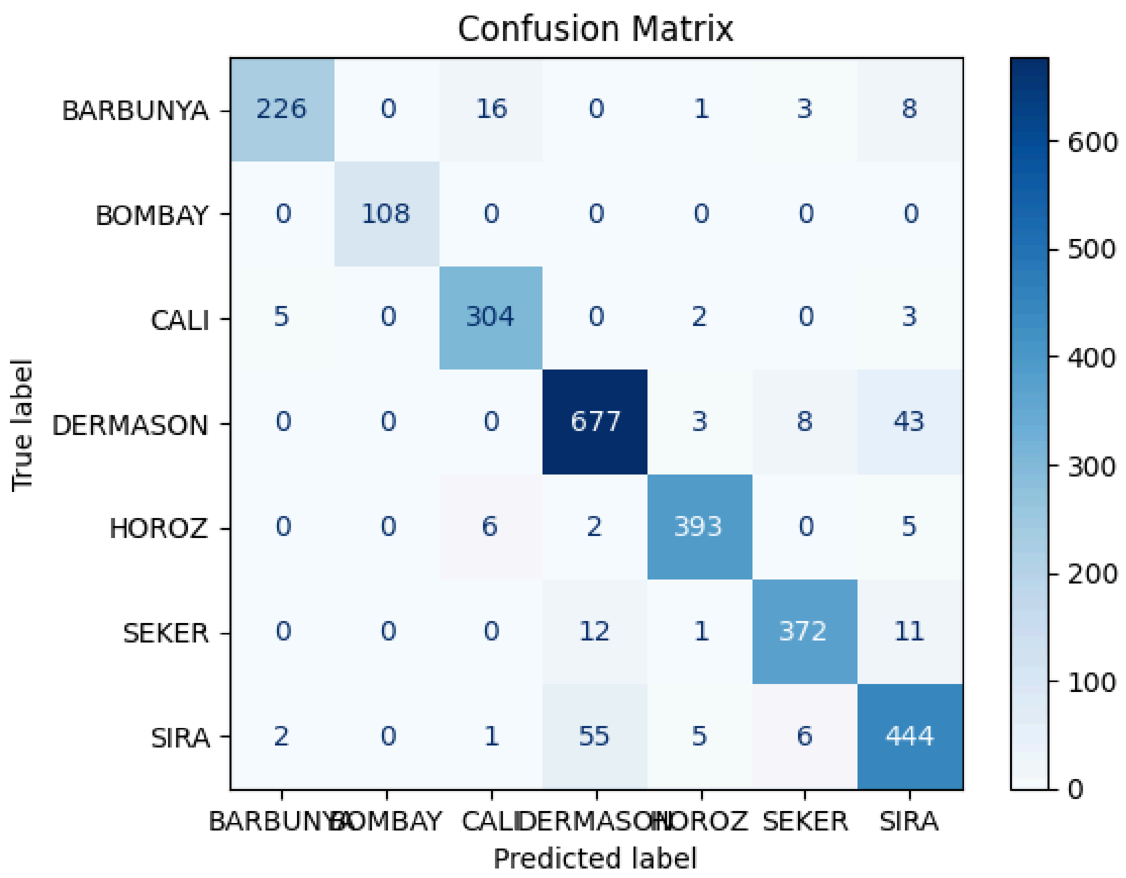
In [59]:
```
predictions = knn_model_final.predict(standardized_drybeans_dataset
_test_X)
test_accuracy_knn = compute_test_accuracy(drybeans_dataset_test_y,
predictions)
print('Classification Test Accuracy:', test_accuracy_knn*100, '%')
```

```
Classification Test Accuracy: 92.72593681116827 %
```

In [60]:
```
F1_macro_KNN = compute_F1_macro(drybeans_dataset_test_y, prediction
s)
F1_weighted_KNN = compute_F1_weighted(drybeans_dataset_test_y, pred
ictions)

print('Macro-Averaged F1 Score =', F1_macro_KNN)
print('Weighted-Averaged F1 Score =', F1_weighted_KNN)
```

```
Macro-Averaged F1 Score = 0.9392816197104452
Weighted-Averaged F1 Score = 0.9272865713896141
```

In [45]:
```
create_confusion_matrix(knn_model_final, standardized_drybeans_data
set_test_X, drybeans_dataset_test_y)
```




```
<Figure size 2000x2000 with 0 Axes>
```

5. Perceptron

In [118]:

```
def model_selection_perceptron(dataset, labels, k, eta_range, alpha_range):
    average_validation_classification_accuracy_alpha = {}

    for alpha in alpha_range:
        average_validation_classification_accuracy_alpha[alpha] = []
        for eta in eta_range:
            KF = KFold(n_splits=k)
            eta_valid_accuracy = []

            for i, (train, valid) in enumerate(KF.split(dataset, labels)):
                perceptron_model = Perceptron(penalty='l2', alpha=alpha, eta0=eta)
                perceptron_model.fit(dataset[train], labels[train])
                valid_accuracy = perceptron_model.score(dataset[valid], labels[valid])

                eta_valid_accuracy.append(valid_accuracy)

            average_validation_classification_accuracy_alpha[alpha].append(np.mean(eta_valid_accuracy))

    return average_validation_classification_accuracy_alpha
```

In [145]:

```
def find_best_parameter_perceptron(alpha_range, eta_range, average_validation_classification_accuracy):
    best_alpha = 0
    best_eta = 0
    best_avg_valid_accuracy = float('-inf')

    for alpha in alpha_range:
        for i in range(len(average_validation_classification_accuracy[alpha])):
            print('Alpha =', alpha, ', Eta =', eta_range[i], ', Average Validation Accuracy =', average_validation_classification_accuracy[alpha][i])
            if average_validation_classification_accuracy[alpha][i] >= best_avg_valid_accuracy:
                best_avg_valid_accuracy = average_validation_classification_accuracy[alpha][i]
                best_alpha = alpha
                best_eta = eta_range[i]

    return best_alpha, best_avg_valid_accuracy, best_eta
```

In [65]:
```
def find_best_validation_accuracy(mean_validation_accuracy, alpha_v
als):
    best_mean_validation_accuracy_alpha = []

    for alpha in alpha_vals:
        best_mean_validation_accuracy_alpha.append(max(mean_validat
ion_accuracy[alpha]))
        best_index = np.argmax(mean_validation_accuracy[alpha])

    return best_mean_validation_accuracy_alpha
```

In [66]:
```
def plot_ave_valid_accuracy_vs_alpha(alpha_range, validation_accura
cy):
    plt.plot(alpha_range, validation_accuracy)
    plt.title('Average Validation Accuracy vs Alpha')
    plt.xlabel('Alpha')
    plt.xscale('log')
    plt.ylabel('Average Validation Accuracy')
    plt.show()
```

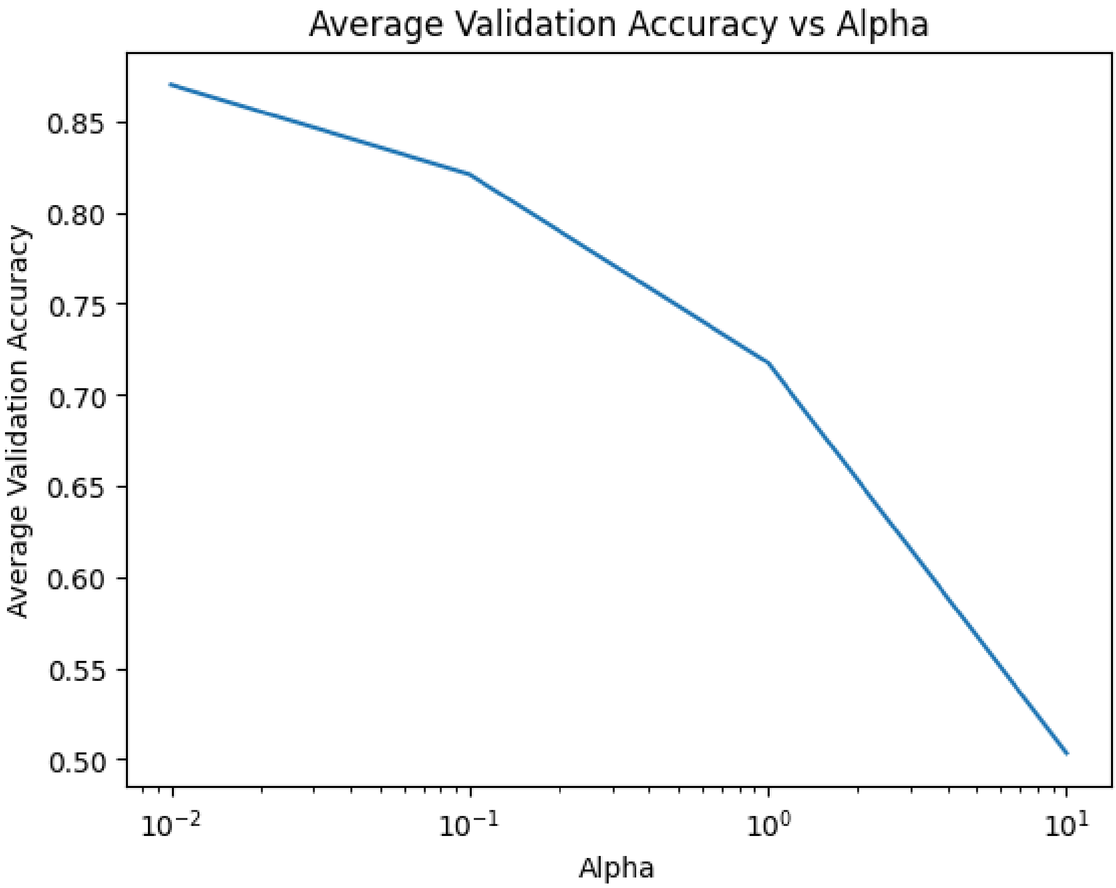
In [67]:
```
def plot_ave_valid_accuracy_vs_eta(eta_range, validation_accuracy):
    plt.plot(eta_range, validation_accuracy)
    plt.title('Average Validation Accuracy vs Eta')
    plt.xlabel('Eta')
    plt.xscale('log')
    plt.ylabel('Average Validation Accuracy')
    plt.show()
```

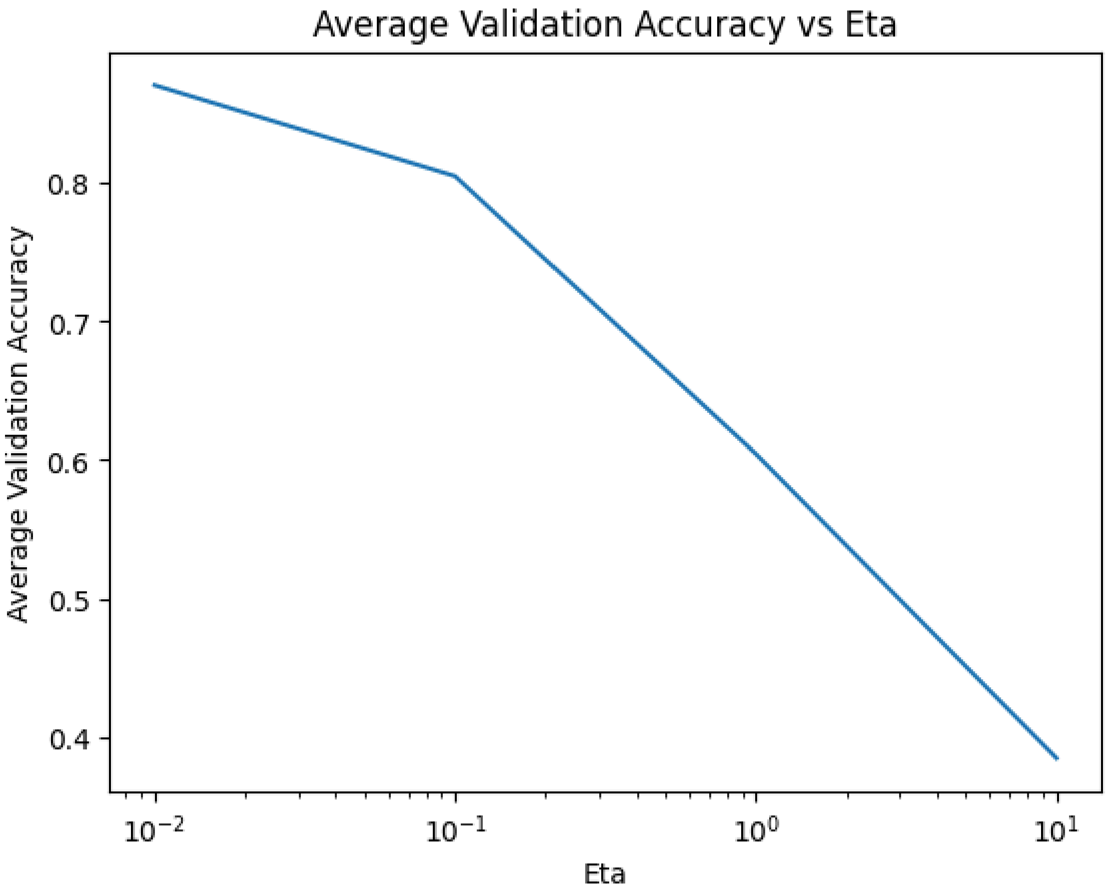
In [120]:
```
alpha_range = [0.01, 0.1, 1, 10]
eta_range = [0.01, 0.1, 1, 10]
K = 5

average_validation_classification_accuracy_alpha = model_selection_
perceptron(standardized_drybeans_dataset_train_X, drybeans_dataset_
train_y, K, eta_range, alpha_range)
```

In [146]:
```
best_alpha, best_avg_valid_accuracy, best_eta = find_best_parameter
_perceptron(alpha_range, eta_range, average_validation_classificati
on_accuracy_alpha)

print('Best alpha:', best_alpha)
print('Best eta:', best_eta)
print('Best Average Validation Accuracy', best_avg_valid_accuracy)
```

```
Alpha = 0.01 , Eta = 0.01 , Average Validation Accuracy = 0.8703294
691602202
Alpha = 0.01 , Eta = 0.1 , Average Validation Accuracy = 0.80465573
59623714
Alpha = 0.01 , Eta = 1 , Average Validation Accuracy = 0.6042837866
281304
Alpha = 0.01 , Eta = 10 , Average Validation Accuracy = 0.384872359
1196553
Alpha = 0.1 , Eta = 0.01 , Average Validation Accuracy = 0.82110434
95463256
Alpha = 0.1 , Eta = 0.1 , Average Validation Accuracy = 0.614745104
1926342
Alpha = 0.1 , Eta = 1 , Average Validation Accuracy = 0.38891883717
957965
Alpha = 0.1 , Eta = 10 , Average Validation Accuracy = 0.1025852967
3905295
Alpha = 1 , Eta = 0.01 , Average Validation Accuracy = 0.7176968245
953923
Alpha = 1 , Eta = 0.1 , Average Validation Accuracy = 0.38745010551
49988
Alpha = 1 , Eta = 1 , Average Validation Accuracy = 0.1025852967390
5295
Alpha = 1 , Eta = 10 , Average Validation Accuracy = 0.102585296739
05295
Alpha = 10 , Eta = 0.01 , Average Validation Accuracy = 0.503346742
5750384
Alpha = 10 , Eta = 0.1 , Average Validation Accuracy = 0.1259161118
8512677
Alpha = 10 , Eta = 1 , Average Validation Accuracy = 0.102585296739
05295
Alpha = 10 , Eta = 10 , Average Validation Accuracy = 0.10258529673
905295
Best alpha: 0.01
Best eta: 0.01
Best Average Validation Accuracy 0.8703294691602202
```

In [122]:
```
best_mean_validation_accuracy_alpha = find_best_validation_accuracy
(average_validation_classification_accuracy_alpha, alpha_range)
plot_ave_valid_accuracy_vs_alpha(alpha_range, best_mean_validation_
accuracy_alpha)
plot_ave_valid_accuracy_vs_eta(eta_range, average_validation_classi
fication_accuracy_alpha[best_alpha])
```


Average Validation Accuracy vs Alpha
Average Validation Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
10^{-2}
10^{-1}
10^{0}
10^{1}
Alpha


Average Validation Accuracy vs Eta
Average Validation Accuracy
0.8
0.7
0.6
0.5
0.4
10^{-2}
10^{-1}
10^{0}
10^{1}
Eta

In [131]:
```
perceptron_model_final = Perceptron(alpha=best_alpha, eta0=best_eta)
perceptron_model_final.fit(standardized_drybeans_dataset_train_X, drybeans_dataset_train_y)
```

Out[131]:
```
▾ Perceptron
Perceptron(alpha=0.01, eta0=0.01)
```

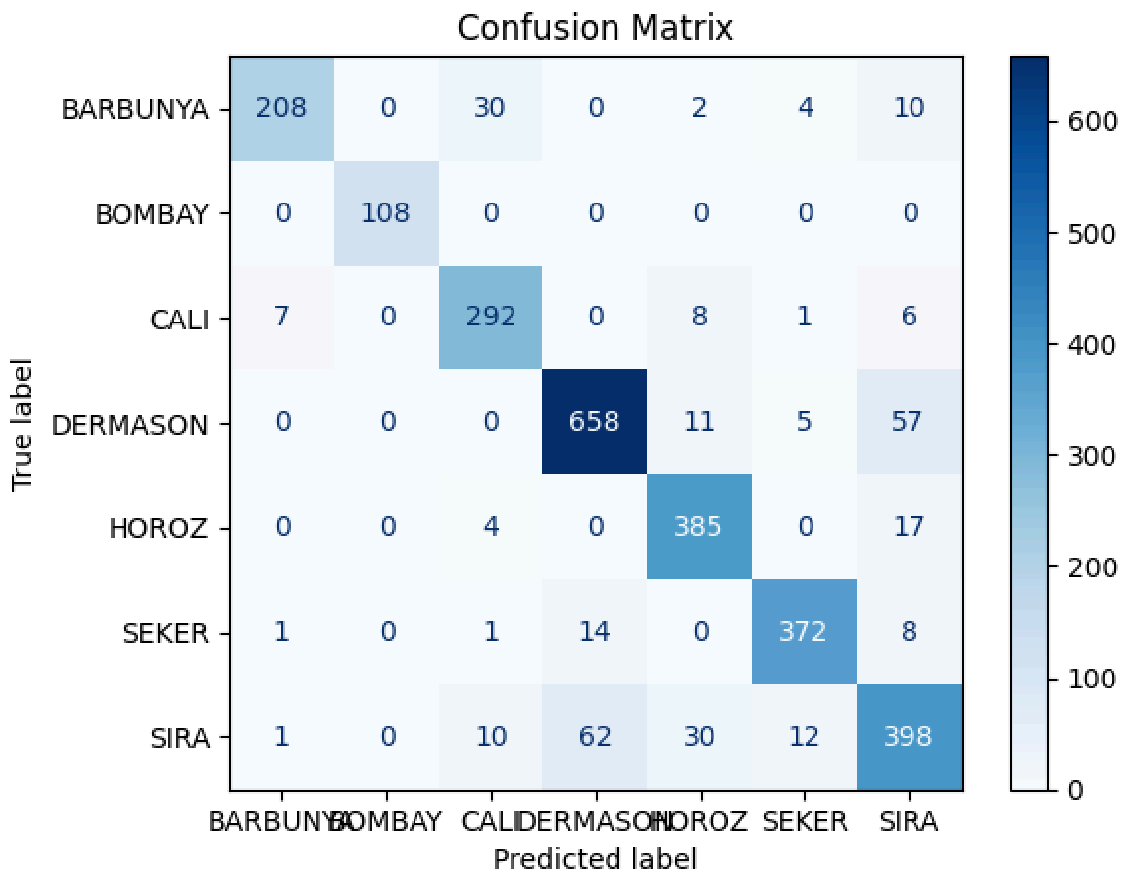
In [132]:
```
predictions = perceptron_model_final.predict(standardized_drybeans_dataset_test_X)
test_accuracy_perceptron = compute_test_accuracy(drybeans_dataset_test_y, predictions)
print('Classification Test Accuracy:', test_accuracy_perceptron*100, '%')
```

```
Classification Test Accuracy: 88.94195444526083 %
```

In [133]:
```
F1_macro_perceptron = compute_F1_macro(drybeans_dataset_test_y, predictions)
F1_weighted_perceptron = compute_F1_weighted(drybeans_dataset_test_y, predictions)

print('Macro-Averaged F1 Score =', F1_macro_perceptron)
print('Weighted-Averaged F1 Score =', F1_weighted_perceptron)
```

```
Macro-Averaged F1 Score = 0.9033947812491739
Weighted-Averaged F1 Score = 0.8889073871701505
```

In [134]:
```
create_confusion_matrix(perceptron_model_final, standardized_drybea
ns_dataset_test_X, drybeans_dataset_test_y)
```




```
<Figure size 2000x2000 with 0 Axes>
```