Multiclass Classification

Hagar Hesham Kamel Ismail Negm

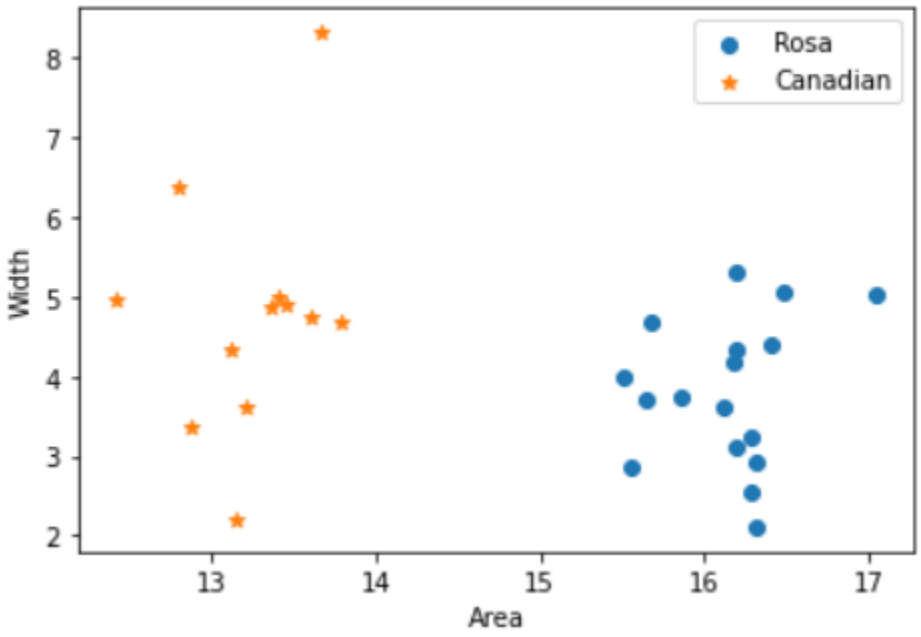
Ekhlas Soliman Khlil Mosa

1 DATASET

Our dataset consists of two features – area and width – and 3 classes – Kama, Rosa and Canadian.

2 BINARY CLASSIFICATION

The Kama class is initially removed from the dataset, so that we can compare the performance of an SVM model and a Perceptron model on the binary classification of Rosa and Canadian.



The test data is found to be linearly separable.

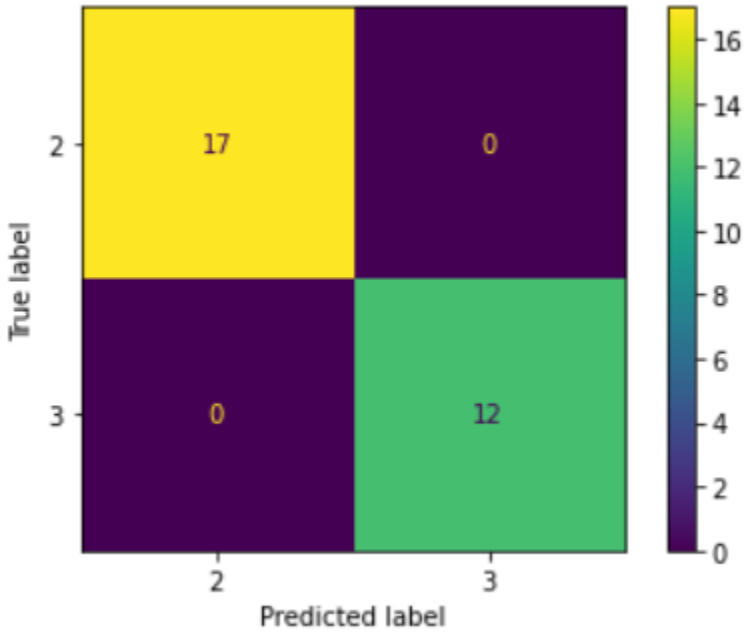
2.1 SVM CLASSIFIER

```
svm_model = svm.SVC(kernel = "linear", random_state = 42)
svm_model.fit(binary_features_train, binary_labels_train)

binary_label_pred = svm_model.predict(binary_features_test)
svm_accuracy = svm_model.score(binary_features_test, binary_labels_test)
print(svm_accuracy)
print(classification_report(binary_labels_test, binary_label_pred))
svm_confusion_matrix = confusion_matrix(binary_labels_test, binary_label_pred)
ConfusionMatrixDisplay(svm_confusion_matrix, display_labels = svm_model.classes_).plot()
```

The SVM model achieved an accuracy of **100%** since the test data is very easily linearly separable.

	precision	recall	f1-score	support
2	1.00	1.00	1.00	17
3	1.00	1.00	1.00	12
accuracy			1.00	29
macro avg	1.00	1.00	1.00	29
weighted avg	1.00	1.00	1.00	29

Confusion Matrix:



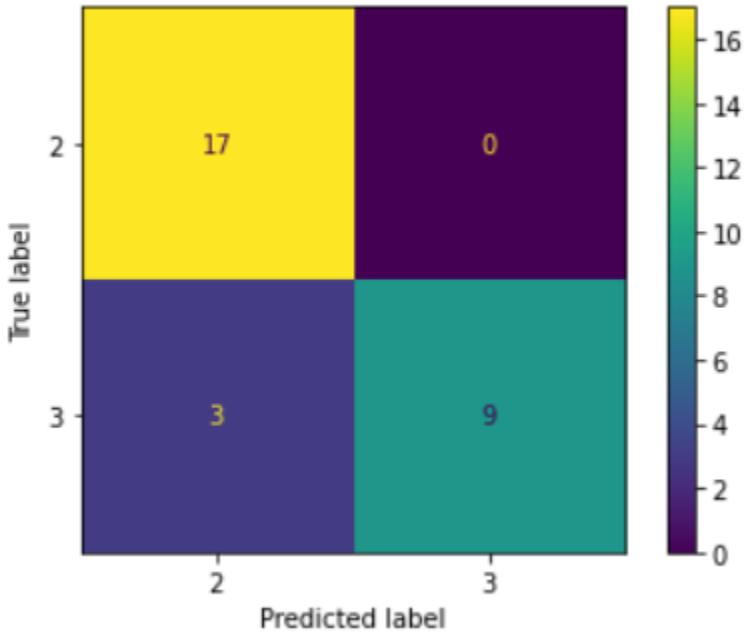
2.2 PERCEPTRON CLASSIFIER

```
percptron_model = Perceptron(penalty="elasticnet", alpha=0.001, class_weight="balanced", random_state=42)
percptron_model.fit(binary_features_train, binary_labels_train)

binary_label_pred = percptron_model.predict(binary_features_test)
perceptron_accuracy = percptron_model.score(binary_features_test, binary_labels_test)
print(perceptron_accuracy)
print(classification_report(binary_labels_test, binary_label_pred))
perceptron_confusion_matrix = confusion_matrix(binary_labels_test, binary_label_pred)
ConfusionMatrixDisplay(perceptron_confusion_matrix, display_labels = percptron_model.classes_).plot()
```

The Perceptron achieved an accuracy of **90%**. The classifier misclassified three instances of the Canadian class as Rosa.

	precision	recall	f1-score	support
2	0.85	1.00	0.92	17
3	1.00	0.75	0.86	12
accuracy			0.90	29
macro avg	0.93	0.88	0.89	29
weighted avg	0.91	0.90	0.89	29

Confusion Matrix:



Comparison:

The SVM performed better than the Perceptron in finding a decision boundary that linearly separated the two classes perfectly. We tried adjusting the hyperparameters of the Perceptron model, but it still misclassified three instances of the Canadian class. The SVM was better at generalizing to new unseen instances than the Perceptron.

3 ONE-VERSUS-REST

```
def binarize(cls, labels):

    label_bin = labels.copy()
    label_bin.loc[labels["label"] == cls, "label"] = 1
    label_bin.loc[labels["label"]  != cls, "label"] = -1

    return label_bin
```

```
def model_building(train_features, train_labels, model, cls):

    model = model.fit(train_features, train_labels["label"].tolist())

    return model
```

```
def model_performance(test_features, test_labels, model, cls):

    predictions = model.predict(test_features)
    pred_prob =  model.predict_proba(test_features)
    accuracy = model.score(test_features, test_labels["label"].tolist())
    print(accuracy)

    print(classification_report(test_labels["label"].tolist(), predictions))
    conf_mat = confusion_matrix(test_labels["label"].tolist(), predictions)
    ConfusionMatrixDisplay(conf_mat, display_labels = model.classes_).plot()
    plt.figure()

    return predictions, pred_prob,
```

```
def model_plotting(test_features, test_labels, model, cls):

    plotData(np.array(test_features), test_labels["label"], class_names, cls)
    plotRegions(model, np.array(test_features))
    plt.figure()
```

3.1 OvR-SVM

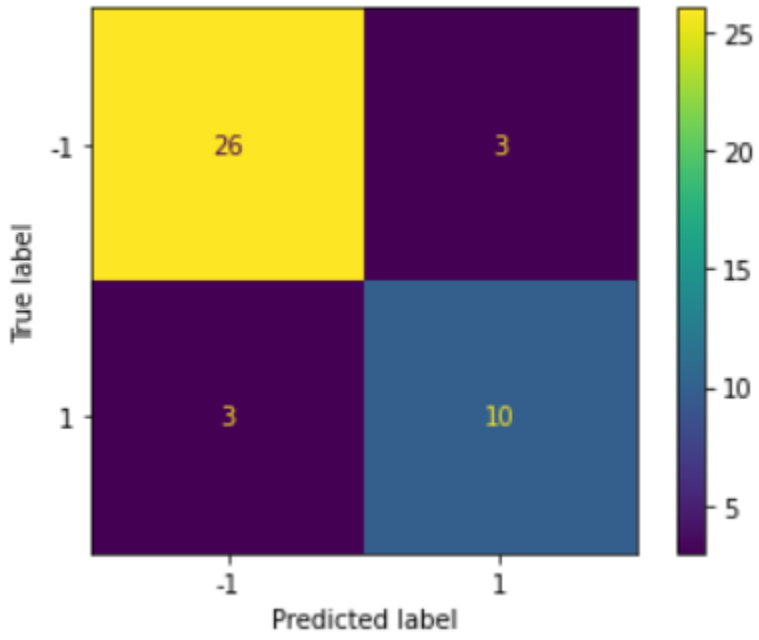
```
svm_model = svm.SVC(kernel = "poly", probability = True, random_state = 42)
svm_prediction_probs = []
svm_predictions = []

for cls in classes:
    train_label_bin = binarize(cls, train_labels)
    test_label_bin = binarize(cls, test_labels)

    model = model_building(train_features, train_label_bin, svm_model, cls)
    predictions, prediction_probs = model_performance(test_features, test_label_bin, model, cls)
    svm_prediction_probs.append(prediction_probs)
    svm_predictions.append(predictions)
    model_plotting(test_features, test_label_bin, model, cls)
```

3.1.1 SVM First Classifier (Kama-versus-Rest)

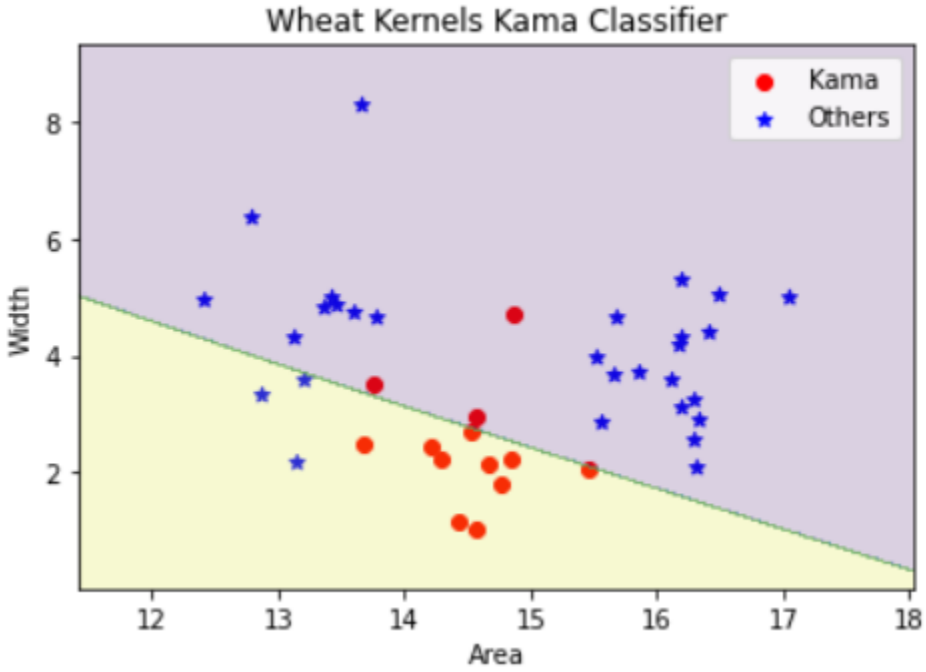
Accuracy: 85.71428571428571

	precision	recall	f1-score	support
-1	0.90	0.90	0.90	29
1	0.77	0.77	0.77	13
accuracy			0.86	42
macro avg	0.83	0.83	0.83	42
weighted avg	0.86	0.86	0.86	42

Confusion Matrix:



Decision Boundary:

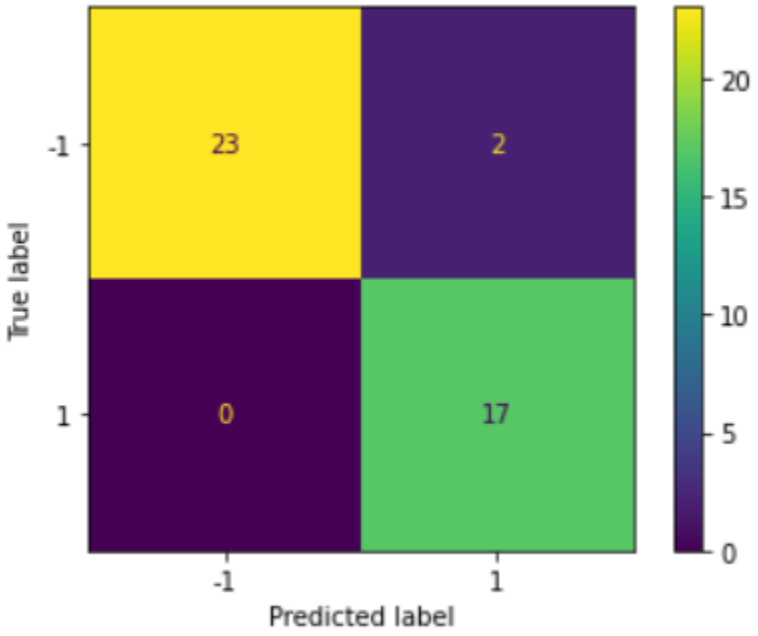


3.1.2 SVM Second Classifier (Rosa-versus-Rest)

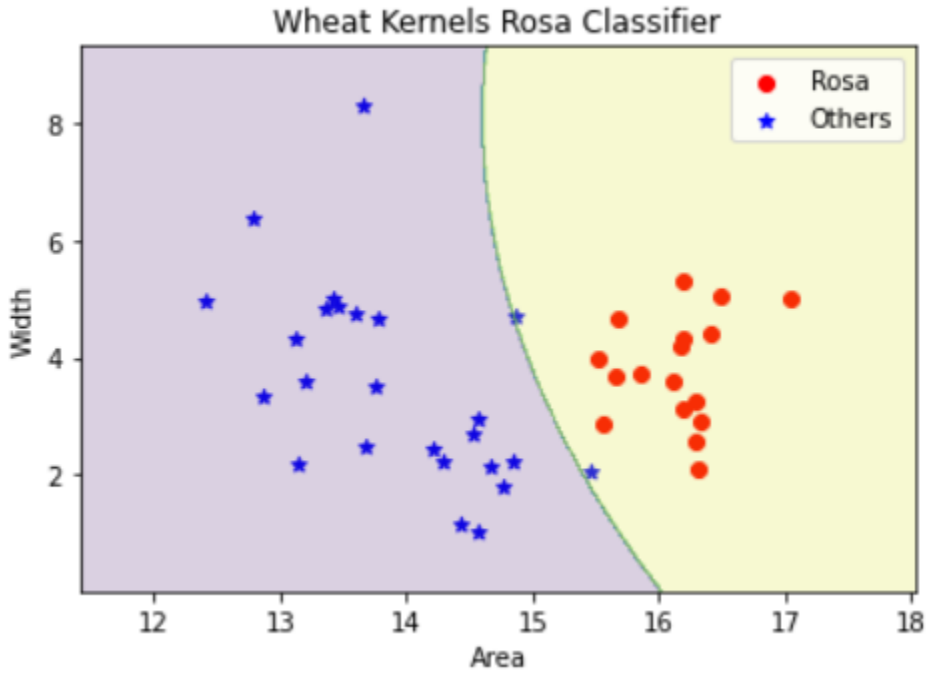
Accuracy: 95.23809523809523

	precision	recall	f1-score	support
-1	1.00	0.92	0.96	25
1	0.89	1.00	0.94	17
accuracy			0.95	42
macro avg	0.95	0.96	0.95	42
weighted avg	0.96	0.95	0.95	42

Confusion Matrix:



Decision Boundary:

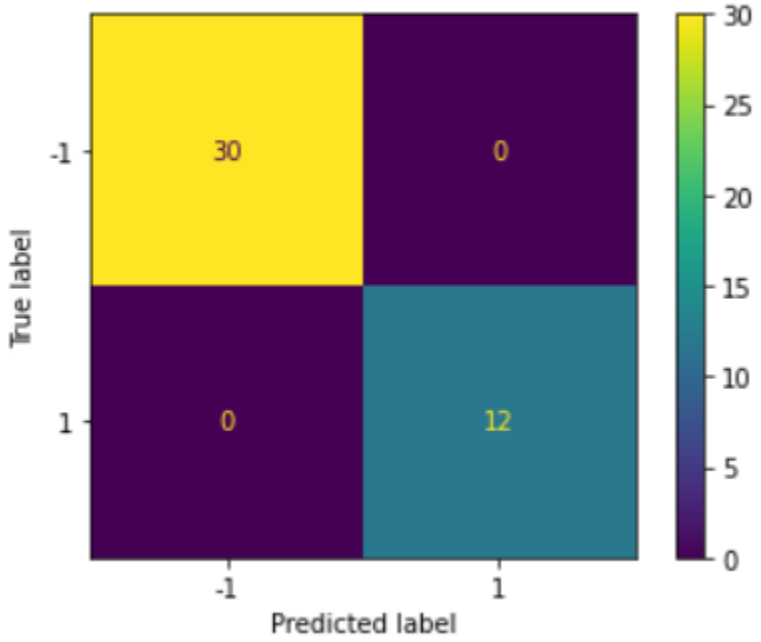


3.1.3 SVM Third Classifier (Canadian-versus-Rest)

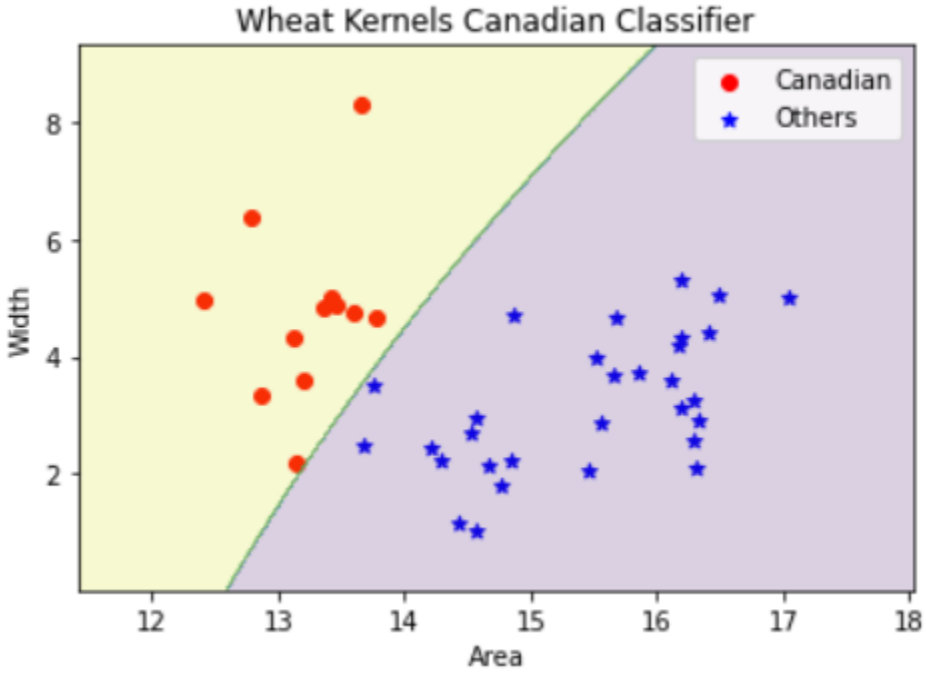
Accuracy: 100.0

	precision	recall	f1-score	support
-1	1.00	1.00	1.00	30
1	1.00	1.00	1.00	12
accuracy			1.00	42
macro avg	1.00	1.00	1.00	42
weighted avg	1.00	1.00	1.00	42

Confusion Matrix:



Decision Boundary:



3.2 OvR-Perceptron

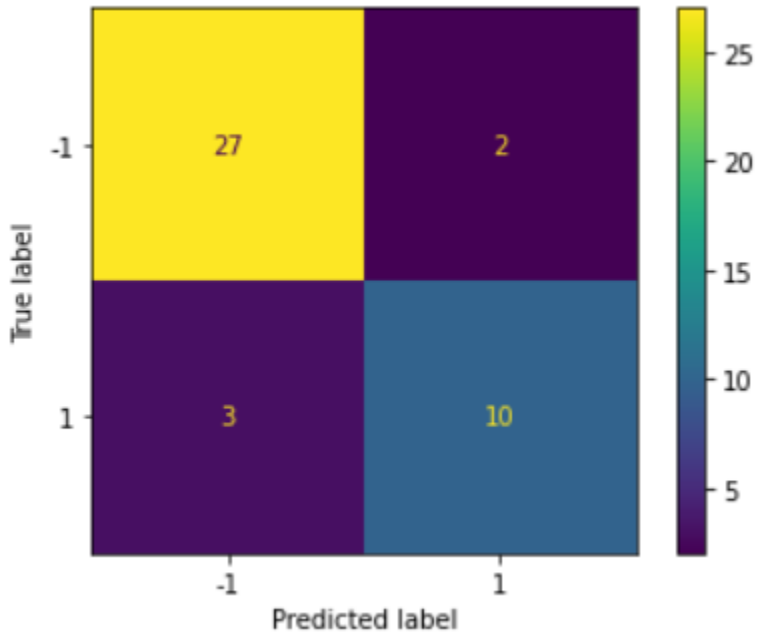
```
perceptron_model = MLPClassifier(solver = "adam", learning_rate_init = 0.01, max_iter = 450, hidden_layer_sizes = 30, random_stat
perc_prediction_probs = []
perc_predictions = []

for cls in classes:
    train_label_bin = binarize(cls, train_labels)
    test_label_bin = binarize(cls, test_labels)

    model = model_building(train_features, train_label_bin, perceptron_model, cls)
    predictions, prediction_probs = model_performance(test_features, test_label_bin, model, cls)
    perc_prediction_probs.append(prediction_probs)
    perc_predictions.append(predictions)
    model_plotting(test_features, test_label_bin, model, cls)
```

3.2.1 Perceptron First Classifier (Kama-versus-Rest)

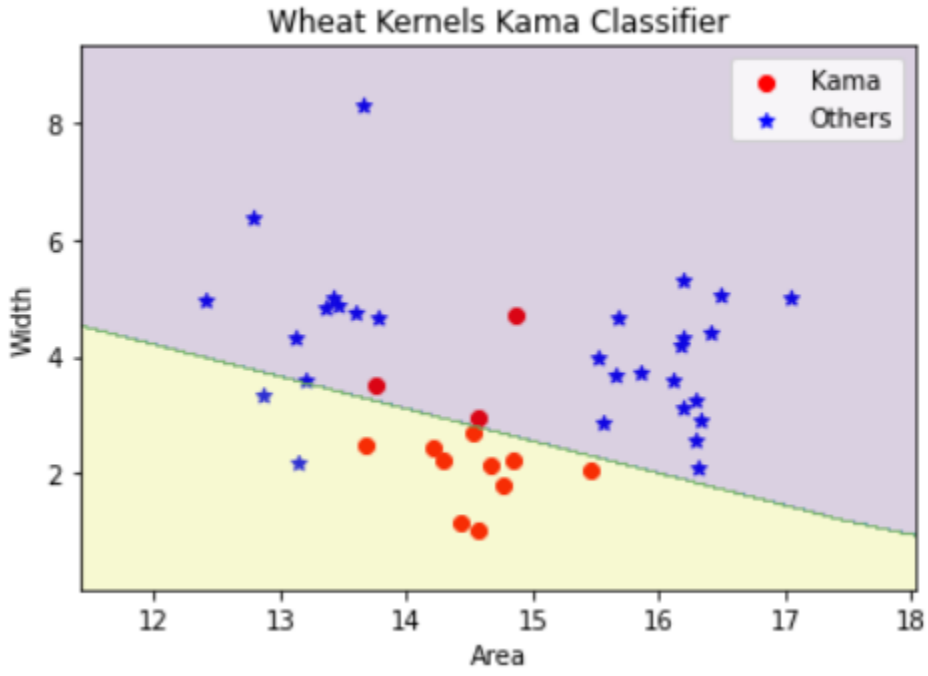
Accuracy: 88.09523809523809

	precision	recall	f1-score	support
-1	0.90	0.93	0.92	29
1	0.83	0.77	0.80	13
accuracy			0.88	42
macro avg	0.87	0.85	0.86	42
weighted avg	0.88	0.88	0.88	42

Confusion Matrix:



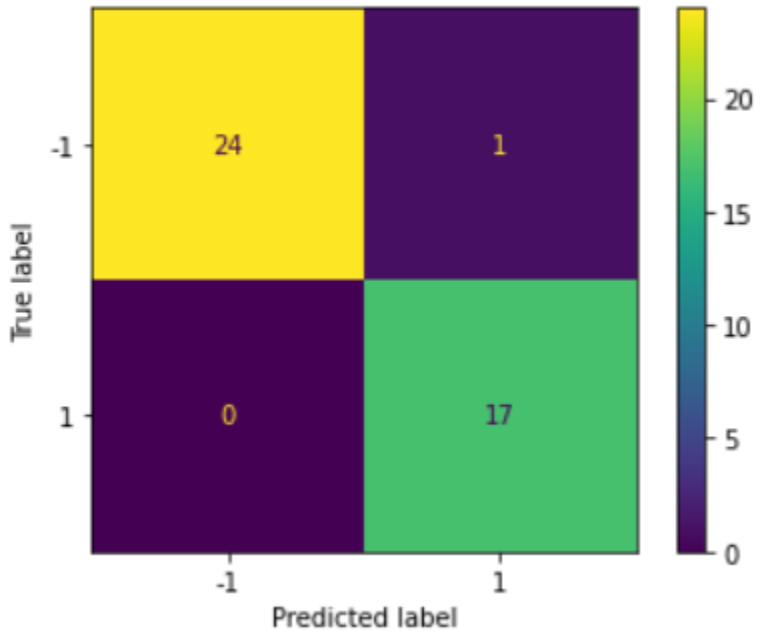
Decision Boundary:



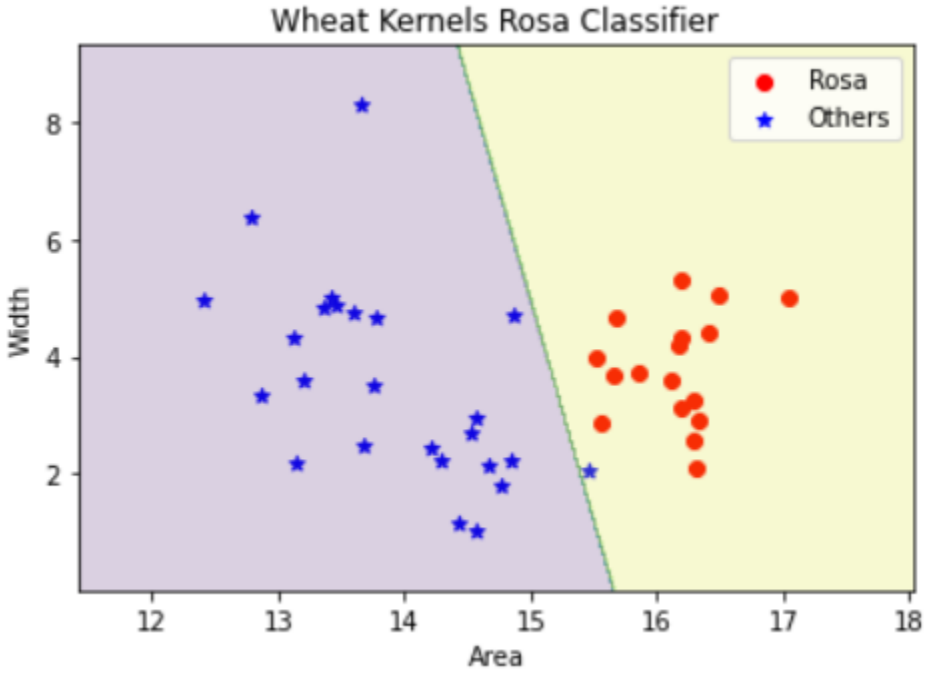
3.2.2 Perceptron Second Classifier (Rosa-versus-Rest)

```
Accuracy:  97.61904761904762
              precision    recall  f1-score   support

          -1       1.00      0.96      0.98        25
           1       0.94      1.00      0.97        17

    accuracy                           0.98        42
   macro avg       0.97      0.98      0.98        42
weighted avg       0.98      0.98      0.98        42
```

Confusion Matrix:



Decision Boundary:

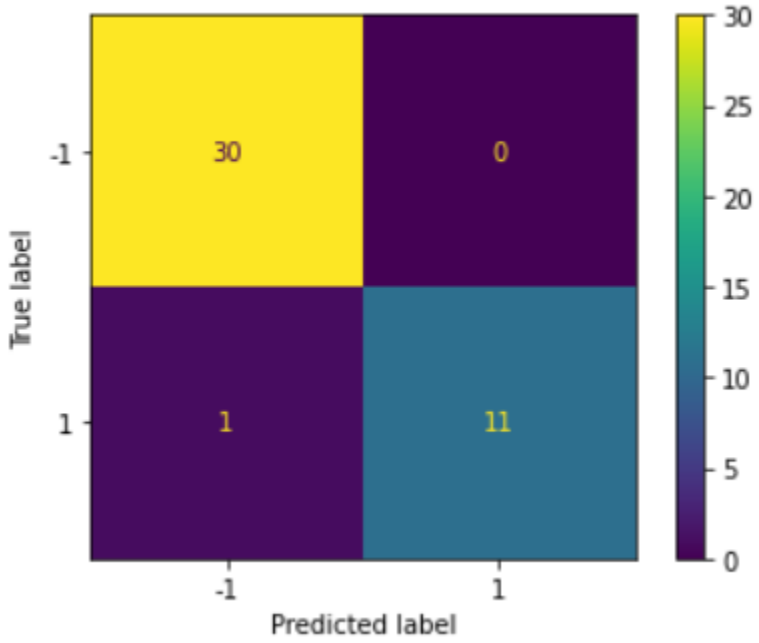


3.2.3 Perceptron Third Classifier (Canadian-versus-Rest)

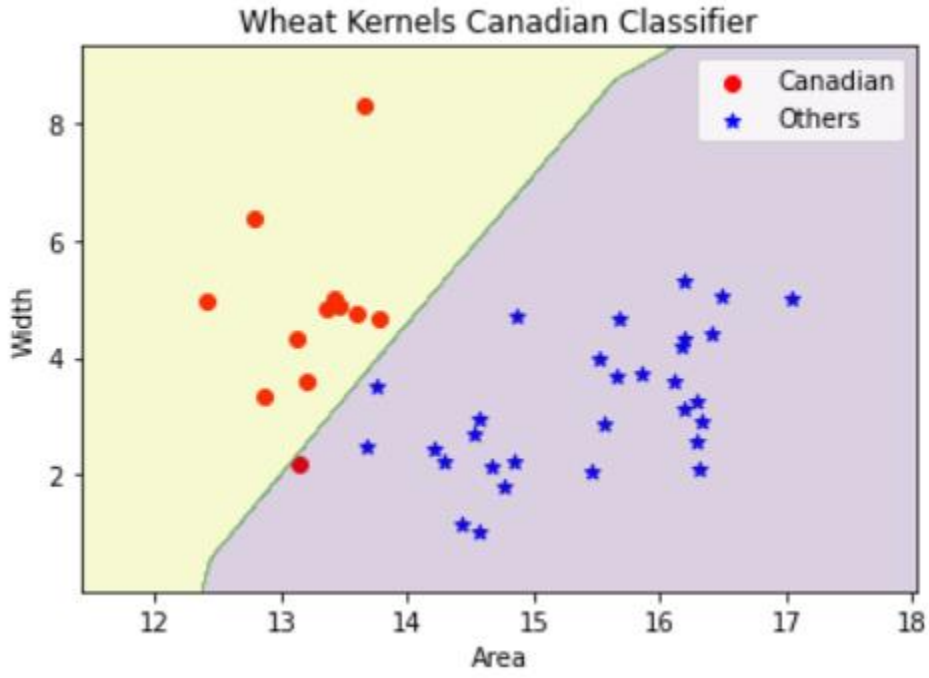
Accuracy: 97.61904761904762

	precision	recall	f1-score	support
-1	0.97	1.00	0.98	30
1	1.00	0.92	0.96	12
accuracy			0.98	42
macro avg	0.98	0.96	0.97	42
weighted avg	0.98	0.98	0.98	42

Confusion Matrix:



Decision Boundary:



Note: We used a Multi-Layer Perceptron model to be able to obtain the confidence scores for each classifier, so that they can be used with the argmax aggregation strategy.

Comparison:

SVM Classifiers	Accuracy	Precision	Recall
Kama	85.7%	0.77	0.77
Rosa	95.2%	0.89	1.0
Canadian	100%	1.0	1.0

Perceptron Classifiers	Accuracy	Precision	Recall
Kama	88.1%	0.83	0.77
Rosa	97.6%	0.94	1.0
Canadian	97.6%	1.0	0.92

The overall performance of the Perceptron model was much better than the SVM. It achieved an accuracy of 88.1% for the Kama class, having a lower false positive rate than the SVM, which achieved an accuracy of 85.7% for the Kama class. This is significant because the Kama class was not linearly separable from the other classes. Both the Rosa and the Canadian classes were linearly separable. The Perceptron model also achieved a higher accuracy of 97.6% for the Rosa class classifier and a lower false positive rate in comparison to the SVM classifier.

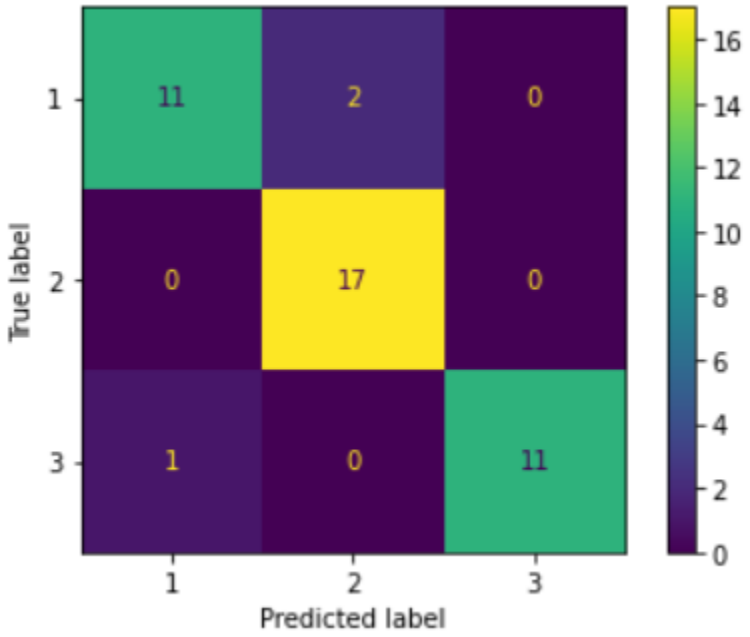
4 ARGMAX AGGREGATION

4.1 OvR-SVM

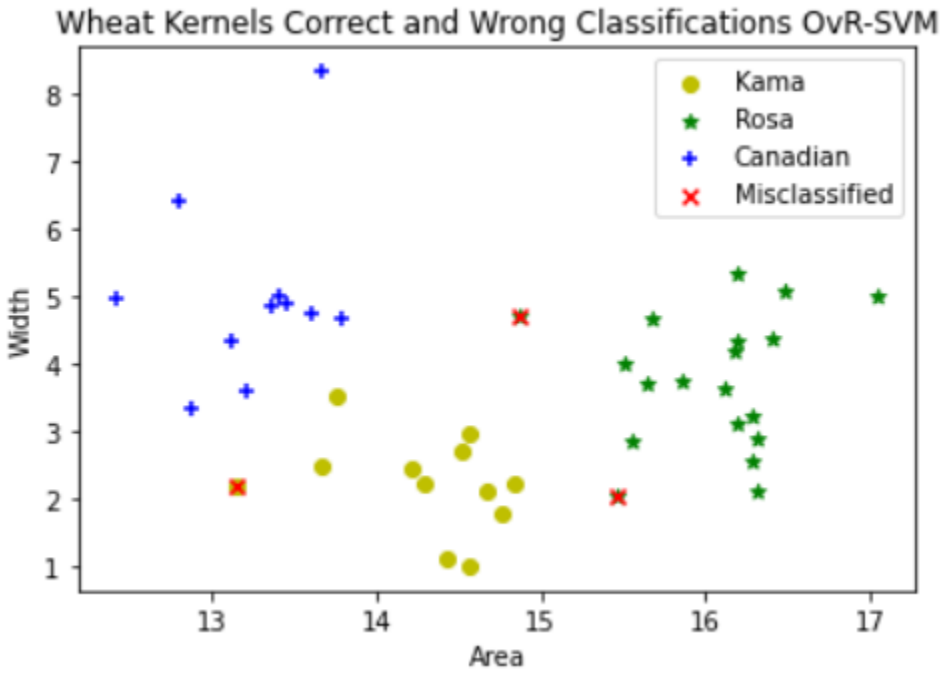
```
svm_prediction_probs = np.array(svm_prediction_probs)
svm_predictions = np.array(svm_predictions)
best_pred = argmax(svm_prediction_probs[:, :, 1], axis = 0)

best_pred_classes = []
for pred in best_pred:
    if pred == 0:
        best_pred_classes.append(1)
    if pred == 1:
        best_pred_classes.append(2)
    if pred == 2:
        best_pred_classes.append(3)
```

Accuracy: 92.8%

Confusion Matrix:



Correct and Wrong Predictions:

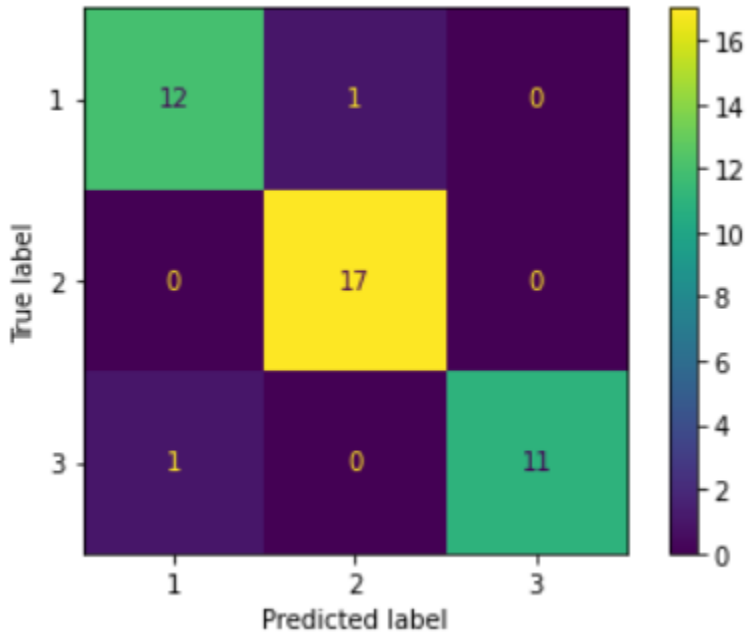


4.2 OvR-Perceptron

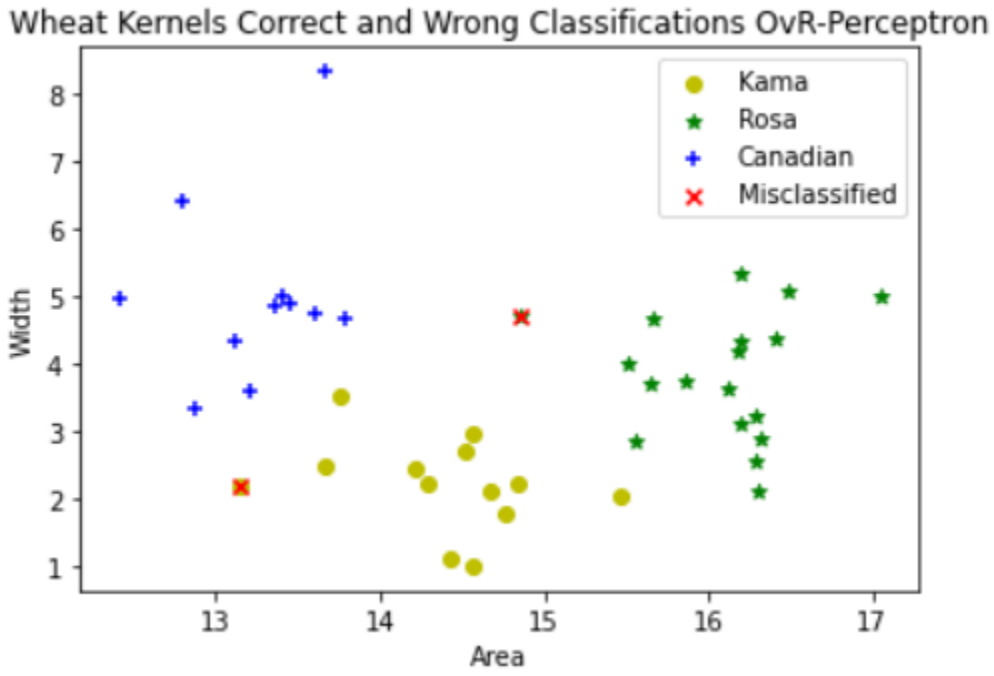
```
perc_prediction_probs = np.array(perc_prediction_probs)
perc_predictions = np.array(perc_predictions)
best_pred = argmax(perc_prediction_probs[:, :, 1], axis = 0)

best_pred_classes = []
for pred in best_pred:
    if pred == 0:
        best_pred_classes.append(1)
    if pred == 1:
        best_pred_classes.append(2)
    if pred == 2:
        best_pred_classes.append(3)
```

Accuracy: 95.2%

Confusion Matrix:



Correct and Wrong Predictions:



5 ALTERNATIVE AGGREGATION STRATEGY: KNN VOTES

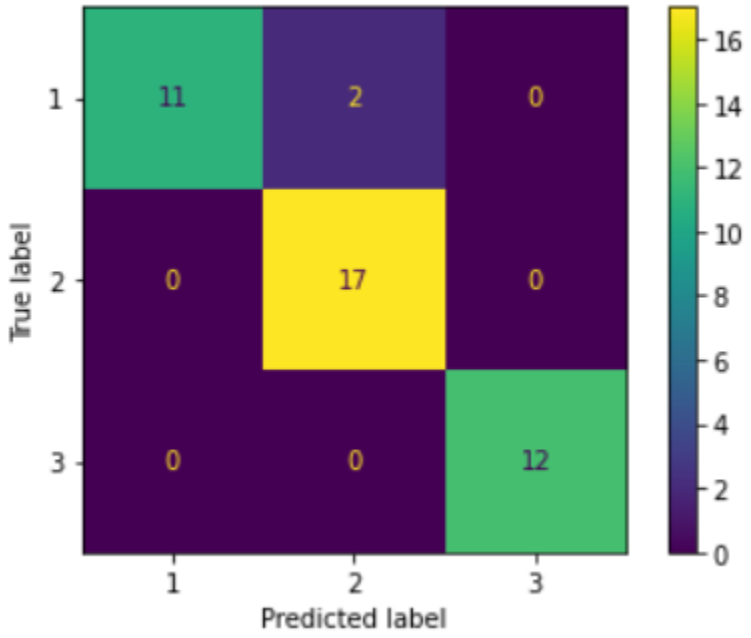
```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(train_features, train_labels["label"].tolist())

best_pred = []
for index in range(len(test_labels)):
    votes = knn.kneighbors([test_features.iloc[index].tolist()], n_neighbors = 3, return_distance = False)
    votes = train_labels["label"].iloc[votes[0].tolist()]

    c1_vote = np.count_nonzero(votes == 1)
    c2_vote = np.count_nonzero(votes == 2)
    c3_vote = np.count_nonzero(votes == 3)
    vote = [svm_prediction_probs[0, index, 1]*c1_vote, svm_prediction_probs[1, index, 1]*c2_vote, svm_prediction_probs[2, index,
    best_pred.append(argmax(vote))

best_pred_classes = []
for pred in best_pred:
    if pred == 0:
        best_pred_classes.append(1)
    if pred == 1:
        best_pred_classes.append(2)
    if pred == 2:
        best_pred_classes.append(3)
```

Our alternative aggregation strategy is a simple KNN-based voting strategy. Initially, we train a KNN model on our training set. For each test point, we use its K nearest neighbors as votes. We then count the frequency of each class in the K votes and use them as weights for the confidence scores of the three classifiers. Finally, we perform an argmax on the three weighted confidence scores and choose the best classifier for each point.

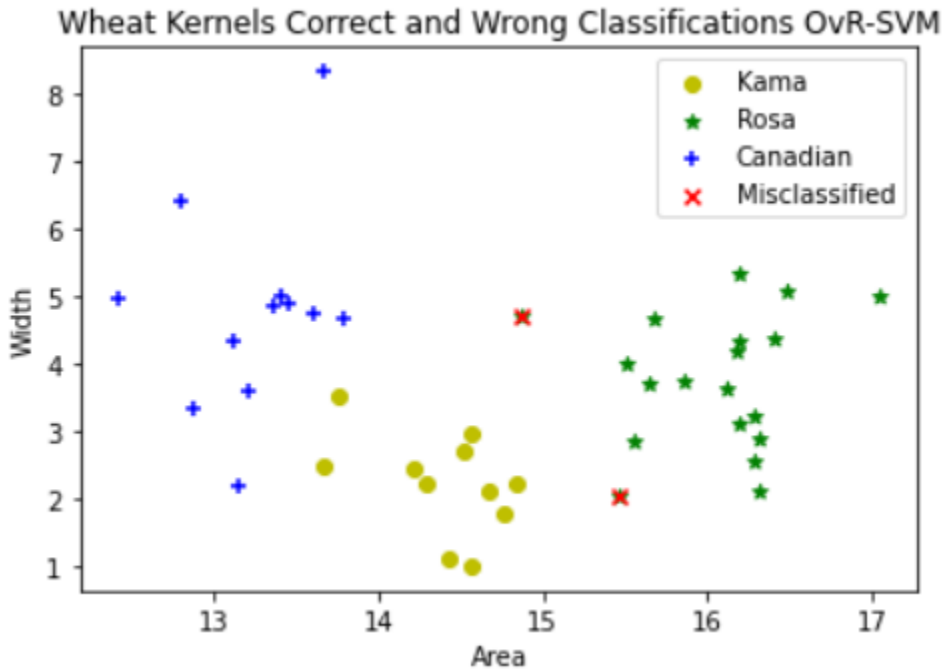
5.1 OvR-SVM

Accuracy: 95.2%

Confusion Matrix:



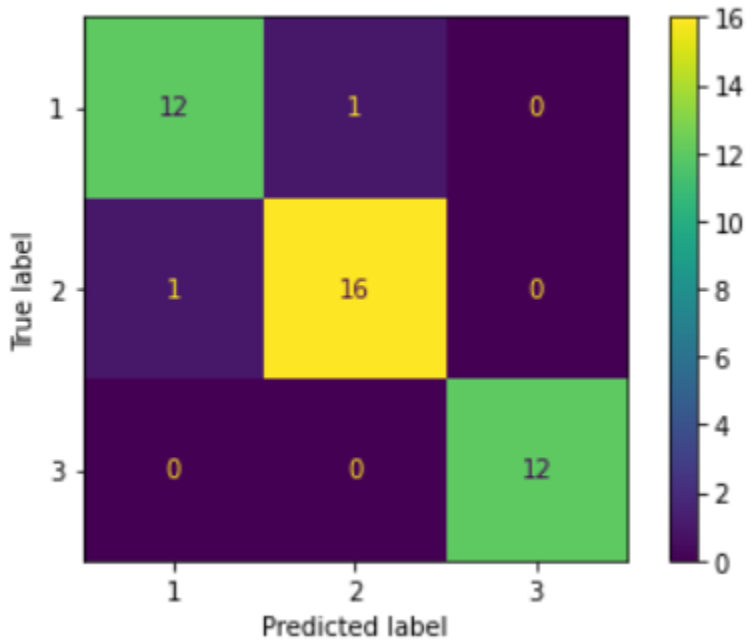
Correct and Wrong Predictions:



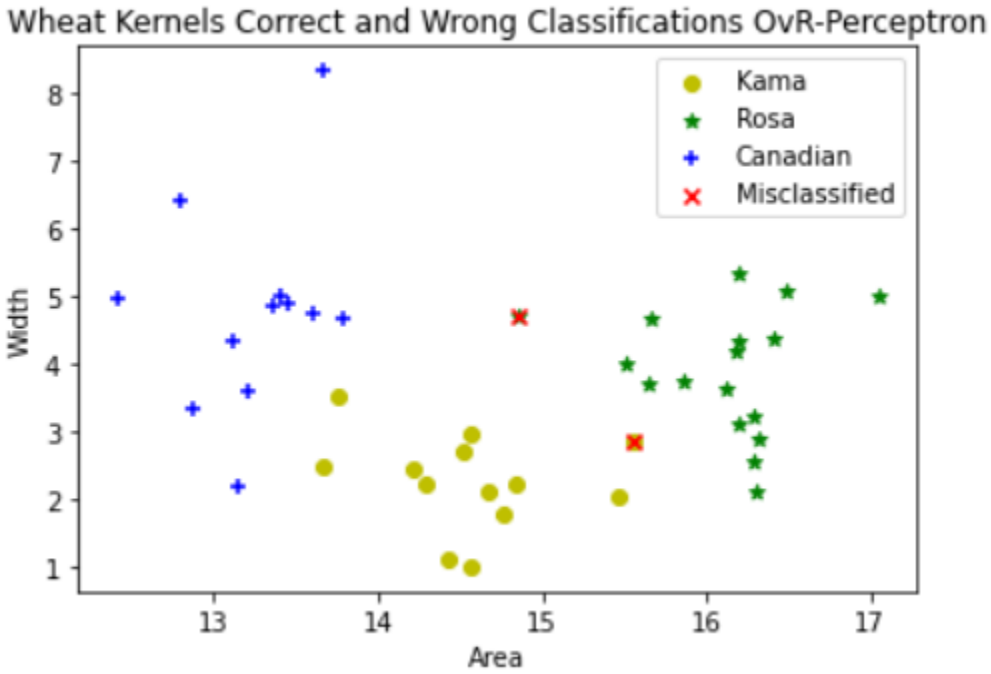
5.2 OvR-Perceptron

Accuracy: 95.2%

Confusion Matrix:



Correct and Wrong Predictions:



Comparison:

The alternative strategy improved the performance of the OvR-SVM with an increase in accuracy from 92.8% to 95.3%, but it did not affect the performance of the OvR-Perceptron, which already achieved an accuracy of 95.3%. The KNN voting strategy improved the performance of the OvR-SVM by providing votes for points that had more neighbors in the correct class than in other classes, which were originally misclassified by argmax alone. The points that stayed misclassified by both strategies were closer to points of other classes, hence they were easily misclassified.

6 CONCLUSION

In binary classification, we trained an SVM model and a perceptron model on two classes – Rosa and Canadian. The SVM classifier was trained using a linear kernel and was better at generalizing to the test dataset and achieved an accuracy of 100% since the data was linearly separable. The Perceptron classifier did not generalize well to the test data, and misclassified points that were easily classifiable.

In the one-versus-rest strategy, we trained three classifiers using both SVM and MLPerceptron. Each classifier was responsible for the classification of one of the three classes. The class labels had to be binarized into 1 and -1 so that each classifier can focus on finding a decision boundary that can separate one class from the rest of the other classes. Overall, the MLPerceptron model achieved a higher performance and was able to separate each of the three classes better than the SVM classifiers.

To make a final decision on which classifier was the most optimum in classifying a certain test point, we had to obtain the prediction probabilities/confidence scores from each of the three classifiers. An argmax function was used to obtain the maximum confidence score for each point, returning the index of the classifier that had the highest confidence. This aggregation strategy provided the optimum classifications for each class using the three trained classifiers.

The KNN voting strategy improved on the argmax by providing weights to the confidence scores of each of the three classifiers. These votes consisted of the frequency of each class in the K nearest neighbors of each test point.