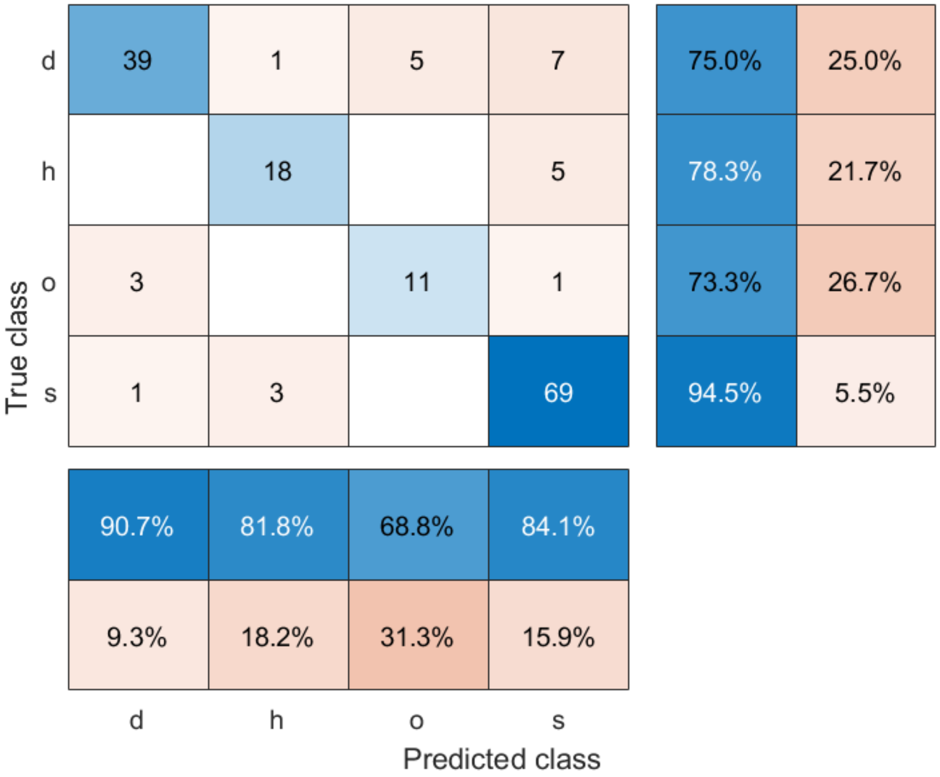
Forrest Classification

Details on how the data was split into training, validation, and testing sets

The Testing and training datasets were provided within the files of the assignment however due to the lack of validation data, the testing set was split equally between testing and validation data. Considering the number of rows in the testing dataset, this was done to maintain a good distribution between the sets.

K-Nearest Neighbors Classifier

4 CKNN models were trained using 5, 10, 20 and 30 number of neighbours. Based on the confusion chart, while there isn't a major difference between the models, the models with the higher number of neighbours tend to overclassify in the more populated classes (especially the "Sugi forest" class). Therefore, the model with 10 neighbours represents the best balance.

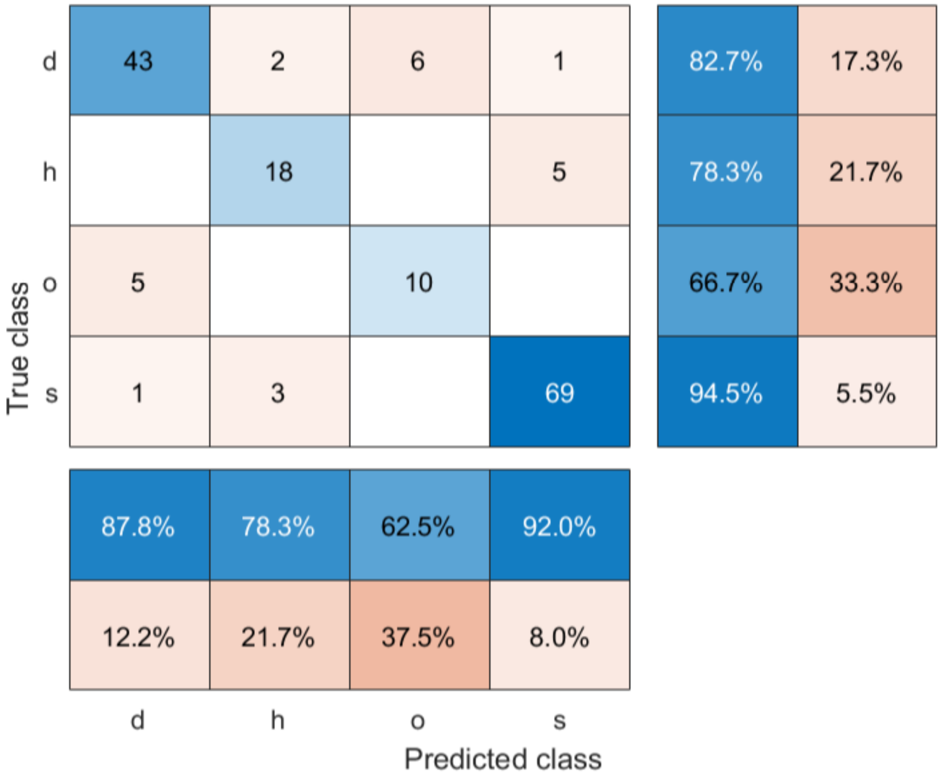


By exploring the categorical distribution of the training data, a slight imbalance can be observed, especially in the "other" category. However, it is not significant enough to justify taking steps to balance the data.

Binary classifiers

SVM

SVM models with box constraints ranging from infinity to 0.001 were trained and compared. It was concluded that the tested C values above 0.001 did not impact the results. In addition, reducing the box constraint to 0.001 negatively impacted the model's accuracy. Consequently, the default SVM model with a C value of 1 will be used moving forward.



True class	d	h	o	s		
d	43	2	6	1	82.7%	17.3%
h		18		5	78.3%	21.7%
o	5		10		66.7%	33.3%
s	1	3		69	94.5%	5.5%
	87.8%	78.3%	62.5%	92.0%		
	12.2%	21.7%	37.5%	8.0%		
	d	h	o	s		

Predicted class

To find the best possible model, the following hyper-parameters were experimented with:

Standardize: true

KernelFunction: RBF

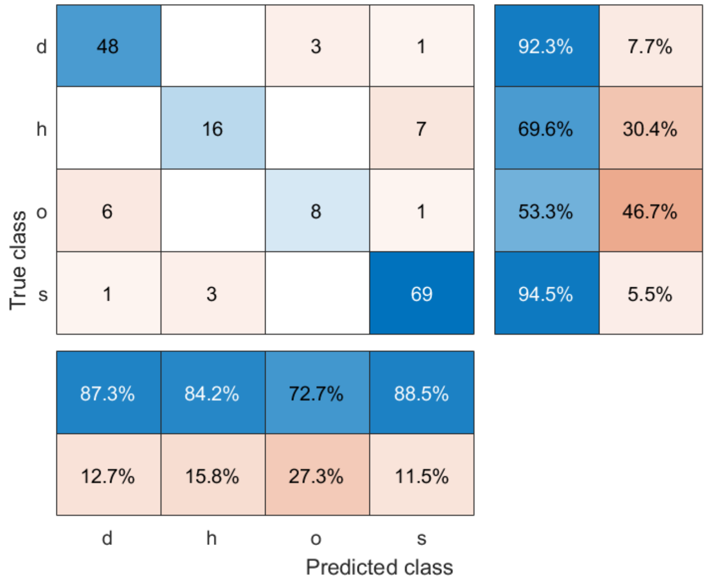
KernelScale: Default, 100

PolynomialOrder: 2, 5

Though, the best prediction of the validation data was achieved using the default SVM model.

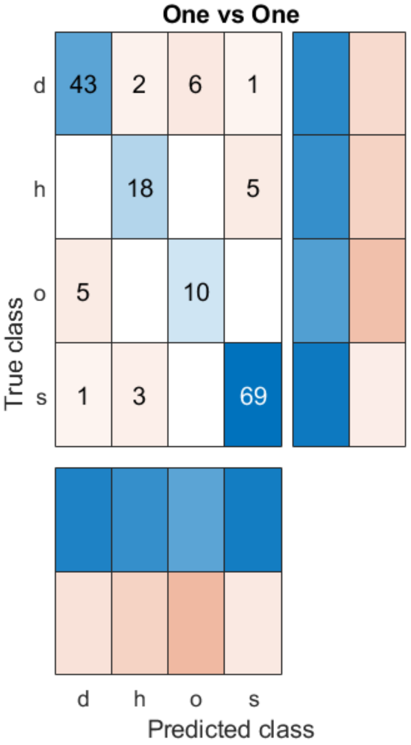
Linear model

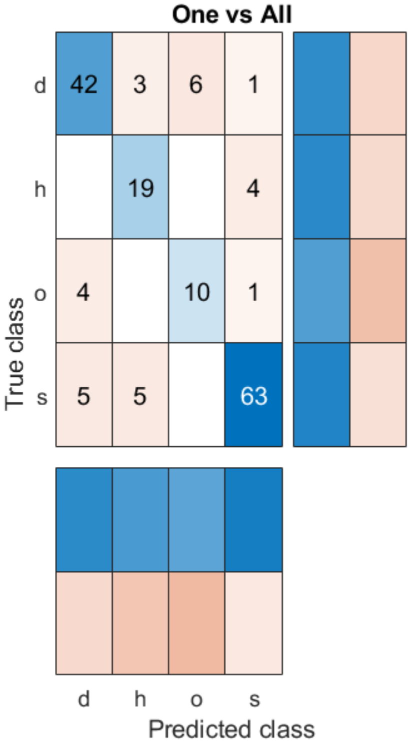
The linear model produced a slightly more accurate prediction compared to the SVM model at the cost of more false predictions in the more populated categories. Based on the usage requirements, either model can be benedictional. Although for the sake of better generalisation, The SVM model is preferred.



Multi-class Classification

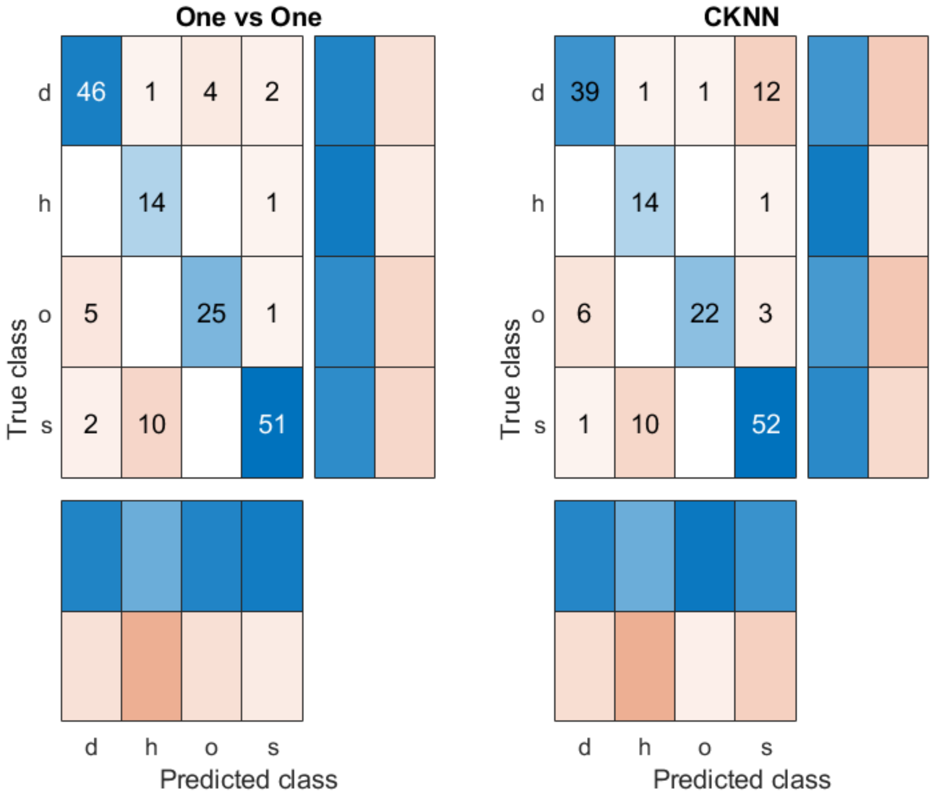
Comparing One vs One and One vs All Classification methods, the former performs poorly due to the imbalance between the categories of the training data. Considering the computational penalty inherent with the One vs One method is insignificant when working with smaller data, this approach is justifiable.





Evaluation and comparison of the final two models

The confusion chart of the One vs One SVM model and the CKNN model on the test dataset are illustrated below. The SVM model has correctly classified 136 of the entries whereas the CKNN model has only managed to identify 127 true cases. In addition, the misclassifications in the SVM model are better distributed comparatively. In the end, considering that the One vs One SVM model is more computationally demanding, the choice of model will depend on the desired compromise between speed and accuracy.



Appendices

Loading the data

Breaking the training set into training and validation sets with a rough distribiution on 70% to 30%

```
% spectral_data = readtable('training.csv');
% test_data = readtable('testing.csv');
%
% x_sp = table2array(spectral_data(2:198,2:28));
% y_sp = table2array(spectral_data(2:198,1));
%
% x_test_sp = table2array(test_data(2:end,2:end));
% y_test_sp = table2array(test_data(2:end,1));
%
% x_train_sp = x_sp(1:150, :);
% x_val_sp = x_sp(151:end, :);
%
% y_train_sp = y_sp(1:140);
% y_val_sp = y_sp(141:end);
```

Loading the data

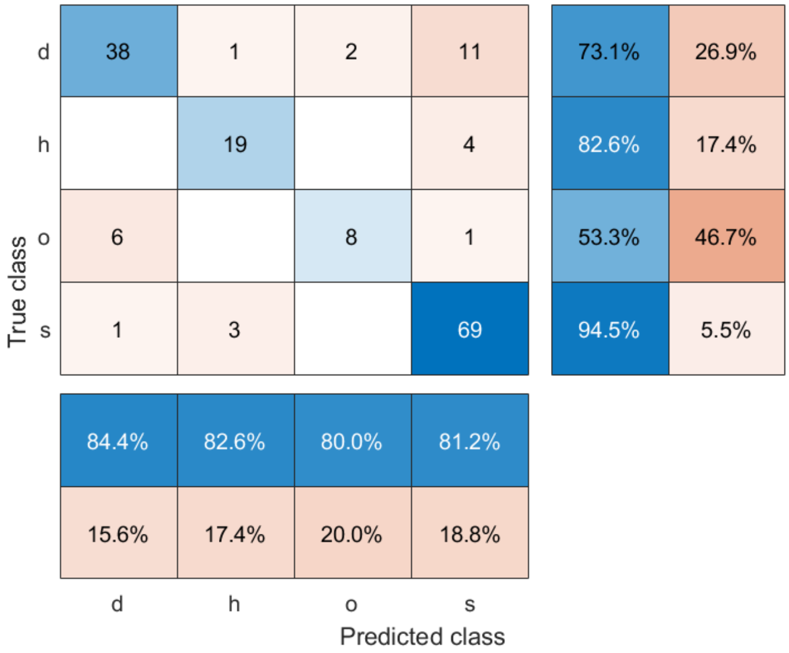
Breaking the testing set into testing and validation sets with a rough distribiution of 50% to 50%

```
spectral_data = readtable('training.csv');
test_data = readtable('testing.csv');

x_old_test = table2array(test_data(2:end,2:end));
y_old_test = table2array(test_data(2:end,1));

x_train_sp = table2array(spectral_data(2:end,2:end));
y_train_sp = table2array(spectral_data(2:end,1));

x_test_sp = x_old_test(1:162, :);
x_val_sp = x_old_test(162:end, :);

y_test_sp = y_old_test(1:162);
y_val_sp = y_old_test(162:end);
```

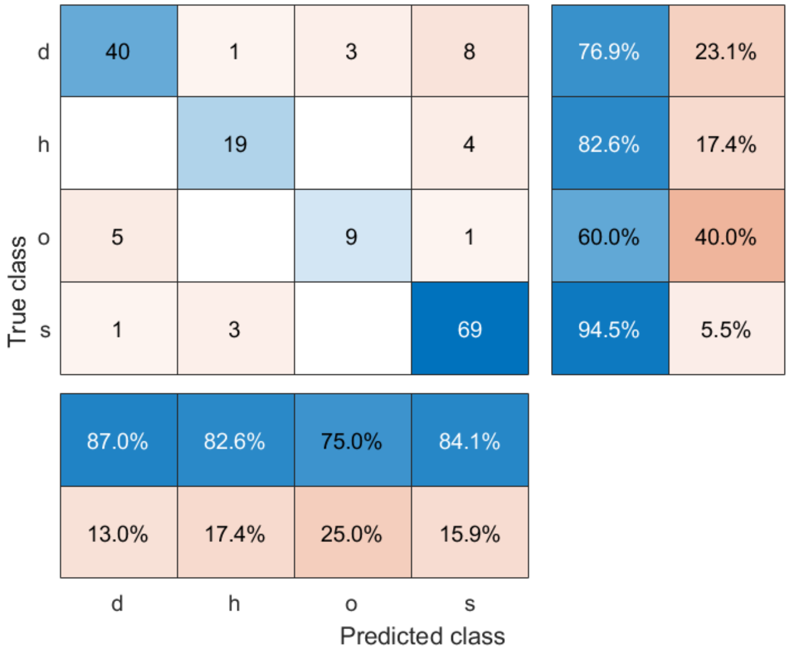
Part 1: CKNN

We'll start with a simple CKNN model.

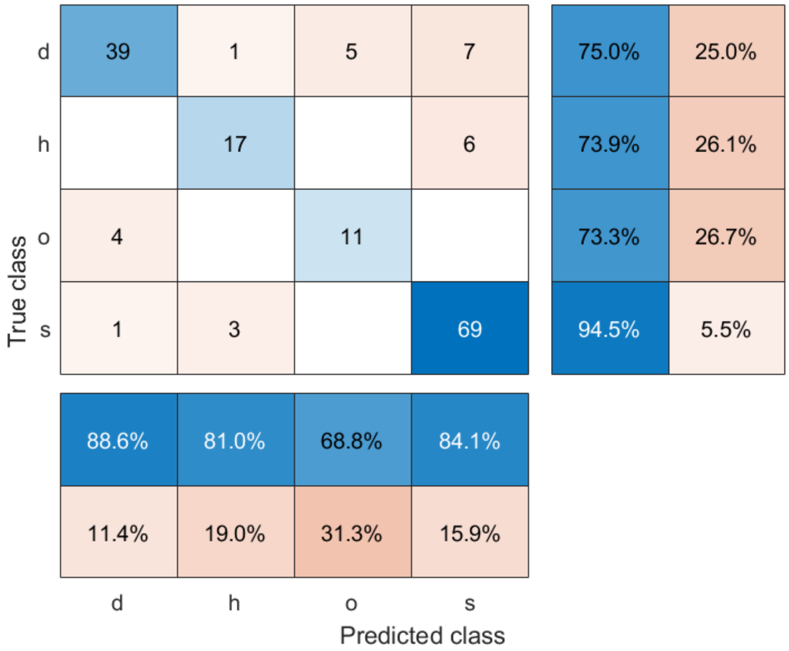
```
cknn_sp = fitcknn(x_train_sp, y_train_sp, 'NumNeighbors', 30);
pred = cknn_sp.predict(x_val_sp);
figure
confusionchart(y_val_sp, pred, 'ColumnSummary','column-normalized','RowSummary','row-normalized');
```



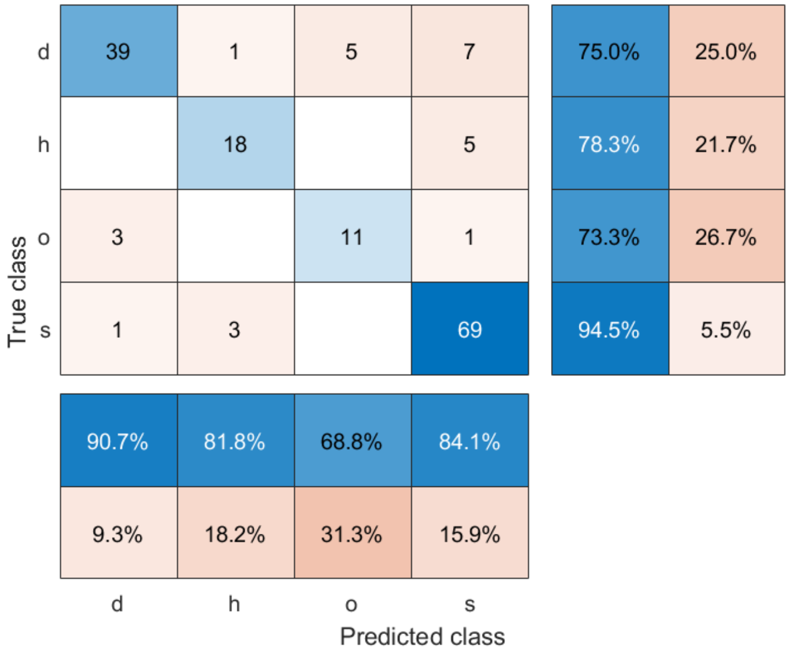
```
cknn_sp = fitcknn(x_train_sp, y_train_sp, 'NumNeighbors', 20);
pred = cknn_sp.predict(x_val_sp);
figure
confusionchart(y_val_sp, pred, 'ColumnSummary','column-normalized','RowSummary','row-normalized');
```



```
cknn_sp = fitcknn(x_train_sp, y_train_sp, 'NumNeighbors', 5);
pred = cknn_sp.predict(x_val_sp);
figure
confusionchart(y_val_sp, pred, 'ColumnSummary','column-normalized','RowSummary','row-normalized');
```



```
cknn_sp = fitcknn(x_train_sp, y_train_sp, 'NumNeighbors', 10);
pred = cknn_sp.predict(x_val_sp);
figure
confusionchart(y_val_sp, pred, 'ColumnSummary','column-normalized','RowSummary','row-normalized');
```

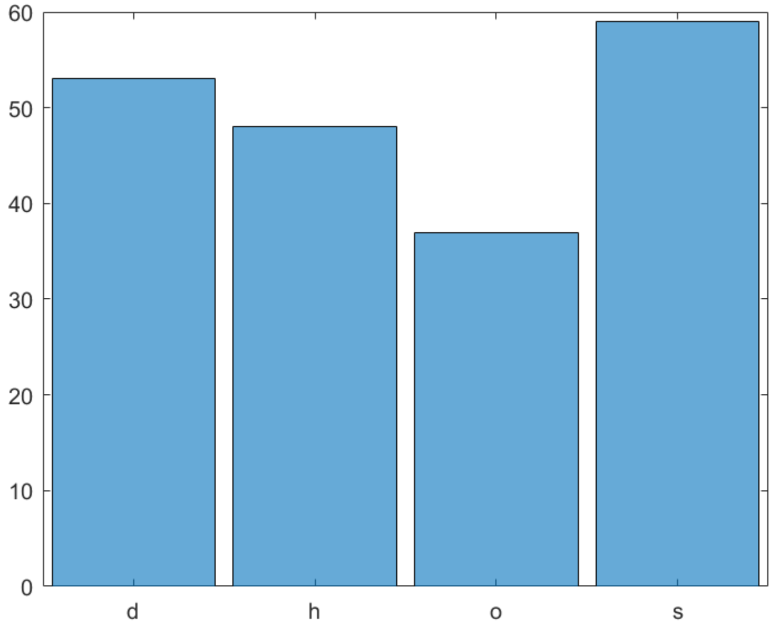


	d	h	o	s		
d	39	1	5	7	75.0%	25.0%
h		18		5	78.3%	21.7%
o	3		11	1	73.3%	26.7%
s	1	3		69	94.5%	5.5%
	90.7%	81.8%	68.8%	84.1%		
	9.3%	18.2%	31.3%	15.9%		

NumNeighbors 10 is the most balanced model

Checking the distribution of the data

```
figure
cat_y_sp = categorical(y_train_sp);
histogram(cat_y_sp)
```

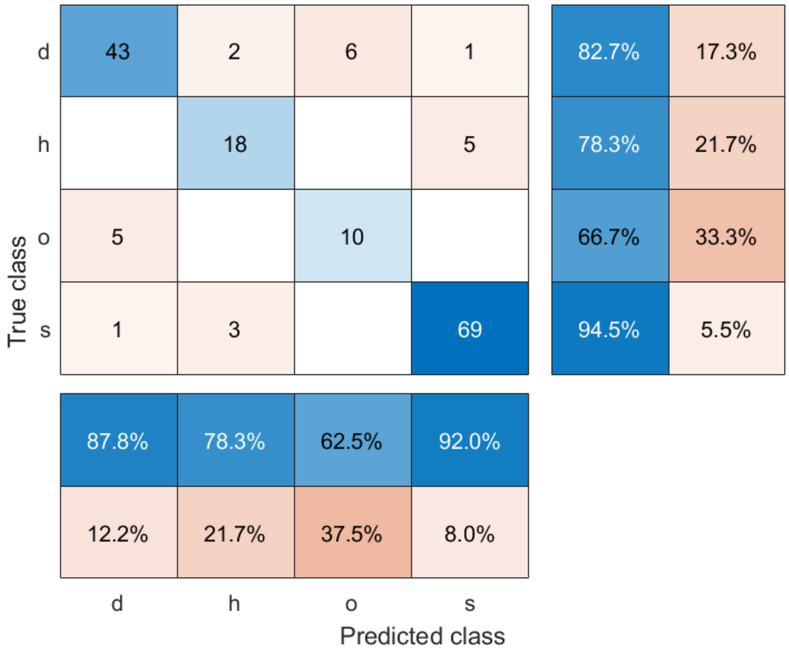


Part 2: Ensemble Classifier

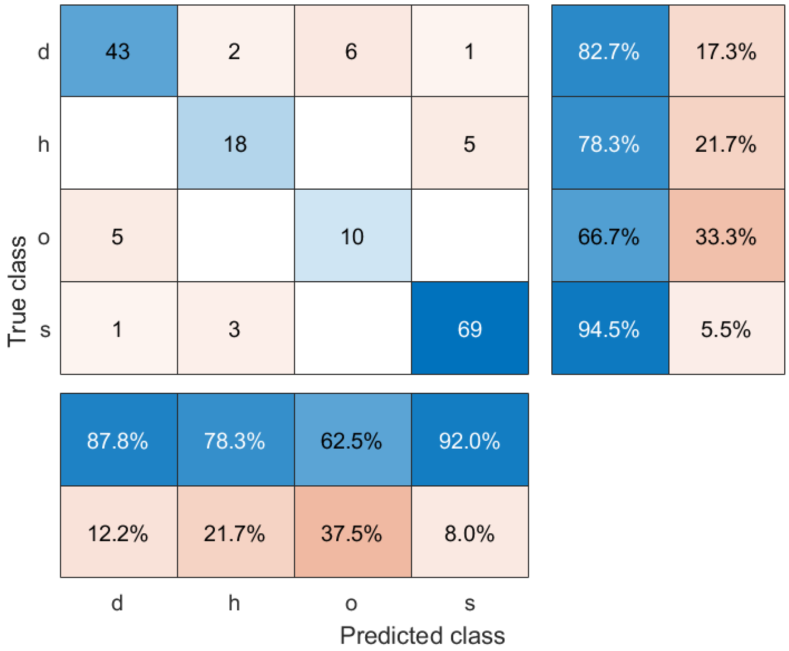
Finding the best binary classifier:

SVM-BoxConstraint:

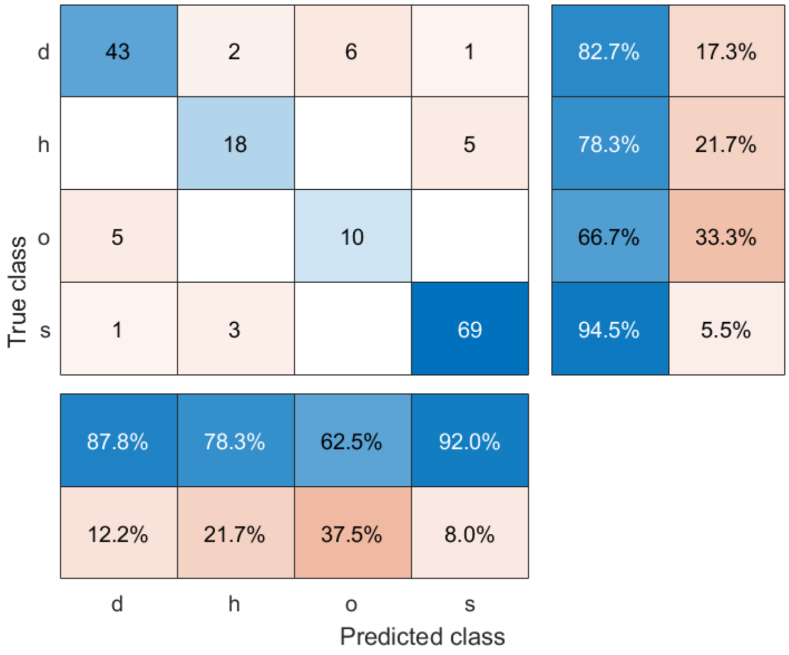
```
 svm_inf = fitcecoc(x_train_sp,
y_train_sp,'Learners',templateSVM('BoxConstraint', inf));
 pred = svm_inf.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```



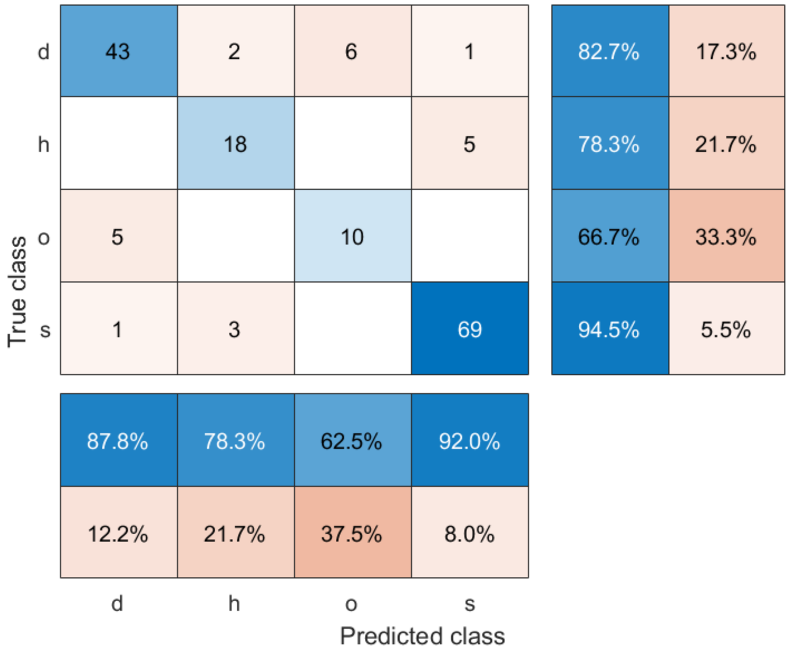
```
 svm_1000 = fitcecoc(x_train_sp,
y_train_sp,'Learners',templateSVM('BoxConstraint', 1000));
 pred = svm_1000.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```



```
 svm_1 = fitcecoc(x_train_sp, y_train_sp,'Learners',templateSVM('BoxConstraint',
1));
 pred = svm_1.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```



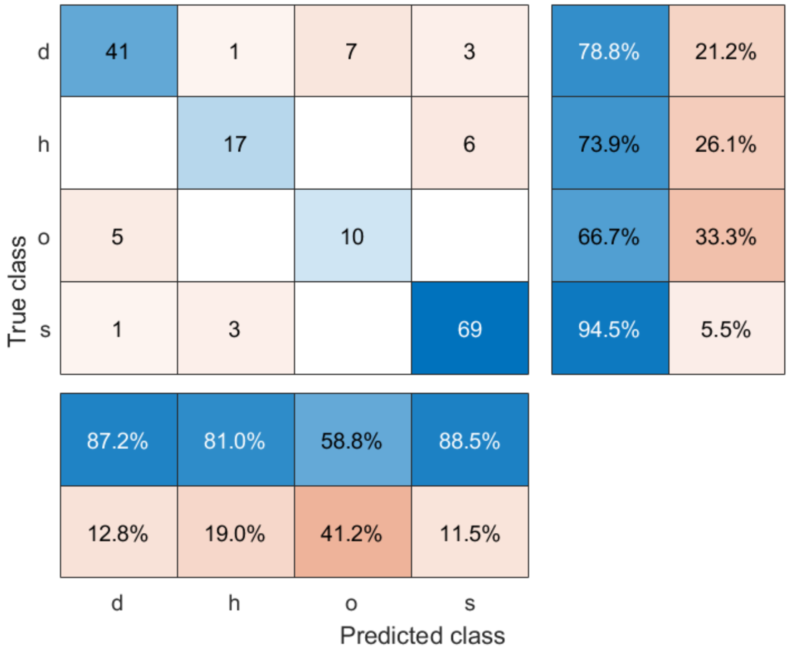
```
 svm_01 = fitcecoc(x_train_sp,
y_train_sp,'Learners',templateSVM('BoxConstraint', 0.1));
 pred = svm_01.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```



True class	d	h	o	s		
d	43	2	6	1	82.7%	17.3%
h		18		5	78.3%	21.7%
o	5		10		66.7%	33.3%
s	1	3		69	94.5%	5.5%
	87.8%	78.3%	62.5%	92.0%		
	12.2%	21.7%	37.5%	8.0%		
	d	h	o	s		

Predicted class

```
 svm_001 = fitcecoc(x_train_sp,
y_train_sp,'Learners',templateSVM('BoxConstraint', 0.001));
 pred = svm_001.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```



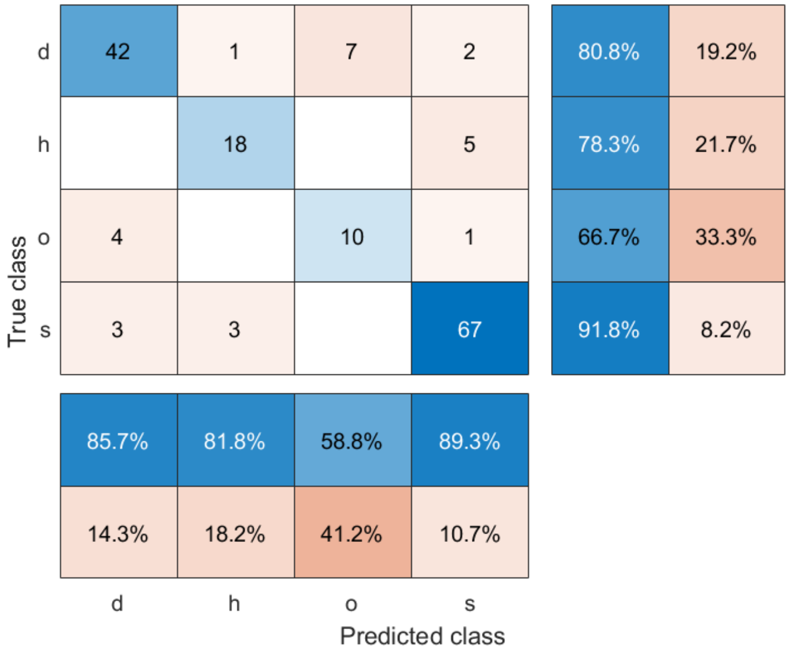
True class	d	h	o	s		
d	41	1	7	3	78.8%	21.2%
h		17		6	73.9%	26.1%
o	5		10		66.7%	33.3%
s	1	3		69	94.5%	5.5%
	87.2%	81.0%	58.8%	88.5%		
	12.8%	19.0%	41.2%	11.5%		
	d	h	o	s		

Predicted class

Box Constraint does not seem to change the results above 0.001 which causes a worse prediction and therefore the default box constraint (1) will be used.

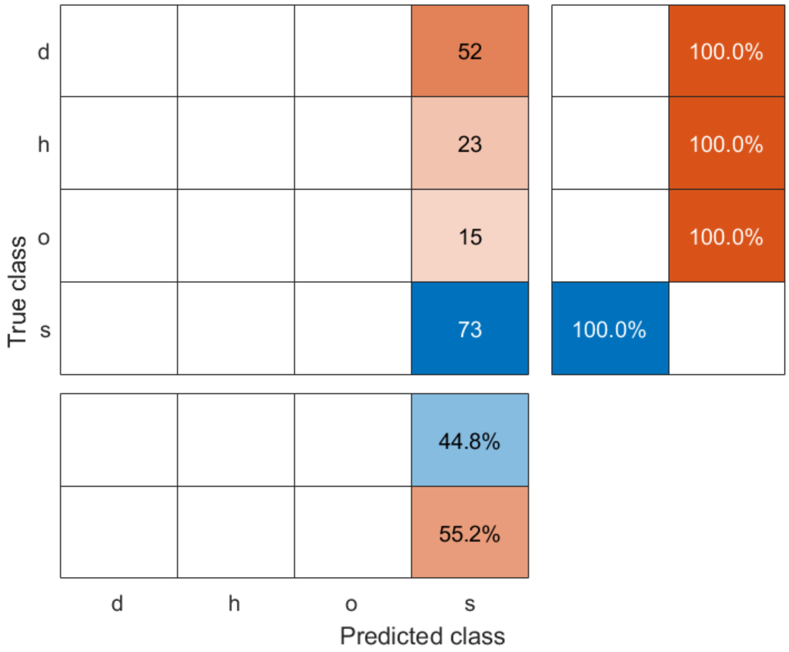
Standardize

```
 svm_1_s = fitcecoc(x_train_sp,
y_train_sp,'Learners',templateSVM('BoxConstraint', 1, 'Standardize',true));
 pred = svm_1_s.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```

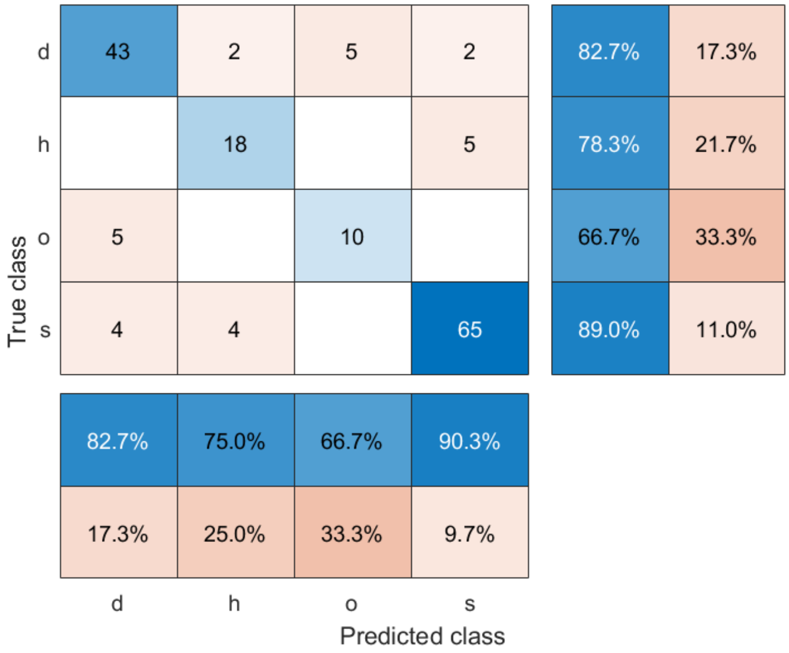


Standardizing does not improve prediction

```
 svm_1_rbf = fitcecoc(x_train_sp,
y_train_sp,'Learners',templateSVM('KernelFunction','rbf'));
 pred = svm_1_rbf.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```

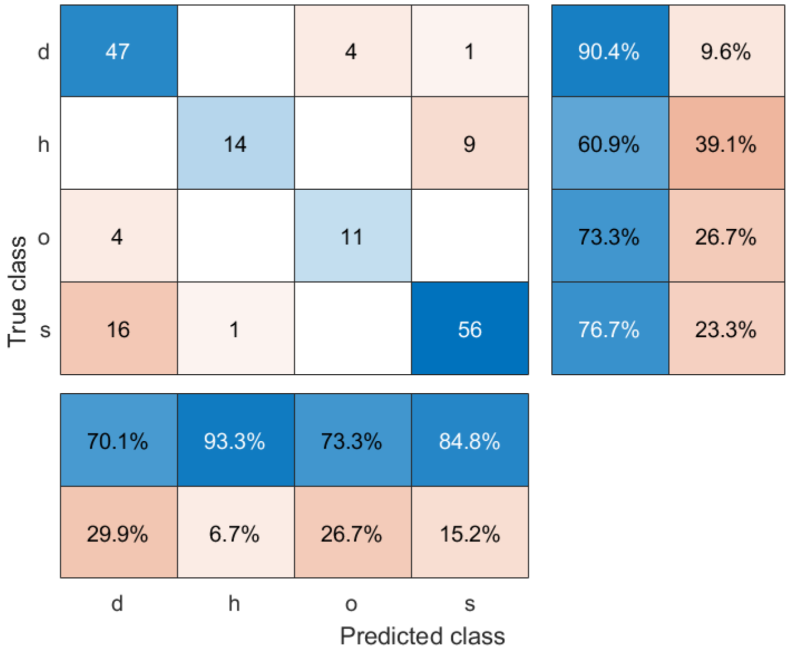


```
 svm_1_rbf_100 = fitcecoc(x_train_sp,
y_train_sp,'Learners',templateSVM('KernelFunction','rbf', 'KernelScale',100,
'BoxConstraint',1000, 'Standardize',true));
 pred = svm_1_rbf_100.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```

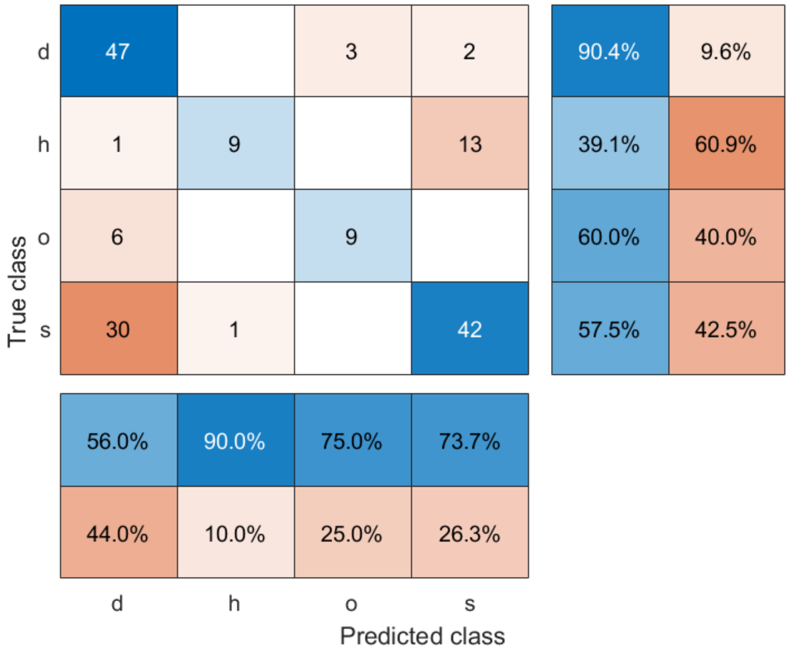


The default svm still produces better results

```
 svm_1_poly_2 = fitcecoc(x_train_sp,
y_train_sp,'Learners',templateSVM('KernelFunction','polynomial',
'Standardize',true, 'PolynomialOrder', 2));
 pred = svm_1_poly_2.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```



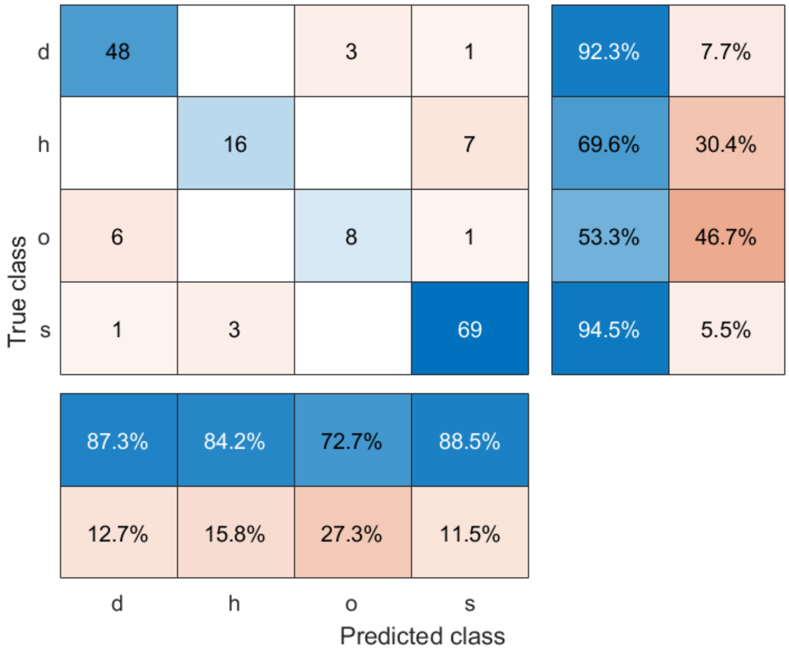
```
 svm_1_poly_5 = fitcecoc(x_train_sp,
y_train_sp,'Learners',templateSVM('KernelFunction','polynomial',
'Standardize',true, 'PolynomialOrder', 5));
 pred = svm_1_poly_5.predict(x_val_sp);
 confusionchart(y_val_sp, pred, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
```



Default svm model is still perfoming better

Linear model

```
linear = fitcecoc(x_train_sp, y_train_sp,'Learners',templateLinear());
pred = linear.predict(x_val_sp);
confusionchart(y_val_sp, pred, 'ColumnSummary','column-normalized','RowSummary','row-normalized');
```



	d	h	o	s		
d	48		3	1	92.3%	7.7%
h		16		7	69.6%	30.4%
o	6		8	1	53.3%	46.7%
s	1	3		69	94.5%	5.5%
	87.3%	84.2%	72.7%	88.5%		
	12.7%	15.8%	27.3%	11.5%		

The best model was the default SVM

One vs One

```
svm_onevone = fitcecoc(x_train_sp, y_train_sp, 'Coding','onevsone');
disp(svm_onevone.CodingMatrix)
```

```
     1     1     1     0     0     0
    -1     0     0     1     1     0
     0    -1     0    -1     0     1
     0     0    -1     0    -1    -1
```

One vs All

```
svm_onevall = fitcecoc(x_train_sp, y_train_sp, 'Coding','onevsall');
disp(svm_onevall.CodingMatrix)
```

```
     1    -1    -1    -1
    -1     1    -1    -1
    -1    -1     1    -1
    -1    -1    -1     1
```

```
figure
subplot(1,2,1)
```

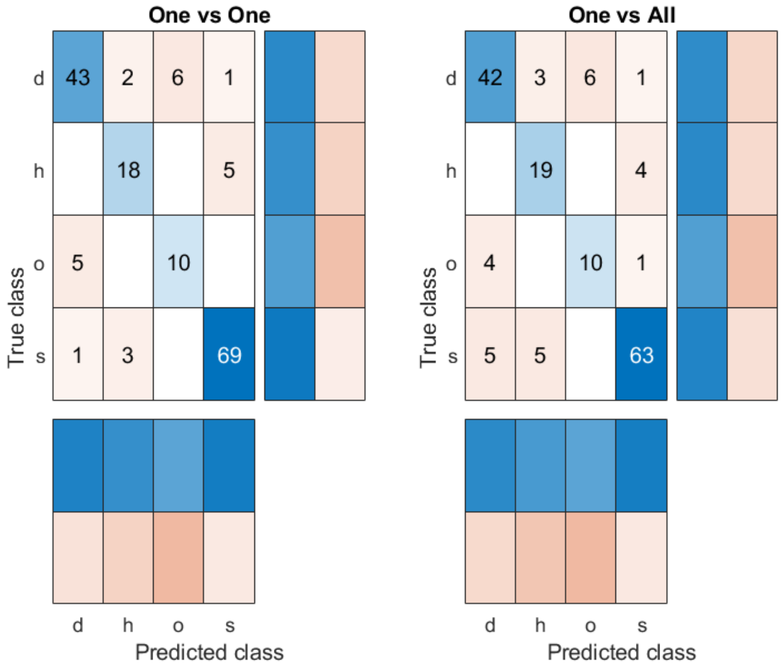
```
 cat_y_val_sp = categorical(y_val_sp);
 pred_onevone = svm_onevone.predict(x_val_sp);
 disp("Accuracy (One v One): " + string(sum(pred_onevone == cat_y_val_sp) / length(cat_y_val_sp)))
```

```
Accuracy (One v One): 0.8589
```

```
 confusionchart(y_val_sp, pred_onevone, 'ColumnSummary','column-normalized','RowSummary','row-normalized');
 title('One vs One')
 subplot(1,2,2)
 pred_onevall = svm_onevall.predict(x_val_sp);
 disp("Accuracy (One v All): " + string(sum(pred_onevall == cat_y_val_sp) / length(cat_y_val_sp)))
```

```
Accuracy (One v All): 0.82209
```

```
 confusionchart(y_val_sp, pred_onevall, 'ColumnSummary','column-normalized','RowSummary','row-normalized');
 title('One vs All')
```



The one vs one model perfoms better compared to all binary classifiers.

Applying the models to the test data

```
 figure
 subplot(1,2,1)
 cat_y_test_sp = categorical(y_test_sp);
```

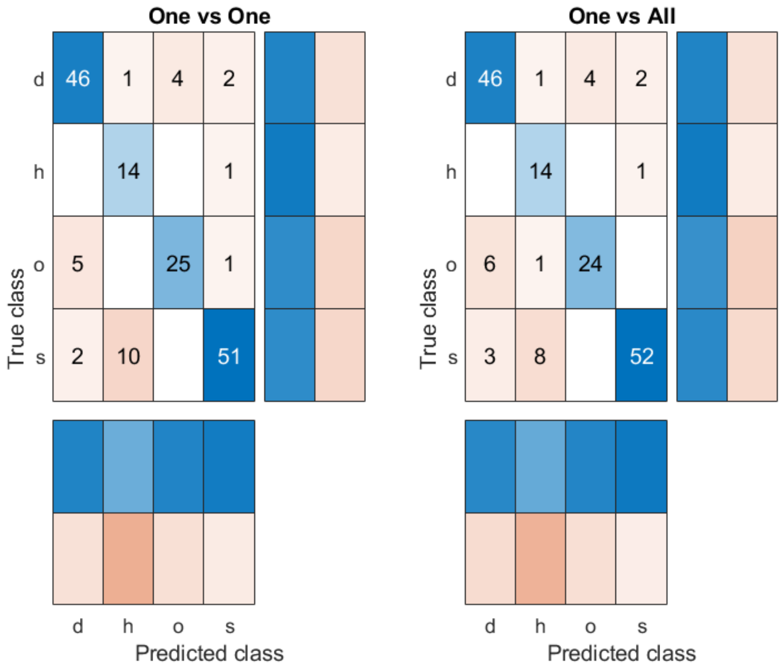
```
pred_onevone = svm_onevone.predict(x_test_sp);
disp("Accuracy (One v One): " + string(sum(pred_onevone == cat_y_test_sp) / length(cat_y_test_sp)))
```

```
Accuracy (One v One): 0.83951
```

```
confusionchart(y_test_sp, pred_onevone, 'ColumnSummary','column-normalized','RowSummary','row-normalized');
title('One vs One')
subplot(1,2,2)
pred_onevall = svm_onevall.predict(x_test_sp);
disp("Accuracy (One v All): " + string(sum(pred_onevall == cat_y_test_sp) / length(cat_y_test_sp)))
```

```
Accuracy (One v All): 0.83951
```

```
confusionchart(y_test_sp, pred_onevall, 'ColumnSummary','column-normalized','RowSummary','row-normalized');
title('One vs All')
```



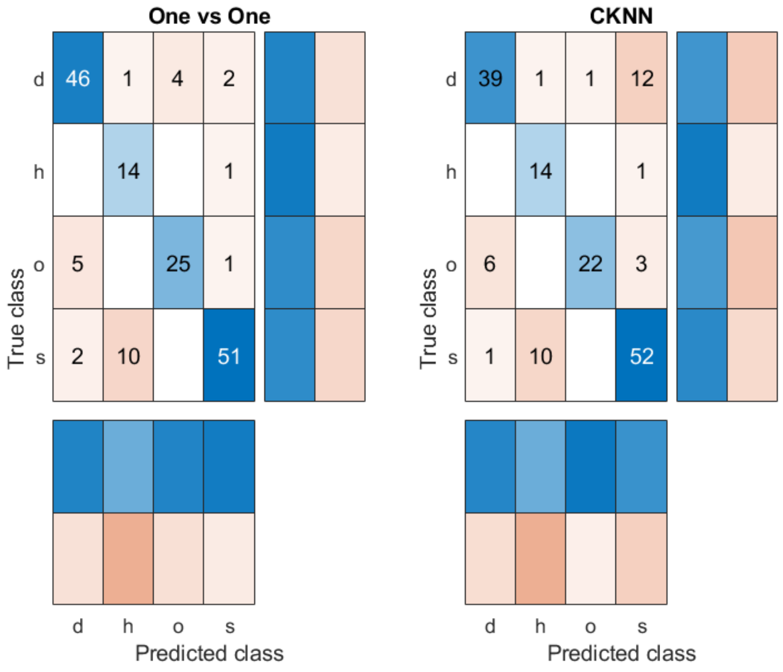
```
figure
subplot(1,2,1)
cat_y_test_sp = categorical(y_test_sp);
pred_onevone = svm_onevone.predict(x_test_sp);
disp("Accuracy (One v One): " + string(sum(pred_onevone == cat_y_test_sp) / length(cat_y_test_sp)))
```

```
Accuracy (One v One): 0.83951
```

```
 confusionchart(y_test_sp, pred_onevone, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
 title('One vs One')
 subplot(1,2,2)
 pred_CKNN = cknn_sp.predict(x_test_sp);
 disp("Accuracy (One v All): " + string(sum(pred_CKNN == cat_y_test_sp) /
length(cat_y_test_sp)))
```

```
Accuracy (One v All): 0.78395
```

```
 confusionchart(y_test_sp, pred_CKNN, 'ColumnSummary','column-
normalized','RowSummary','row-normalized');
 title('CKNN')
```


One vs One
True class
d
h
o
s
d
h
o
s
Predicted class
CKNN
True class
d
h
o
s
d
h
o
s
Predicted class