# Letter Classification

## Data Import

```
traindata = readtable("Data\traindata.xlsx");
testdata = readtable("Data\testdata.xlsx");

testfiles = readcell("Data\testFiles.xlsx");
```

# Models Evaluation

## kNN

### Simple kNN

```
kNNmodel = fitcknn(traindata,"Character");

% Model Evaluation
loss(kNNmodel,testdata)
```

```
ans = 0.2254
```

#### Weighted kNN

```
kNNmodel = fitcknn(traindata, "Character",'Distance', 'Cityblock', 'Exponent', [],
...
    'NumNeighbors', 6, 'DistanceWeight', 'SquaredInverse', 'Standardize', true);

% Model Evaluation
loss(kNNmodel,testdata)
```

```
ans = 0.1511
```

## Bagged Trees

```
rng(123)
TBmodel =
fitcensemble(traindata,"Character","Method","Bag","Learners","tree","NumLearningCycl
es",30);

% Model Evaluation
loss(TBmodel,testdata)
```

```
ans = 0.1407
```

## Neural Networks

### Wide Neural Network

```
modelNN = fitcnet(traindata,"Character",'LayerSizes', 170,'Activations', 'relu', ...
    'Lambda',0,'IterationLimit', 1000,'Standardize', true);

% Model Evaluation
loss(modelNN,testdata)
```

```
ans = 0.1558
```

# SVM

### Linear SVM

```
template = templateSVM('KernelFunction', 'linear', 'PolynomialOrder', [], ...
    'KernelScale', 'auto', 'BoxConstraint', 1, 'Standardize', true);

linearSVM = fitcecoc(traindata,"Character",'Learners',template,'Coding','onevsone');

% Model Evaluation
loss(linearSVM,testdata)
```

```
ans = 0.1505
```

### Quadratic SVM

```
template = templateSVM('KernelFunction','polynomial','PolynomialOrder',2, ...
    'KernelScale','auto','BoxConstraint',1,'Standardize',true);

quadraticSVM =
fitcecoc(traindata,"Character",'Learners',template,'Coding','onevsone');

% Model Evaluation
loss(quadraticSVM,testdata)
```

```
ans = 0.1132
```

### Cubic SVM

```
template = templateSVM('KernelFunction','polynomial','PolynomialOrder',3, ...
    'KernelScale','auto','BoxConstraint', 1,'Standardize', true);

cubicSVM = fitcecoc(traindata,"Character",'Learners',template,'Coding','onevsone');

% Model Evaluation
loss(cubicSVM,testdata)
```

```
ans = 0.1146
```

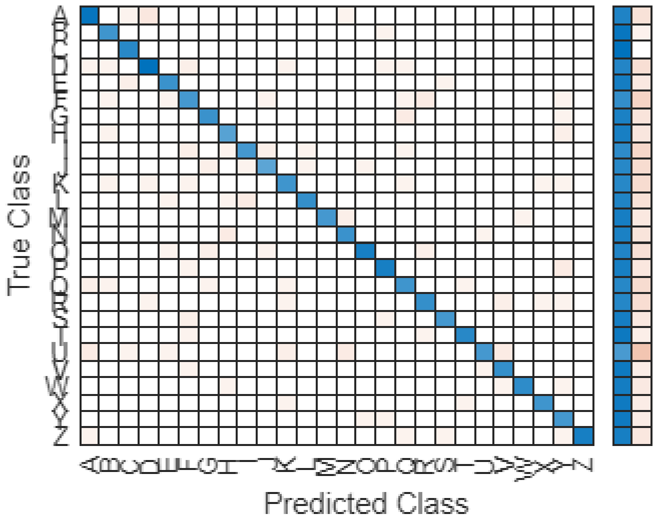
# Best Models Evaluation

```
% Testing; best models under that category
 model = cubicSVM
model = 
  ClassificationECOC
           PredictorNames: {1×25 cell}
             ResponseName: 'Character'
    CategoricalPredictors: []
               ClassNames: {'A'   'B'   'C'   'D'   'E'   'F'   'G'   'H'   'I'   'J'   'K'   'L'   'M'   'N'   'O'   'P'   'Q'   'R
           ScoreTransform: 'none'
           BinaryLearners: {325×1 cell}
               CodingName: 'onevsone'


  Properties, Methods
```

```
predLetter = predict(model, testdata);

% accuracy
accurate = nnz(predLetter == string(testdata.Character)) / numel(testdata.Character)
```

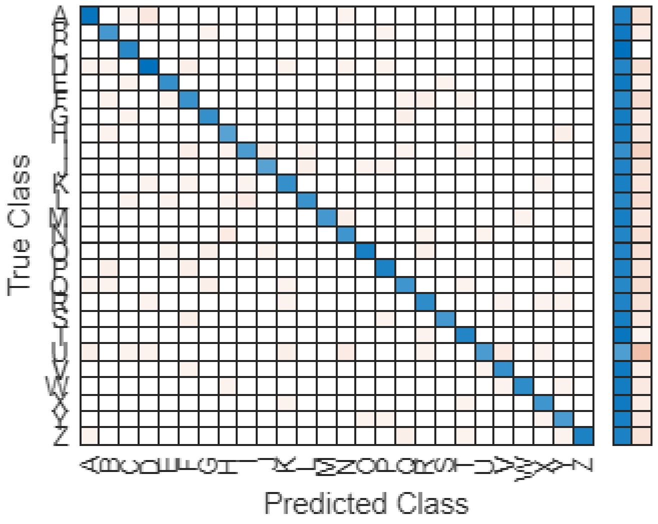
```
accurate = 0.8812
```

```
% Confusion Matrix
confusionchart(testdata.Character,predLetter, "RowSummary","row-normalized");
```



## Best Result : Quadratic SVM => 88.22%

```
% best prediction on test cases by Quadratic SVM
predLetter = predict(quadraticSVM, testdata);

accuracy = nnz(predLetter == string(testdata.Character)) / 
numel(testdata.Character);

confusionchart(testdata.Character,predLetter, "RowSummary","row-normalized");
```



## Analyze misclassifications

### Confusion matrix

```
cm = confusionmat(testdata.Character,predLetter);

% Split into correct and incorrect classifications
yes = diag(cm);
no = cm - diag(yes);
```

### Misclassification rate for each letter

```
misratebyletter = sum(no,2) ./ sum(cm,2);
```

### Table with letter names and misclassification rate

```
letters = categories(categorical(traindata.Character));
misratebyletter = table(letters,misratebyletter,'VariableNames',
["Letter","MisClassRate"]);

% Sort by worst misclassification
misratebyletter = sortrows(misratebyletter,"MisClassRate","descend")
```

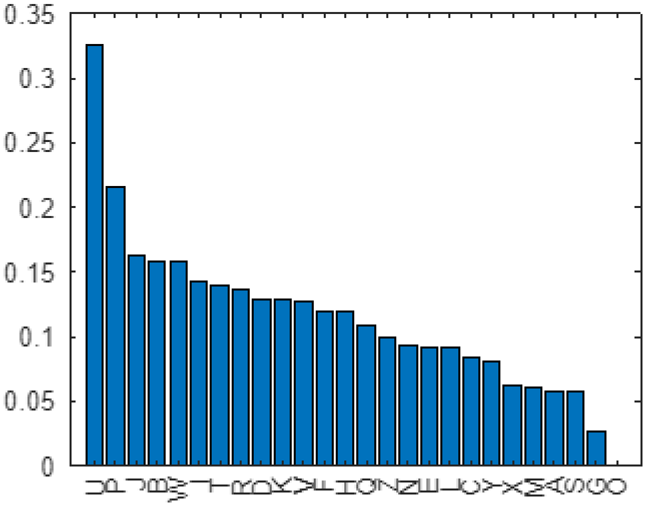
misratebyletter = 26×2 table

| | Letter | MisClassRate |
|---|---|---|
| 1 | 'U' | 0.3256 |
| 2 | 'P' | 0.2162 |
| 3 | 'J' | 0.1622 |
| 4 | 'B' | 0.1579 |
| 5 | 'W' | 0.1579 |
| 6 | 'I' | 0.1429 |

|  | Letter | MisClassRate |
|---|---|---|
| 7 | 'T' | 0.1389 |
| 8 | 'R' | 0.1364 |
| 9 | 'D' | 0.1290 |
| 10 | 'K' | 0.1282 |
| 11 | 'V' | 0.1277 |
| 12 | 'F' | 0.1190 |
| 13 | 'H' | 0.1190 |
| 14 | 'Q' | 0.1081 |

⋮

```
bar(misratebyletter.MisClassRate)
xticks(1:26)
xticklabels(misratebyletter.Letter)
```



### Individual misclassification

```
letter = "K";
```

### True class that were misclassified as something else

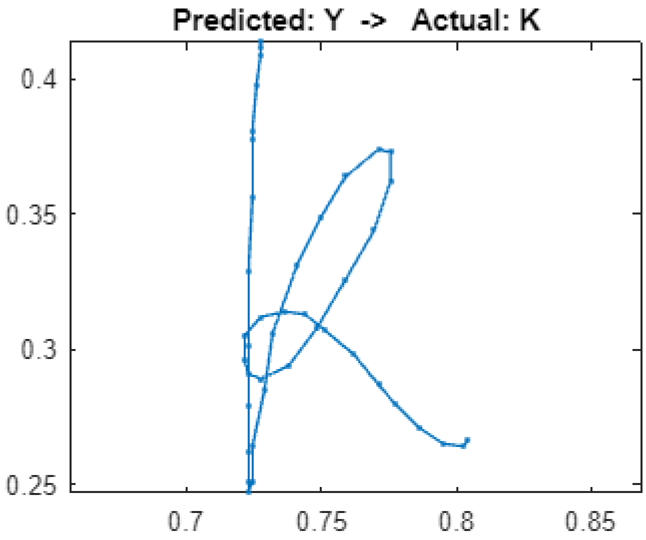
```
misclassidx = (string(testdata.Character) == letter) & (predLetter ~=
string(testdata.Character));
```

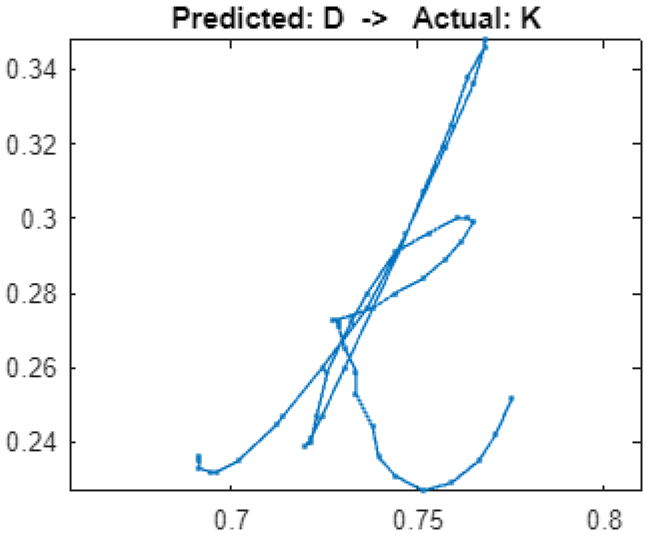
### Table of the misclassified observations, with the predicted letter

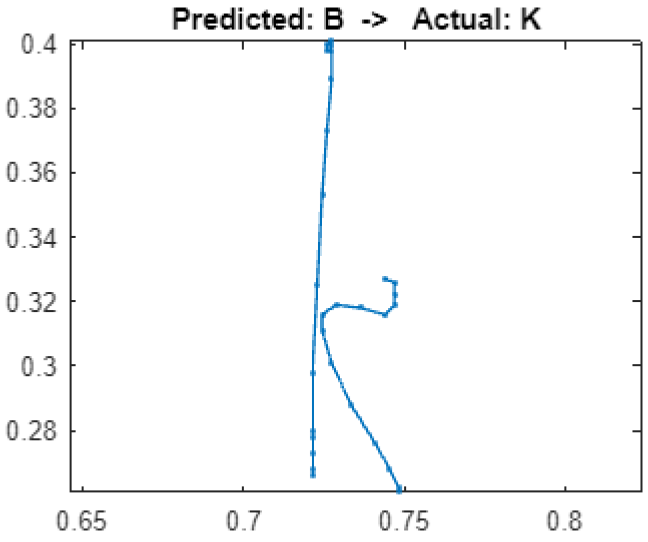
```
badpred = testdata(misclassidx,:);
badpred.Prediction = predLetter(misclassidx);
```

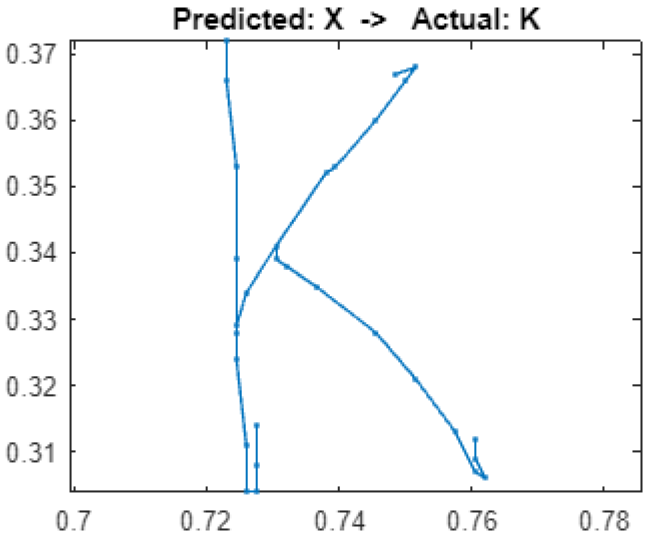
## Plot

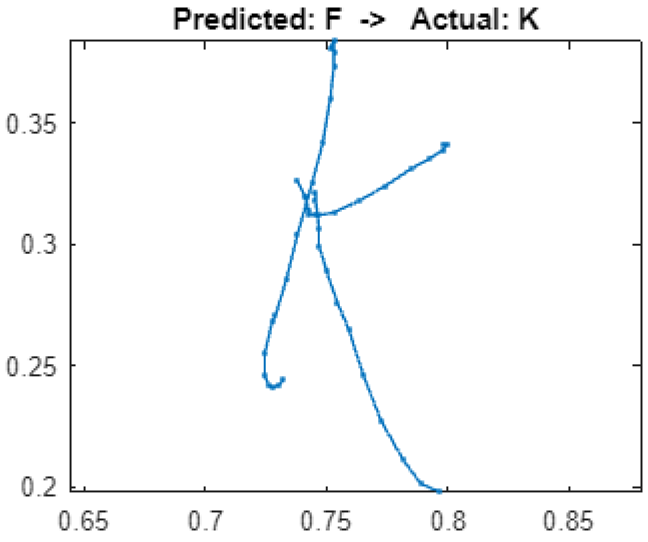
```
badfiles = testfiles(misclassidx);

for k = 1:numel(badfiles)
    badletter = readtable("Files"+filesep+badfiles(k));
    figure
    plot(1.5*badletter.X,badletter.Y,".-")
    title("Predicted: "+string(badpred.Prediction(k))+"  ->   Actual: "+letter)
    axis equal
end
```

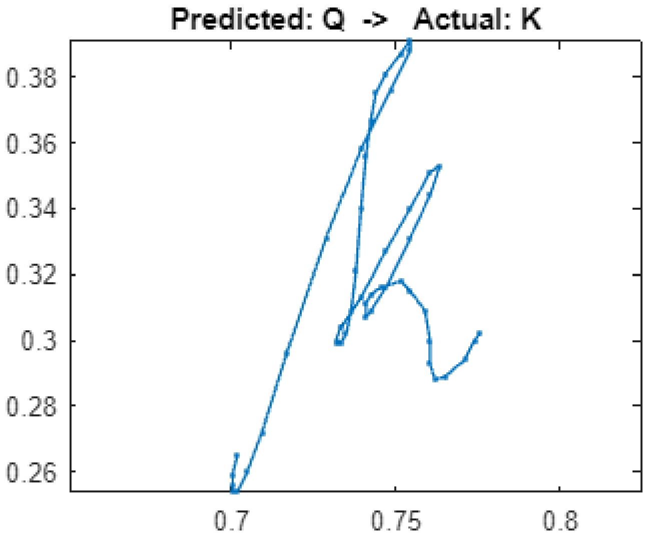





Predicted: B -> Actual: K
0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.65
0.7
0.75
0.8


Predicted: X -> Actual: K
0.37
0.36
0.35
0.34
0.33
0.32
0.31
0.7
0.72
0.74
0.76
0.78


Predicted: F -> Actual: K
0.35
0.3
0.25
0.2
0.65
0.7
0.75
0.8
0.85


Predicted: Q  ->   Actual: K
0.38
0.36
0.34
0.32
0.3
0.28
0.26
0.7
0.75
0.8