Devanagari Script OCR

This document contains the hit and trials of different classification strategies with results.

My dataset contains Handwritten Devanagari characters taken from UCI repository (https://archive.ics.uci.edu/ml/datasets/Devanagari+Handwritten+Character+Dataset). There are 46 classes of characters with 2000 examples each. The dataset is split into training set (85%) and testing set (15%). Each image is 32*32*1 binary image with black background.

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique. A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor. KNN is a widely used algorithm in OCRs so I thought to give it a try.

1.) k-NN(k-Nearest Neighbors)

I ran the k-NN algorithm with Euclidian Distance metric on 10000 training and 1000 testing images and the results are as follows -

```
model score for K = 1 is 0.813
model score for K = 3 is 0.79
model score for K = 5 is 0.782
model score for K = 7 is 0.782
model score for K = 9 is 0.771
model score for K = 11 is 0.766
model score for K = 13 is 0.755
model score for K = 15 is 0.756
model score for K = 17 is 0.754
model score for K = 19 is 0.743
```

It is a well-known fact that Euclidian distance has got the curse of dimensionality so it doesn't go well with high dimensional vector spaces which in this case is 32*32 = 1024 dimensional space.

So, it becomes necessary to change the Distance metric to achieve higher accuracy.

2.) k-NN(k-Nearest Neighbors) with PCA (Principal Component Analysis) and Euclidian Distance metric.

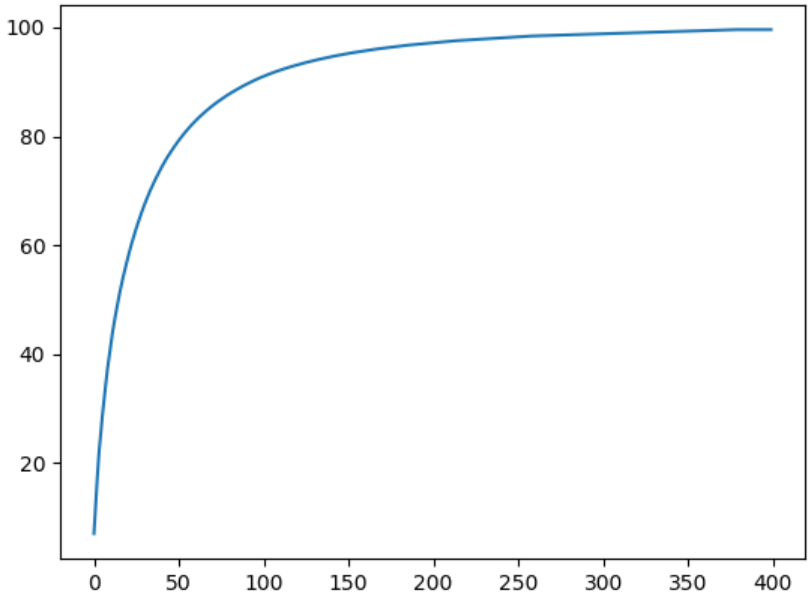
Because I was facing challenges due to the dimensionality I tried using a common dimensionality reduction algorithm PCA with retaining 400 dimensions out of 1024. The results are as follows -

```
Train shape after pca is (10000, 400)
Test shape after pca is (1000, 400)
model score for K = 1 is 0.057
model score for K = 3 is 0.062
model score for K = 5 is 0.065
model score for K = 7 is 0.072
model score for K = 9 is 0.062
model score for K = 11 is 0.062
model score for K = 13 is 0.068
model score for K = 15 is 0.071
model score for K = 17 is 0.076
model score for K = 19 is 0.079
```

It is evident that the approach didn't work as PCA can be useful in image analysis if you we want to find a simplified representation for a _set of images_, but it doesn't work well on a single image to reduce its dimension. The PCA dimensionality reduction imply a complete change of the feature space, the projection will be in fact a linear combination of the initial feature dimension depending of the variance of the data mainly.

Also, PCA doesn't care about the localization of features which is very crucial in OCR like tasks.

When I plotted the cumulative variance w.r.t components, I saw that even if 100 components were capturing around 90% of the variance but still it was not sufficient as there are some classes like □, □, □ etc which have very less variance between them so we can't afford to lose even 1% of variance.



So, I went on to use some algorithm which reduces the dimensions along with taking care about the localization.

3.) k-NN(k-Nearest Neighbors) with HOG(Histogram Oriented Gradients) Feature Descriptor and Euclidian Distance metric.

- A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. HOG is one such descriptor.
- Also HOG reduces the number of features so it can be used as a Dimensionality reduction algorithm.

Results are as follows-

```
Shape of Training data after HOG is: (10000, 128)
Shape of Testing data after HOG is: (1000, 128)
model score for K = 1 is 0.87
model score for K = 3 is 0.87
model score for K = 5 is 0.871
model score for K = 7 is 0.876
model score for K = 9 is 0.863
model score for K = 11 is 0.86
model score for K = 13 is 0.85
model score for K = 15 is 0.849
model score for K = 17 is 0.843
model score for K = 19 is 0.849
```

Optimum k value was 7 for which the model gave 87.6% accuracy.

The results are as expected superior to the previous approaches and it also took lesser time to train as the dimensions were reduced to 128 from 1024.

Then to improve the accuracy further I played with different Distance Metric and observed the results.

4.) k-NN(k-Nearest Neighbors) with HOG(Histogram Oriented Gradients) Feature Descriptor and Bray Curtis Distance metric.

- Bray Curtis Distance metric is used for multidimensional vector spaces. Its distance function is -

  D = sum(|x - y|) / (sum(|x|) + sum(|y|))

The results of the classifier with HOG Bray Curtis Distance metric are as follows -

```
Shape of Training data after HOG is: (10000, 128)
Shape of Testing data after HOG is: (1000, 128)
model score for K = 1 is 0.896
model score for K = 3 is 0.904
model score for K = 5 is 0.898
model score for K = 7 is 0.901
model score for K = 9 is 0.9
model score for K = 11 is 0.895
model score for K = 13 is 0.891
model score for K = 15 is 0.891
model score for K = 17 is 0.887
model score for K = 19 is 0.884
```

Optimum K value was 7 for which the model gave 90.1 % accuracy.

Bray Curtis Distance Metric seemed to give the best results as compared to the other Distance metrics(mahanabolis, jaccard etc.) that I tried.

To further increase the accuracy, I increased the data size which almost always helps. This time I trained it on 73600 images and tested it on 18400 images.

The results are as follows -

```
Shape of Training data is: (73600, 128)
Shape of Testing data is: (18400, 128)
model score for K = 1 is 0.9602173913043478
model score for K = 3 is 0.9615760869565217
model score for K = 5 is 0.9635869565217391
model score for K = 7 is 0.9628260869565217
model score for K = 9 is 0.9621195652173913
model score for K = 11 is 0.9607608695652174
model score for K = 13 is 0.959945652173913
```

Adding more examples, adds diversity. It decreases the generalization error because our model becomes more general by virtue of being trained on more examples.

So with K = 5 and more training examples I got the maximum accuracy of 96.358% on 18400 testing images each of which was different from the training images.

SVC(Support Vector Classifier)

Support Vector Machine is a supervised learning method for classification and regression analysis. This algorithm output the optimal hyper-plane and maximizes the margin between two classes. For overcome the problem of non-linear data, SVM has the kernel trick that can obtain the better accuracy. There are several most common kernels for SVM. The simplest kernel is Linear Kernel. I will use ‘one vs rest’ multiclass classification stratergy

I ran the Support vector classifier with Linear Kernel and C=1 on 20000 training and 8000 testing images. The precision ,recall and f1 scores for each class are as follows-:

	precision	recall	f1-score	support
1	0.84	0.86	0.85	180
2	0.68	0.69	0.69	175
3	0.72	0.74	0.73	180
4	0.59	0.67	0.63	184
5	0.63	0.76	0.69	177
6	0.73	0.86	0.79	175
7	0.69	0.67	0.68	177
8	0.79	0.82	0.80	175
9	0.90	0.80	0.85	171
10	0.80	0.79	0.79	177
11	0.82	0.93	0.87	181
12	0.84	0.83	0.83	205
13	0.76	0.72	0.74	187
14	0.83	0.88	0.85	190
15	0.78	0.85	0.81	171
16	0.76	0.85	0.80	169
17	0.61	0.62	0.62	150
18	0.68	0.66	0.67	173
19	0.65	0.73	0.69	146
20	0.75	0.76	0.76	189
21	0.62	0.75	0.68	166
22	0.82	0.85	0.84	164
23	0.74	0.58	0.65	168
24	0.73	0.73	0.73	159
25	0.67	0.61	0.64	163
26	0.64	0.46	0.54	170
27	0.84	0.83	0.83	157
28	0.83	0.81	0.82	166
29	0.69	0.61	0.64	188
30	0.72	0.70	0.71	189
31	0.79	0.74	0.76	169
32	0.63	0.57	0.60	169
33	0.78	0.72	0.75	160
34	0.84	0.70	0.76	155
35	0.72	0.72	0.72	170
36	0.82	0.82	0.82	159
37	0.95	0.98	0.96	191
38	0.90	0.96	0.93	178
39	0.74	0.83	0.78	159
40	0.87	0.76	0.81	201
41	0.91	0.87	0.89	170
42	0.90	0.92	0.91	167
43	0.93	0.87	0.90	195
44	0.96	0.96	0.96	180
45	0.93	0.92	0.92	178
46	0.90	0.89	0.89	177

Overall accuracy – 77.63%

Here, we can see that the classifier is doing much better on some classes than others. Overall it is doing better on Devanagari numerals (Class: 37-46) than on Devanagari alphabets (Class: 1-36).

To get more intuitions about the classes on which the classifier is having a hard time classifying between, I calculated the confusion matrix -:

154 2 0 0 1 0 2 0 2 0 0 0 0 1 0 0 0 1 0 0 2 1 2 0 2 0 0 0 2 0 0 5 0 0 0 0 0 1 2 0 0 0 0 0 0 0

2 121 1 1 0 2 2 1 2 0 0 0 0 0 2 2 0 0 2 1 1 0 5 2 1 2 1 0 0 2 2 9 1 0 2 1 0 1 0 0 0 4 0 0 0 2

0 2 133 0 1 0 0 0 0 1 0 1 0 0 **11** 1 2 1 0 0 1 0 0 0 1 3 0 0 5 **11** 1 3 0 0 0 1 0 0 0 0 0 0 0 0 0 1

0 1 0 123 2 1 5 0 0 0 0 0 0 0 0 1 1 8 1 **11** 2 11 0 2 1 1 5 1 0 2 0 2 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0

1 2 1 2 134 0 9 1 0 0 1 0 **13** 0 0 2 0 5 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 1

1 1 0 0 0 151 0 2 0 3 0 0 1 0 0 0 3 0 0 1 6 0 0 0 0 0 0 0 0 1 0 1 0 0 0 2 1 0 0 0 0 0 0 0 1 0 0

1 0 1 9 **14** 0 118 0 0 2 0 0 3 0 0 0 1 7 0 0 5 0 0 0 1 2 2 1 3 0 1 0 1 0 3 0 0 0 0 0 0 0 1 0 0 1

1 1 1 0 1 0 0 143 0 8 0 0 0 0 0 0 1 0 0 0 2 0 0 0 0 0 0 0 0 4 1 0 0 0 0 0 0 4 6 0 0 0 0 2 0 0 0 0 0

5 1 0 0 0 1 1 0 137 1 0 0 0 0 0 0 0 0 0 0 0 7 1 8 2 1 0 1 0 0 0 0 2 1 2 0 0 0 0 0 0 0 0 0 0 0 0

0 0 2 0 1 1 0 6 0 140 0 0 1 0 1 4 0 0 0 4 0 0 0 3 0 0 0 1 1 0 0 3 0 1 6 2 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 169 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 6 0

1 0 0 0 4 0 1 0 0 0 9 170 3 0 0 0 1 2 0 0 0 1 2 0 0 0 0 0 1 0 0 0 7 0 0 0 1 0 0 0 1 0 0 0 1 0

0 1 2 0 **23** 2 0 4 0 0 0 2 134 3 0 0 0 0 0 2 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 1 2 1 0 1 0 1 0 0 0 0 0 4

0 0 0 0 2 0 2 0 0 0 1 6 2 167 0 0 0 5 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 1 0

0 1 4 3 0 1 0 0 0 0 0 0 1 0 145 0 0 1 2 0 6 0 0 0 1 5 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 143 0 0 0 3 0 1 0 0 1 0 0 6 7 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1

0 **1 1 7 1 6 2** 0 0 0 0 1 1 0 4 0 93 0 **16** 0 4 0 0 5 0 6 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0

1 0 0 3 8 0 6 1 0 0 8 1 0 **14** 0 0 0 114 0 0 1 1 1 0 1 0 1 0 1 0 0 1 2 0 0 0 1 0 1 1 0 0 2 0 3 0

0 1 1 **11** 1 2 0 0 0 0 0 0 0 1 0 0 0 **10** 0 107 1 1 0 0 0 0 0 0 0 1 2 0 0 0 1 1 0 3 0 0 0 0 0 0 1 0 1

0 0 0 0 1 5 0 4 1 0 0 0 1 0 0 7 1 1 0 144 0 1 3 0 8 0 0 0 2 6 0 0 0 1 0 2 0 0 0 1 0 0 0 0 0 0

0 0 1 **12** 2 4 2 0 0 0 0 0 0 0 0 3 0 0 0 1 0 125 1 0 0 0 6 2 0 0 0 5 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0

2 0 1 0 0 0 1 0 2 0 0 0 0 0 0 0 0 0 0 0 6 140 0 0 1 0 1 0 0 0 7 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0

5 3 0 **5 1 3 4 1** 0 1 0 **1 1 7** 0 0 2 **10** 6 1 0 2 98 0 **1 1 1 3 4** 0 0 0 **2 1 1 1** 0 0 0 0 1 0 0 0 0 1

0 0 1 2 1 0 0 0 4 3 0 0 0 0 0 3 3 0 3 0 3 5 0 116 1 2 0 1 0 1 2 3 0 2 2 0 0 0 0 0 1 0 0 0 0 0

3 5 1 2 0 0 0 0 0 1 0 1 0 0 **2 2 1 1 2 7** 0 **3 2 4** 99 2 0 0 0 **2 1 17 2 1** 0 1 0 0 0 1 0 0 0 0 0 0

0 0 **11 8** 0 6 0 0 0 1 2 4 2 0 6 0 **11** 1 1 0 **12** 1 0 3 3 79 1 1 2 2 6 1 0 0 1 0 0 0 0 0 2 0 0 0 0 3

0 0 4 3 0 2 0 0 0 0 5 1 2 0 0 0 1 0 0 0 1 0 0 0 0 1 130 0 0 1 0 1 0 0 0 1 0 0 1 1 0 1 0 0 1 0

0 0 0 0 0 2 0 2 0 2 1 0 0 0 0 7 0 0 0 8 0 0 1 2 0 0 0 135 3 0 0 0 0 0 2 0 0 0 0 0 0 1 0 0 0 0

2 5 2 1 0 3 0 4 0 2 0 9 1 2 1 4 0 6 0 0 5 0 5 1 2 2 0 2 114 1 1 3 1 2 3 0 1 1 0 0 0 0 0 1 0 1

2 3 9 2 2 2 1 1 0 0 0 0 0 0 0 **1 9 2 4 3** 0 0 0 0 3 2 2 0 1 133 1 3 0 1 1 1 0 0 0 0 0 0 0 0 0 0

0 7 3 0 1 0 1 0 0 0 0 0 0 0 2 0 2 2 0 0 9 1 0 2 0 1 0 0 1 2 125 1 0 0 3 0 0 5 0 0 0 0 1 0 0 0

1 6 1 5 1 2 3 0 1 0 0 0 0 0 1 1 1 0 **1 2 1 2 2 3 14 2** 0 **3 4 3** 0 96 **3 1 4 2** 0 0 0 1 0 1 1 0 0 0

1 0 0 **5 5 1 4 1** 0 0 3 2 2 0 0 2 0 2 1 0 0 0 3 0 0 0 2 1 0 1 1 1 115 1 0 2 0 0 0 1 2 0 1 0 0 0

0 4 1 1 0 0 1 1 0 1 0 1 0 0 1 2 1 0 0 3 0 0 1 4 4 0 0 0 2 7 0 3 0 109 2 4 0 0 0 0 1 0 1 0 0 0

0 3 1 0 2 3 0 4 0 8 0 0 0 0 1 6 0 2 1 1 0 1 1 1 0 1 0 1 4 1 0 1 1 1 122 0 0 1 0 0 0 0 1 0 0 1

0 0 0 0 0 3 0 2 0 2 0 0 1 1 0 0 0 0 0 4 0 0 2 0 0 0 0 1 1 5 0 0 0 3 1 130 0 0 0 1 0 2 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 187 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 2 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 170 1 0 0 0 1 0 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0 2 0 0 1 0 0 1 0 0 0 0 0 0 6 0 0 0 0 1 0 0 0 0 0 1 132 13 0 1 0 0 0 0

0 3 0 0 0 1 0 2 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 37 152 0 0 1 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 2 0 0 0 1 0 0 1 1 1 0 0 0 1 2 1 1 0 2 1 0 0 0 2 0 0 148 3 0 0 0 0

0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 1 0 0 0 1 1 2 154 1 0 0 0

0 0 0 0 2 0 3 0 2 0 0 0 0 0 0 0 1 2 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 1 0 2 170 4 0 1

0 1 0 1 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 172 0 0

0 0 0 2 2 1 0 0 0 0 6 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 163 0

0 0 1 0 1 2 1 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 3 0 1 2 1 0 0 0 0 0 0 0 157

The points of interest are shown in bold.

Going through the confusion matrix I found that there are some pairs which the algorithm is classifying incorrectly and the classifier is not doing so good on some particular classes.

- Class 3(□) is mostly misclassified with Class 15(□) and 30(□).
- Class 4(□) is mostly misclassified with Class 19(□).
- Class 5(□) is mostly misclassified with Class 13(□).
- Class 7(□) is mostly misclassified with Class 5(□).
- Class 13(□) is mostly misclassified with Class 5(□).

- Class 17(□) is mostly misclassified with Class 19(□), although the classifier is having a hard time classifying Class 17(□) overall as it shows multiple errors with different classes.
- Class 18(□) is mostly misclassified with Class 14(□).
- Class 19(□) is mostly misclassified with Class 17(□) and 4(□).
- Class 21(□) is mostly misclassified with Class 4(□).
- Classifier is showing poor results on Class 23(□) and misclassifies it with many classes mostly with class 29(□).
- Class 25(□) is mostly misclassified with Class 32(□) and class 24(□), although the classifier is having a hard time classifying Class 25(□) overall as it shows multiple errors with different classes.
- Class 26(□) is mostly misclassified with Class 3(□),16(□) and 20(□), although the classifier is having a hard time classifying class 26(□) overall as it shows multiple errors with different classes.
- Multiple errors in classifying Class 30(□).
- Class 32(□) is mostly misclassified with Class 25(□), although the classifier is having a hard time classifying class 32(□) overall as it shows multiple errors with different classes.

Thereafter, driven by the past results and experience of using HOG (Histogram Oriented Gradients) feature extractor to achieve higher accuracy on k-NN, I gave it a try reducing the dimensions to 128(16 cells with 8 orientations each) and ran SVC with linear kernel and C=1 and got the following results :-

	precision	recall	f1-score	support
1	0.82	0.77	0.80	180
2	0.46	0.70	0.56	175
3	0.86	0.73	0.79	180
4	0.59	0.46	0.52	184
5	0.71	0.51	0.59	177
6	0.79	0.73	0.76	175
7	0.48	0.69	0.57	177
8	0.74	0.70	0.72	175
9	0.80	0.77	0.79	171
10	0.80	0.80	0.80	177
11	0.62	0.81	0.70	181
12	0.69	0.68	0.68	205
13	0.70	0.72	0.71	187
14	0.65	0.52	0.57	190
15	0.91	0.74	0.81	171
16	0.68	0.67	0.67	169
17	0.67	0.69	0.68	150
18	0.69	0.51	0.58	173
19	0.62	0.79	0.69	146
20	0.66	0.50	0.57	189
21	0.64	0.72	0.68	166
22	0.84	0.82	0.83	164
23	0.67	0.69	0.68	168
24	0.68	0.74	0.71	159
25	0.54	0.67	0.60	163
26	0.56	0.55	0.56	170
27	0.86	0.84	0.85	157
28	0.63	0.67	0.65	166
29	0.55	0.48	0.51	188
30	0.69	0.57	0.62	189
31	0.80	0.79	0.80	169
32	0.48	0.44	0.46	169
33	0.67	0.52	0.59	160
34	0.45	0.81	0.58	155
35	0.87	0.69	0.77	170
36	0.73	0.69	0.71	159
37	0.98	0.98	0.98	191
38	0.91	0.93	0.92	178
39	0.73	0.78	0.75	159
40	0.79	0.74	0.76	201
41	0.98	0.89	0.93	170
42	0.90	0.95	0.92	167
43	0.79	0.86	0.82	195
44	0.97	0.97	0.97	180
45	0.91	0.85	0.88	178
46	0.88	0.90	0.89	177

The overall accuracy was 71.68%, which is worse than the SVC ran alone.

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, we should get misclassified examples, often even if our training data is linearly separable.

So I increased the value of C parameter and gradually started getting better results. After reaching at C = 100, the improvement stabilized.

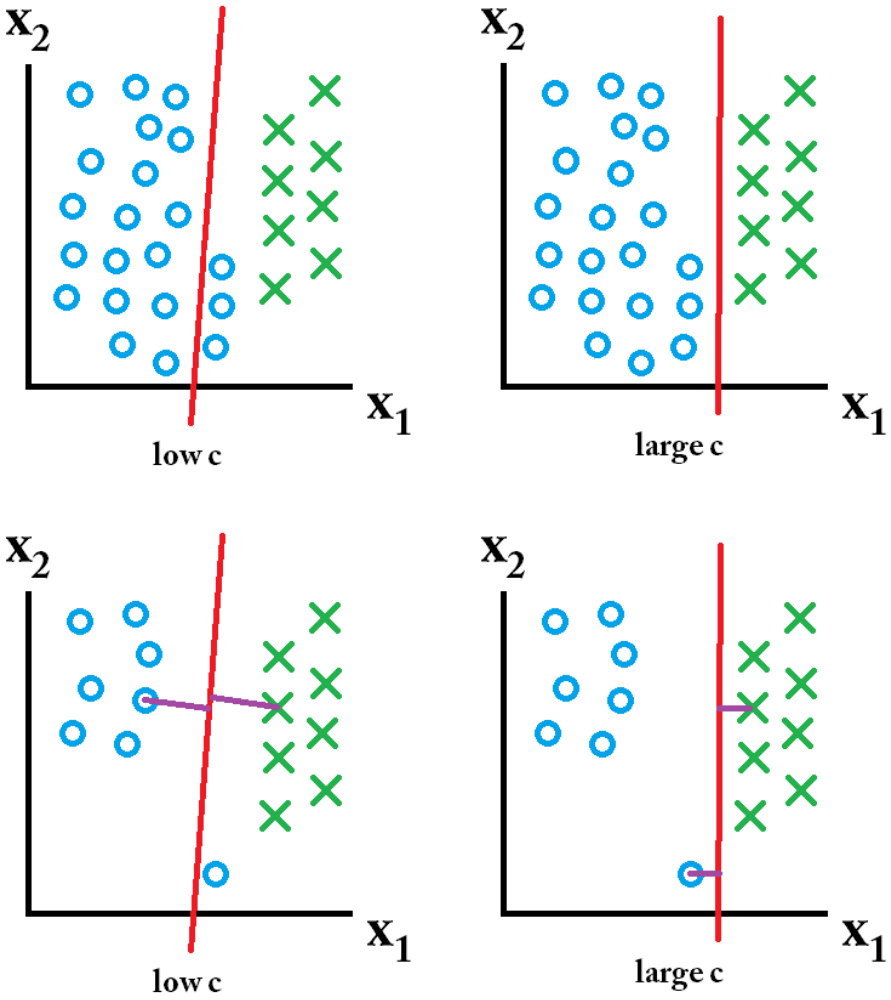
The results at C = 100 with linear kernel and SVC with HOG are as follows :-

	precision	recall	f1-score	support
1	0.96	0.97	0.96	180
2	0.89	0.93	0.91	175
3	0.94	0.94	0.94	180
4	0.81	0.83	0.82	184
5	0.93	0.88	0.90	177
6	0.91	0.93	0.92	175
7	0.85	0.90	0.87	177
8	0.95	0.93	0.94	175
9	0.95	0.96	0.95	171
10	0.91	0.94	0.93	177
11	0.93	0.94	0.94	181
12	0.93	0.96	0.94	205
13	0.88	0.94	0.91	187
14	0.86	0.88	0.87	190
15	0.94	0.98	0.96	171
16	0.90	0.92	0.91	169
17	0.77	0.85	0.80	150
18	0.88	0.83	0.85	173
19	0.88	0.92	0.90	146
20	0.90	0.90	0.90	189
21	0.92	0.93	0.93	166
22	0.98	0.96	0.97	164
23	0.87	0.92	0.89	168
24	0.89	0.91	0.90	159
25	0.93	0.94	0.94	163
26	0.81	0.75	0.78	170
27	0.98	0.92	0.95	157
28	0.94	0.86	0.90	166
29	0.85	0.84	0.84	188
30	0.97	0.87	0.92	189
31	0.94	0.96	0.95	169
32	0.87	0.88	0.88	169
33	0.92	0.88	0.90	160
34	0.93	0.87	0.90	155
35	0.94	0.86	0.90	170
36	0.92	0.91	0.92	159
37	0.99	1.00	1.00	191
38	0.99	0.99	0.99	178
39	0.94	0.96	0.95	159
40	0.95	0.96	0.95	201
41	0.99	0.99	0.99	170
42	0.98	0.98	0.98	167
43	0.96	0.95	0.96	195
44	0.99	0.99	0.99	180
45	0.97	0.96	0.96	178
46	0.99	0.99	0.99	177

Overall accuracy = 92.135%

Evidently, increasing C helped as expected.

These diagrams explains why it happened -:



Because the data points, even in 128-dimensional space are close to each other due to high similarity between some of the Devanagari characters, increasing the Cost for misclassifying decreased the misclassified examples and increased accuracy and also helped the algorithm generalize better.

Motivated by the results, I played with HOG changing the number of cells and orientations per cell. Taking 64 cells with 8 orientations each, which increased the features to 64*8 = 512, I got the superior results which are as follows -:

	precision	recall	f1-score	support
1	0.98	0.96	0.97	180
2	0.92	0.93	0.92	175
3	0.94	0.95	0.95	180
4	0.82	0.90	0.86	184
5	0.91	0.89	0.90	177
6	0.94	0.93	0.93	175
7	0.88	0.92	0.90	177
8	0.95	0.91	0.93	175
9	0.97	0.95	0.96	171
10	0.96	0.97	0.96	177
11	0.94	0.94	0.94	181
12	0.97	0.96	0.96	205
13	0.92	0.92	0.92	187
14	0.91	0.94	0.92	190
15	0.95	0.98	0.97	171
16	0.89	0.92	0.90	169
17	0.84	0.85	0.84	150
18	0.86	0.85	0.86	173
19	0.93	0.95	0.94	146
20	0.89	0.89	0.89	189
21	0.94	0.94	0.94	166
22	0.96	0.98	0.97	164
23	0.87	0.89	0.88	168
24	0.93	0.92	0.92	159
25	0.94	0.93	0.94	163
26	0.85	0.83	0.84	170
27	0.97	0.92	0.94	157
28	0.94	0.91	0.93	166
29	0.82	0.81	0.82	188
30	0.96	0.92	0.94	189
31	0.93	0.98	0.95	169
32	0.93	0.91	0.92	169
33	0.92	0.91	0.92	160
34	0.96	0.95	0.95	155
35	0.96	0.91	0.93	170
36	0.91	0.94	0.93	159
37	0.99	1.00	1.00	191
38	0.98	1.00	0.99	178
39	0.98	0.97	0.98	159
40	0.97	0.97	0.97	201
41	0.99	0.99	0.99	170
42	0.98	0.99	0.99	167
43	0.96	0.96	0.96	195
44	1.00	0.99	0.99	180
45	0.95	0.96	0.96	178
46	0.98	0.97	0.98	177

Overall accuracy – 93.425%

Increasing the number of cells allows the algorithm to capture more fine details within the same region which in-turn increases the accuracy.

After optimizing the hyperparameters to their best, I went for increasing the training set from 20000 images to 70000 images and test set from 8000 images to 8000 images. Expectedly, the accuracy showed the uptrend.

The results of SVC (Linear kernel and C = 100) and HOG (Cells = (8,8) and 8 orientations each cell) on 70000 training and 8000 testing images (Each of which was unique and distinct from training set) are as follows -:

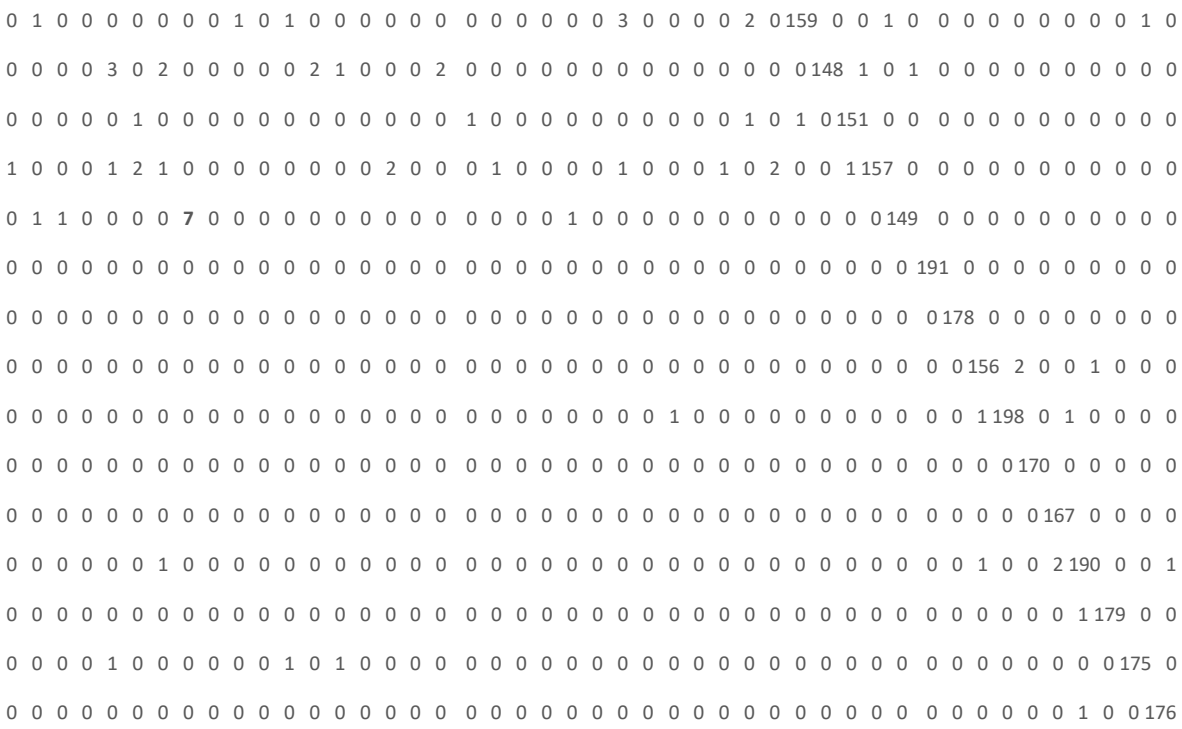
	precision	recall	f1-score	support
1	0.98	0.98	0.98	180
2	0.95	0.97	0.96	175
3	0.96	0.99	0.98	180
4	0.86	0.94	0.90	184
5	0.94	0.95	0.94	177
6	0.95	0.94	0.94	175
7	0.96	0.94	0.95	177
8	0.95	0.98	0.96	175
9	0.97	0.98	0.97	171
10	0.97	0.99	0.98	177
11	0.99	0.97	0.98	181
12	0.98	0.97	0.98	205
13	0.95	0.97	0.96	187
14	0.96	0.97	0.97	190
15	0.97	0.98	0.97	171
16	0.91	0.95	0.93	169
17	0.87	0.87	0.87	150
18	0.93	0.92	0.93	173
19	0.95	0.97	0.96	146
20	0.95	0.94	0.94	189
21	0.94	0.96	0.95	166
22	0.98	0.98	0.98	164
23	0.94	0.92	0.93	168
24	0.99	0.94	0.96	159
25	0.94	0.95	0.95	163
26	0.93	0.86	0.90	170
27	0.99	0.96	0.97	157
28	0.99	0.95	0.97	166
29	0.89	0.92	0.91	188
30	0.97	0.94	0.96	189
31	0.96	0.98	0.97	169
32	0.94	0.94	0.94	169
33	0.95	0.93	0.94	160
34	0.99	0.97	0.98	155
35	0.99	0.92	0.95	170
36	0.96	0.94	0.95	159
37	1.00	1.00	1.00	191
38	0.99	1.00	1.00	178
39	0.98	0.98	0.98	159
40	0.98	0.99	0.98	201
41	1.00	1.00	1.00	170
42	0.98	1.00	0.99	167
43	0.98	0.97	0.98	195
44	1.00	0.99	1.00	180
45	0.98	0.98	0.98	178
46	0.98	0.99	0.99	177

Overall accuracy = 96.12%

The final confusion matix looks much better -:

Confusion matrix:

177	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	169	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	178	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	173	0	1	0	0	0	0	0	0	0	0	0	0	0	0	3	0	1	0	0	0	0	2	0	0	3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	168	0	0	0	0	0	0	0	**7**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	3	0	164	0	0	1	0	0	0	0	0	0	0	2	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	4	2	0	166	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	1	0	0	0	0	171	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	168	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	175	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	176	0	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	199	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	2	0	0	0	0	181	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1	0	0	185	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	167	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	161	0	0	0	3	0	0	0	0	1	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	**8**	0	0	0	0	0	0	0	0	0	0	2	0	130	0	2	0	0	0	0	1	0	6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	0	2	0	0	0	0	1	0	3	0	0	0	160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	2	0	2	0	0	0	0	0	0	0	0	0	0	1	0	141	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	6	0	0	0	177	0	0	1	0	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	0	159	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	154	0	0	0	0	0	**9**	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	150	2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	155	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	5	0	0	0	0	0	1	0	0	0	0	0	0	**15**	0	0	0	2	0	0	0	0	147	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	2	0	0	0	0	0	0	0	157	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	**6**	0	0	2	0	0	173	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2	2	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	178	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	166	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Class 26(□) is mostly misclassified with Class 16(□) and Class 4(□) otherwise there are no severe misclassifications made by the final model.

Although I was getting acceptable results with linear kernel, still I tried using rbf(Radial Basis Function) kernel with different values of gamma ranging from 10e-5 to 10e0. The results were no superior to that of linear kernel and also it took much long to train.

If the number of features is large, one may not need to map data to a higher dimensional space as the points are already linearly separable in 512-dimensional feature space. That is, the nonlinear mapping does not improve the performance. Using the linear kernel is good enough, and one only searches for the parameter C.

Decision Tree - :

Decision Tree influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used

tool in data mining for deriving a strategy to reach a particular goal, its also widely used in machine learning.

I ran Decision Tree on ‘balanced’ class weight for the information gain with min_samples_leaf =3 and 180 sample feature and max depth of 15 and got an accuracy of 51.5 % as the accuracy is not up to the mark, I didn't find the need of precision and recall nor the need of confusion matrix

Thereafter, driven by the past results and experience of using HOG (Histogram Oriented Gradients) feature extractor to achieve higher accuracy on SVM, I gave it a try reducing the dimensions to 128(16 cells with 8 orientations each) and ran Decision Tree with max depth of 15 and got the following results :-

	precision	recall	f1-score	support
1	0.51	0.57	0.54	186
2	0.50	0.54	0.52	178
3	0.55	0.61	0.58	176
4	0.43	0.46	0.45	167
5	0.61	0.52	0.56	166
6	0.57	0.54	0.55	177
7	0.41	0.37	0.39	180
8	0.61	0.53	0.57	177
9	0.68	0.62	0.64	156
10	0.62	0.63	0.63	168
11	0.62	0.63	0.62	166
12	0.67	0.68	0.67	194
13	0.62	0.63	0.63	175
14	0.55	0.64	0.59	166
15	0.70	0.62	0.65	170
16	0.61	0.62	0.62	190
17	0.44	0.46	0.45	155
18	0.36	0.34	0.35	178
19	0.59	0.45	0.51	161
20	0.48	0.49	0.49	152
21	0.58	0.62	0.60	185
22	0.74	0.70	0.72	165
23	0.37	0.38	0.37	170
24	0.54	0.50	0.52	160
25	0.44	0.49	0.46	165
26	0.44	0.47	0.45	181
27	0.69	0.62	0.65	201
28	0.48	0.55	0.51	165
29	0.40	0.42	0.41	155
30	0.50	0.49	0.50	176
31	0.61	0.62	0.62	176
32	0.41	0.37	0.39	168
33	0.47	0.47	0.47	169
34	0.44	0.43	0.44	177
35	0.62	0.64	0.63	173
36	0.53	0.54	0.54	194
37	0.87	0.81	0.84	204
38	0.85	0.83	0.84	192
39	0.74	0.71	0.72	160
40	0.58	0.64	0.61	158
41	0.74	0.75	0.75	187
42	0.72	0.77	0.74	177
43	0.64	0.60	0.62	172
44	0.70	0.75	0.73	162
45	0.73	0.74	0.73	179
46	0.80	0.85	0.83	191
avg / total	0.59	0.58	0.58	8000

Overall accuracy 58.45%

Here also we can see that algorithm is working better on Devanagari digits class I.e for 37-46 class F1 score is increased abruptly

This result proves that HOG is working fine as “Feature Reduction algorithm” in Decision Tree also.

Results of Decision tree makes one thing clear that is “Random Forest” can produce remarkable result.

As Random Forest is ensemble of many “Decision Tree” my next approach is to try “Random Forest Classifier”.

Random Forest :

Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.

This works well because a single decision tree may be prone to a noise, but aggregate of many decision trees reduce the effect of noise giving more accurate results.

I ran Random forest 20,000 training images and tested on 8000 images,with 3000 max_leaf_nodes and 160 n_estimators and got 77.8% accuracy

- Motivated by the result of HOG (Histogram Oriented Gradients) feature extractor to achieve higher accuracy on SVM, I gave it a try reducing the dimensions to 128(16 cells with 8 orientations each) and ran Random Forest with min_samples_split= 3 and criterion='gini' and got the following results: -

	precision	recall	f1-score	support
1	0.96	0.94	0.95	406
2	0.96	0.94	0.95	425
3	0.96	0.91	0.93	436
4	0.84	0.86	0.85	410
5	0.96	0.92	0.94	400
6	0.88	0.90	0.89	418
7	0.91	0.86	0.88	412
8	0.93	0.91	0.92	384
9	0.96	0.94	0.95	388
10	0.95	0.95	0.95	411
11	0.93	0.94	0.93	404
12	0.94	0.97	0.96	429
13	0.91	0.94	0.93	409
14	0.88	0.95	0.91	404
15	0.88	0.96	0.91	382
16	0.93	0.96	0.94	404
17	0.85	0.83	0.84	368
18	0.91	0.81	0.86	381
19	0.90	0.90	0.90	387
20	0.93	0.90	0.92	421
21	0.84	0.92	0.87	393
22	0.91	0.95	0.93	387
23	0.92	0.83	0.87	401
24	0.93	0.89	0.91	397
25	0.90	0.92	0.91	369
26	0.82	0.88	0.85	388
27	0.97	0.95	0.96	378
28	0.94	0.94	0.94	396
29	0.88	0.83	0.85	393
30	0.93	0.88	0.91	423
31	0.82	0.95	0.88	384
32	0.92	0.85	0.88	382
33	0.94	0.88	0.91	381
34	0.87	0.91	0.89	398
35	0.94	0.94	0.94	381
36	0.92	0.94	0.93	376
37	0.99	0.99	0.99	423
38	0.96	0.98	0.97	422
39	0.96	0.98	0.97	388
40	0.98	0.96	0.97	441
41	0.99	0.98	0.98	405
42	0.98	0.99	0.98	386
43	0.95	0.95	0.95	406
44	0.99	0.99	0.99	422
45	0.97	0.98	0.97	422
46	0.97	0.99	0.98	379
avg / total	0.93	0.92	0.92	18400

Accuracy = 92.48%