PRML Assignment 3

V S S Anirudh Sharma (EE18B036), Hema Landa (EE19B036)

Time Series Data

We are given 2 different problems to solve.

1. Isolated spoken digit recognition
2. Telugu Handwritten character recognition

Both these datasets were given as a time series. 2 different methods were used to solve each of the problem:

1. Dynamic Time Warping
2. Hidden Markov Models

We shall now discuss each of these methods and results.

Dynamic Time Warping

5 different methods have been used for classification

Classification 1: Naive

Given M training data per each class, DTW distance was computed with each of these M against every single development data. Owing to such huge complexity, this has to be the most time taking one of all time series methods and techniques discussed in the paper.

Classification 2: Medoid based

Given C classes, for each class, one medoid was computed with DTW distance as the distance metric. Now, instead of computing DTW distances of a development data with $M \times C$ data points, we now simply compute against C data.

Classification 3: Medoid based Mean Shifted

Now instead of straightaway measuring distances, we first shifted our above medoids and all the development data to their corresponding individual means. The benefit of this technique can be appreciated in the Handwriting example, where centering the letters would imply reduction in distances due to writing letters at different coordinates.

Classification 4: Medoid of Mean Shifted data

Here, we started off by centering all our data and only then computed medoids and did everything else we did in classification 2. This method can be better put as a data preparation over classification 2 rather than a novel method in itself.

Classification 5: Normalized Mean Shifted Data

Here we did everything we did in classification 4, adding to which we normalized individual data to ensure no extra distances due to character size differences.

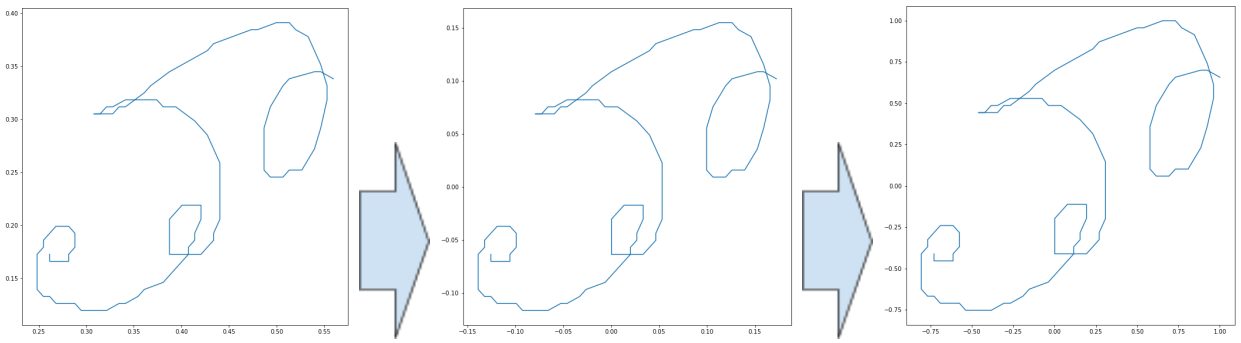

Fig 1: From raw character to mean shifted character to normalized character

Results

Handwriting classification

As expected, **classifier 5** shows the highest accuracy, highest AUC-ROC and least run-time, thus showing the merit of our hypothesis and standing as the best classifier.

Table 1: DTW for Handwriting classification			
Classifier	**Accuracy**	**Pre-processing time**	**Execution time**
Classifier 1	71.0%	0 seconds	1965.00 seconds
Classifier 2	83.0%	705.22 seconds	19.88 seconds
Classifier 3	97.0%	705.22 seconds	19.96 seconds
Classifier 4	98.0%	725.25seconds	16.85 seconds
Classifier 5	98.0%	724.46 seconds	17.67 seconds

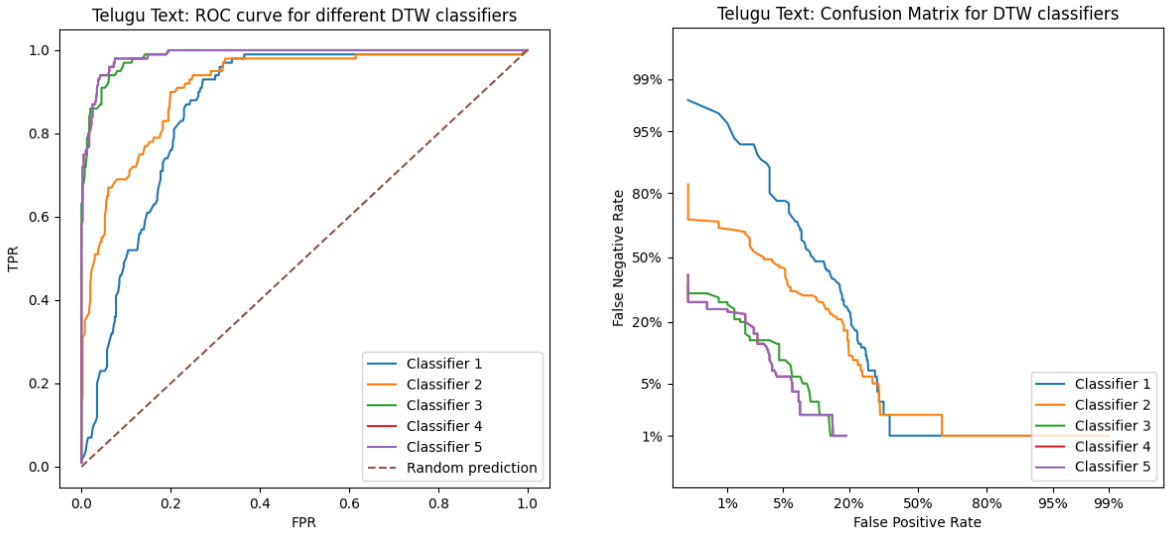

Fig 2: ROC-DET curves for DTW algorithms, Handwriting recognition problem

Spoken Digit

Despite Classifier 1 having the highest accuracy rate, it shows the worst AUC-ROC and run-time.

Table 2: DTW for Spoken Digits			
Classifier	**Accuracy**	**Pre-processing time**	**Execution time**
Classifier 1	96.66666666666667%	0 seconds	835.15 seconds
Classifier 2	95.0%	236.89 seconds	18.30 seconds
Classifier 3	95.0%	236.89 seconds	18.37 seconds
Classifier 4	95.0%	222.49 seconds	18.27 seconds
Classifier 5	95.0%	226.80 seconds	18.27seconds

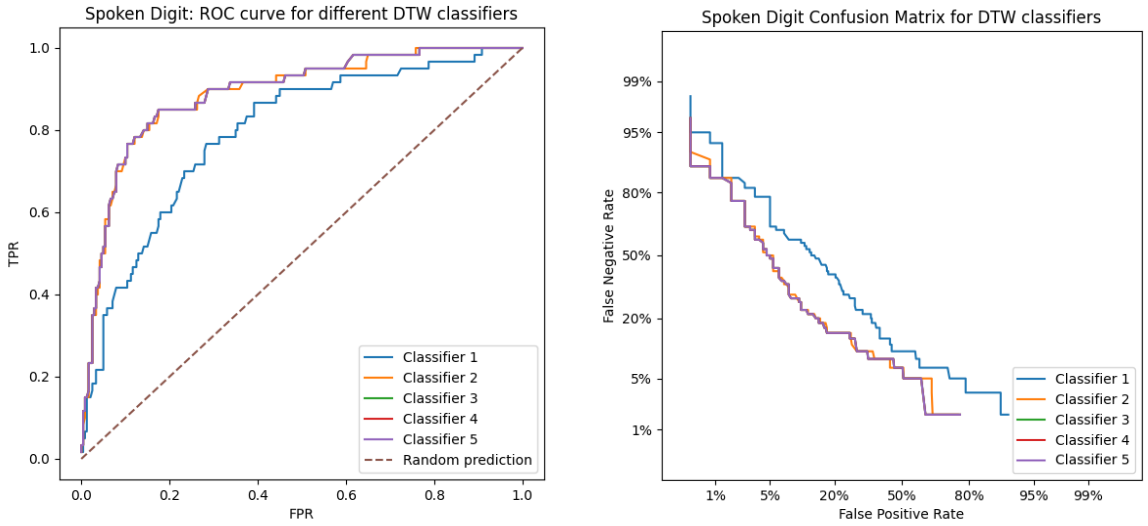

Fig 3: ROC-DET curves for DTW algorithms, spoken digits problem

Hidden Markov Models

Here the setup is different from the one discussed in class since the regular model involves hidden states and for the system being in a state means giving out an observation. But in our given model, an observation is emitted for every transmission (to itself or next state). Moreover, these emission probabilities are different when the transition is to itself and when the transition is to the next state. This means that the standard forward/backward algorithms wouldn't fare.

Thus, we bring out a slight variant of the same.

```
def forward1(O,recP,traP,B,C):
    #recP is recurrence emission probabilities
    #traP is transmission emission probabilities
    N,M = B.shape #  N=#states, M=#symbols
    T = O.shape[0]
    alpha = np.zeros((T+1,N))
    alpha[0, 0] = 1
    for t in range(T):
        for j in range(N):
            #probability of recurrence
            alpha[t+1, j] = alpha[t,j]*recP[j]*B[j][O[t]]
            if j>0:
                #probability of transmission
                alpha[t+1, j] += alpha[t,j-1]*traP[j-1]*C[j-1][O[t]]
    return np.sum(alpha[-1,:])
```

Results

When it comes to classification, the algorithm runs within a couple of seconds. The overall program executes within 5-10 seconds, which is just way faster than DTW.

Spoken Digits

Linguistic and intuitively, we chose the number of states as:

Table 3: HMM State count for Spoken Digits	
Class	**#States**
2	4
3	5
4	6
8	5
9	4

With this, we varied the number of clusters made. **K=35 can be considered the best model owing to high accuracy and AUC-ROC.** But any model from K=15 is giving high AUC-ROC and Accuracy so for simplicity of model and computation, we can do well with K=15, too.

Table 4: HMM for Spoken Digits	
K values	**Accuracy**
5	51.66%
10	93.33%
15	95.0%
20	93.33%
25	95.0%
30	95.0%
35	96.66%

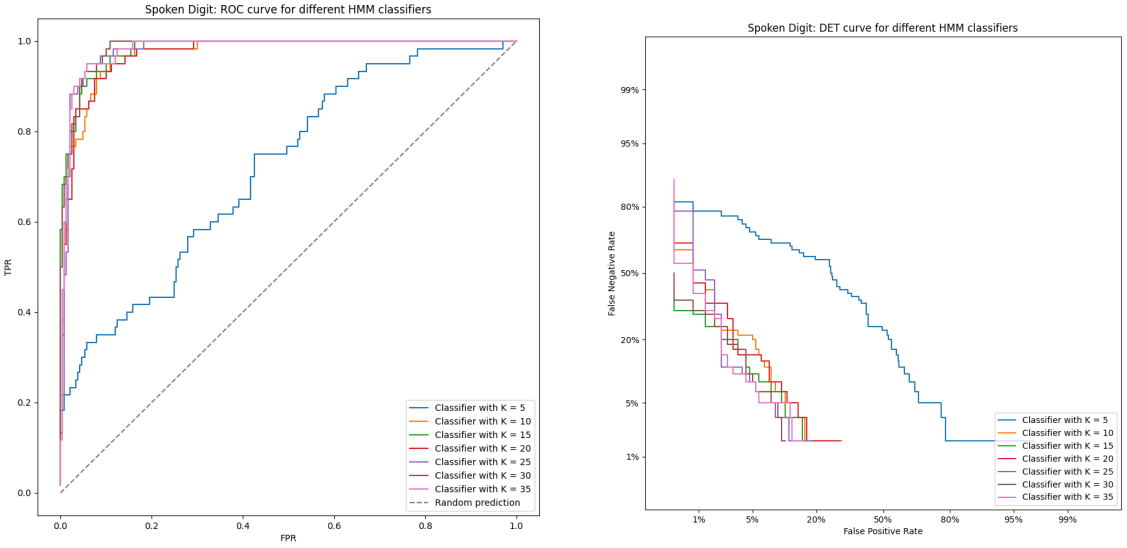

Fig 4: ROC-DET curves for HMM algorithms, spoken digits problem

Handwriting Classification

From an intuitive analysis of types of strokes used, we chose the state count as follows:

Table 5: HMM State count for Handwriting Classification	
Class	**#States**
a	5
ai	7
bA	7
lA	6
tA	7

With this, we varied the number of clusters made. **K=15** stands to be the best model, with highest AUC-ROC and accuracy**.**

Table 6: HMM for Handwriting Classification	
K values	**Accuracy**
5	60.0%
10	91.0%
15	100.0%
20	95.0%
25	96.0%
30	91.0%
35	99%

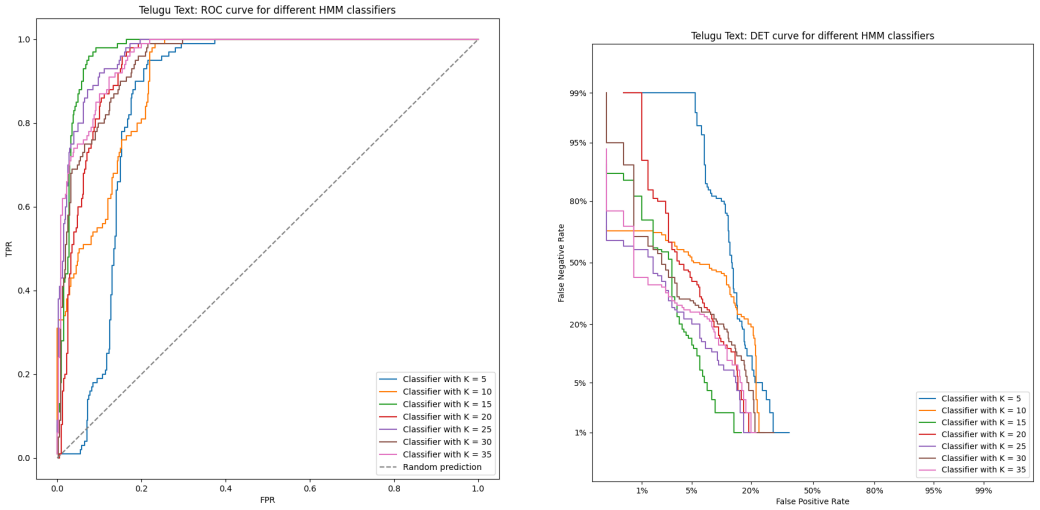

Fig 5: ROC-DET curves for HMM algorithms, handwriting recognition problem

K Means and GMMs for Synthetic Data

- Given synthetic data and images data with 23 features and 5 classes
- Performed K means to get initial variables for GMMs
- Performed GMMs varying number of iterations, number of clusters and with diagonal matrices, full matrices as normal gaussian covariance matrices for classes

Number of clusters(K)	Number of GMM iterations (niter)	Number of K mean iterations(k niter)	Accuracy (Full Normal Matrices)	Accuracy (Diagonal Matrices)
5	2	10	73.9	60.8
5	5	10	76.8	60.2
5	2	2	67.8	58
10	2	10	96.5	90.8
10	5	10	99.4	93.8
10	10	10	96.7	96
12	2	10	99.8	96.4
12	3	10	95.6	94.3
13	2	10	100	96
13	2	2	99.7	95.4
19	2	10	100	100

Observations:

- K means is almost converging within 5-6 iterations
- We are getting maximum accuracy with just 2-3 GMM iterations when initial points are taken from k means, getting minimum with particular n value and upon increasing this we are overshooting and accuracy is decreasing

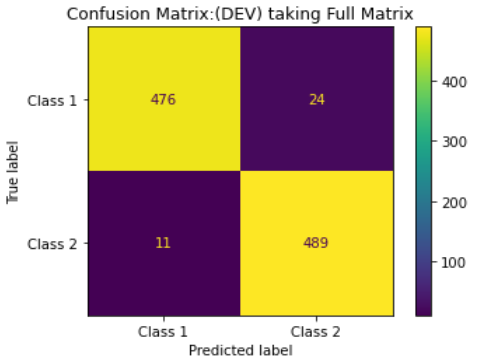

Fig 6: k=10, niter=2, k niter=10

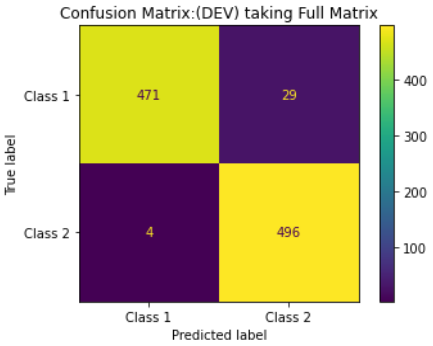

Fig 7: k=10, niter=10, k niter=10

- As number of clusters increases, accuracy increases
- When train data is less, go with diagonal matrices but takes more computation
- When covariance matrices are diagonal, they are taking more clusters which is here 19 with 2 GMMs and with full matrices it can be seen as 13 clusters.

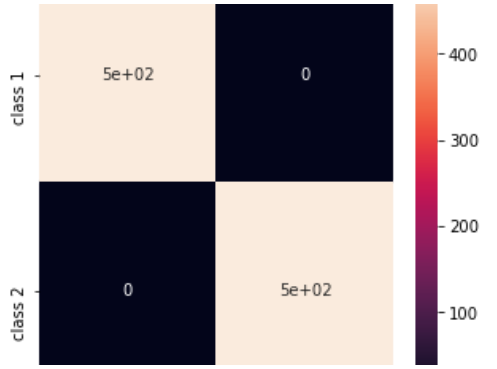

Fig 8: k=13, niter=2, k niter=10

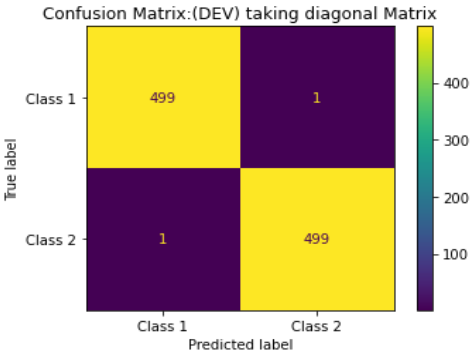

Fig 9: k=18, niter=2, k niter=10

For Synthetic Data

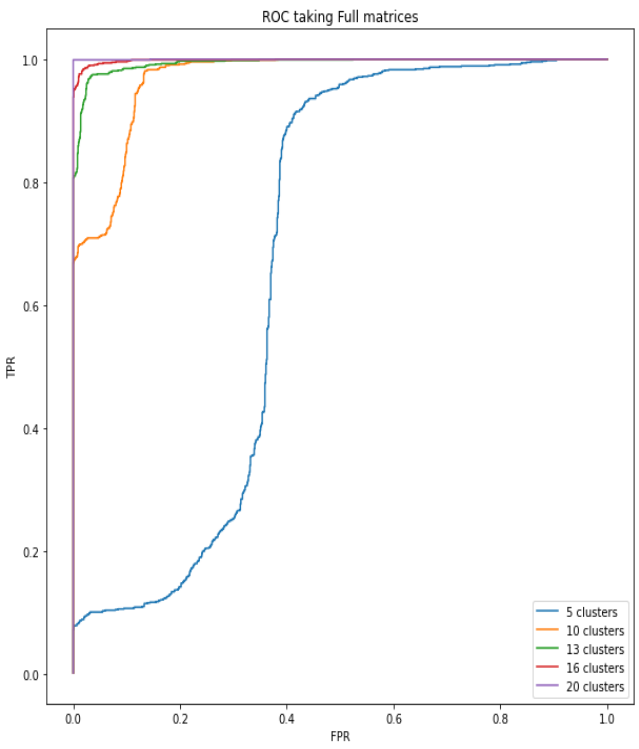

Fig 10: ROC Curves taking Full Matrices

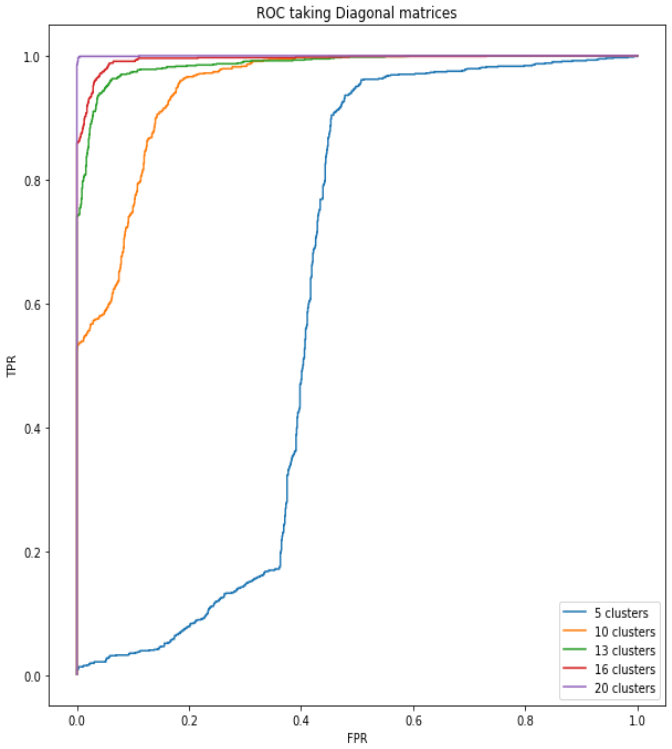

Fig 11: ROC Curves taking Diagonal Matrices

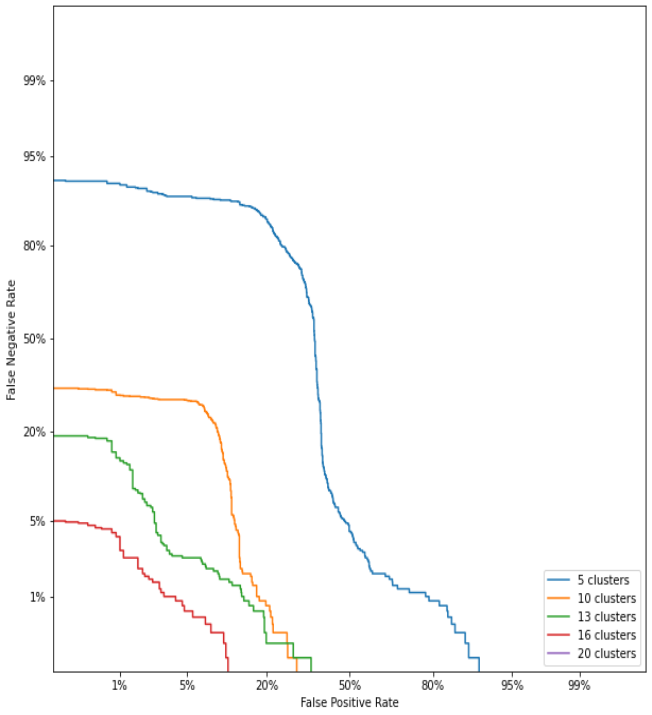

Fig 12: DET Curves taking Full Matrices

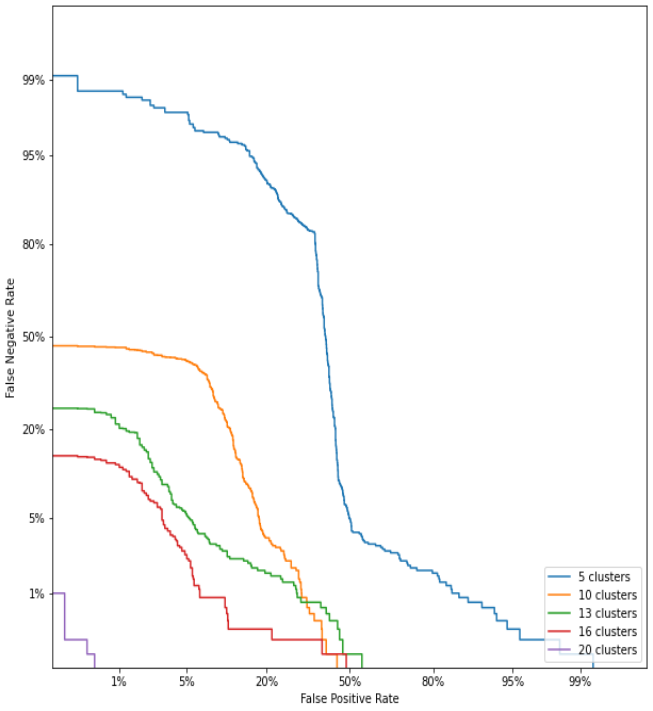

Fig 13: DET Curves taking Diagonal Matrices

- But we got a clear and smooth decision boundary for K=20 even though it classified all development data with 13 clusters

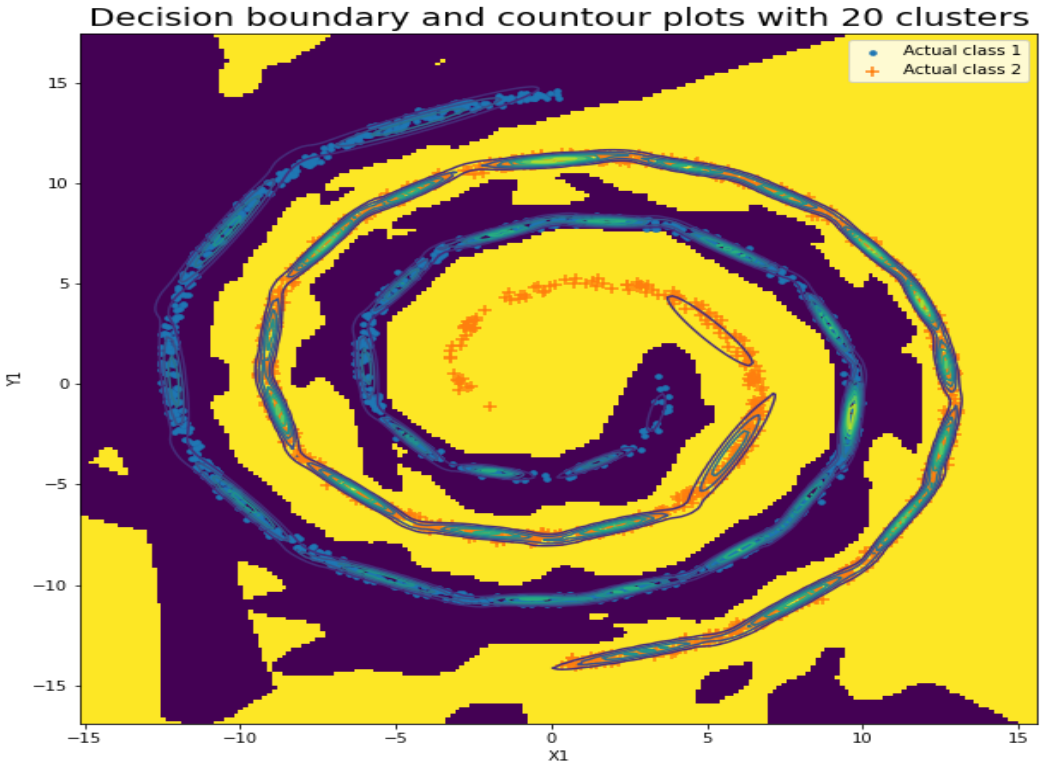

Fig 14: Decision Boundary with 20 clusters

K Means and GMMs for Images Data

- Here every image can be seen as 6*6 block and each block with 23 features
- Since these features are independent of its spacial location we can represent all 36 blocks with same GMM
- We got a maximum accuracy of 55-60 percent
- Almost the results are same for K=2,5,10,20 so each block can be represented with 2 clusters

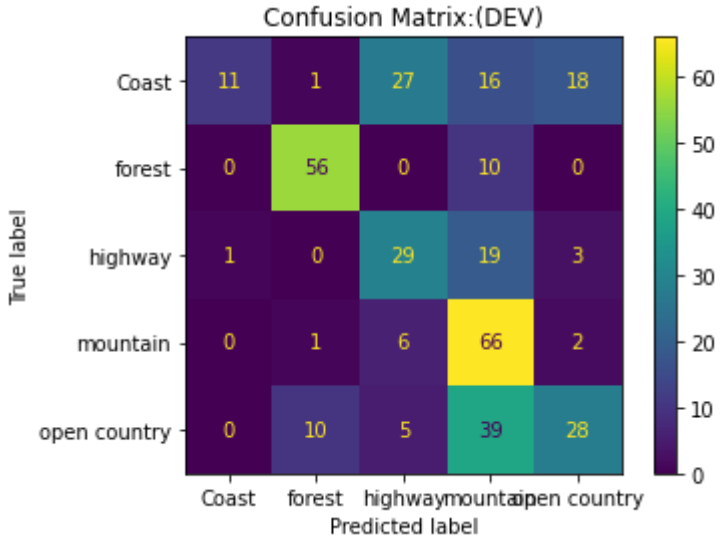

Fig 15: K = 5clusters, Niter = 7 iterations, K Niter = 10

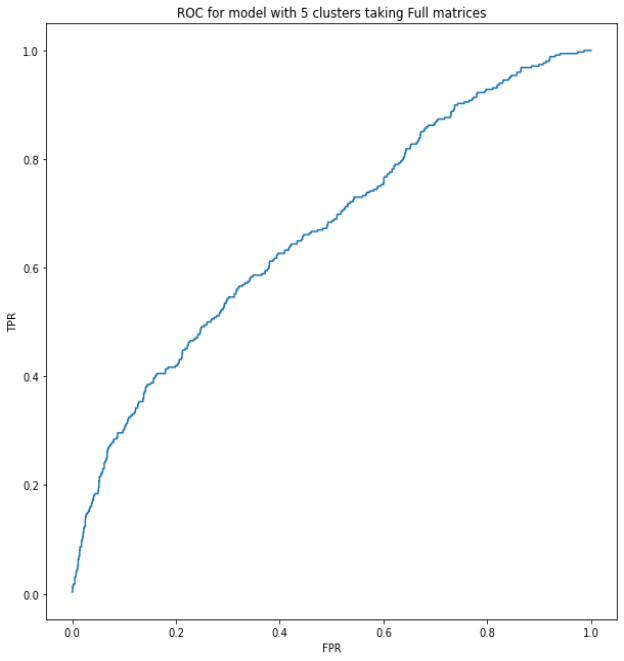

Fig 16: ROC with 5 clusters

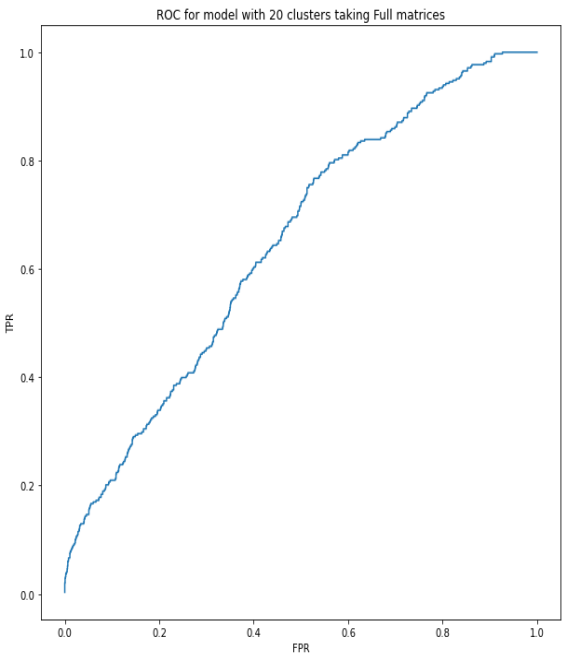

Fig 17: ROC with 20 clusters

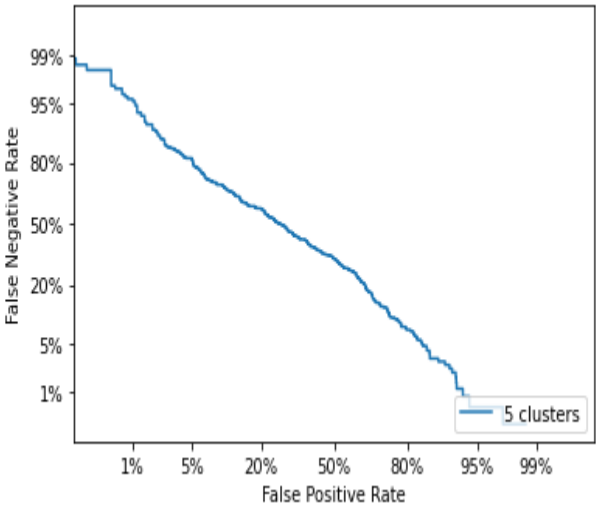

Fig 18: DET with 5 clusters

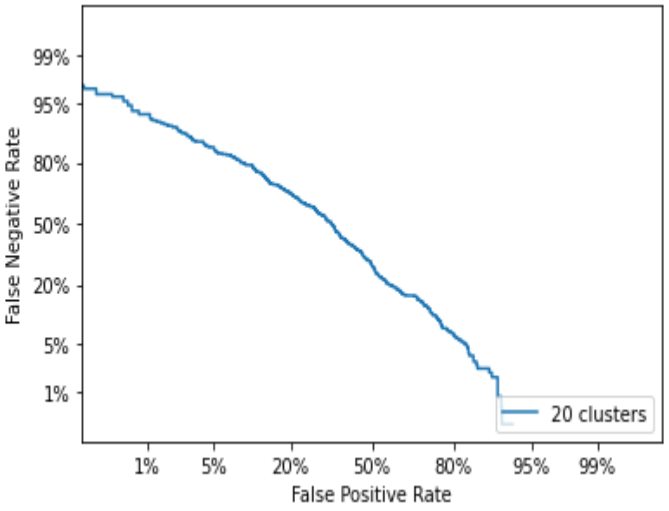

Fig 19: DET with 20 clusters