Mini project

Fundamentals in Statistical Pattern Recognition

Group 1: Batzianoulis Iason, Mirrazavi Salehian Seyed Sina

Reviewed by Group 7: Braun Fabian, Marija Nikolić, Tiago de Freitas Pereira

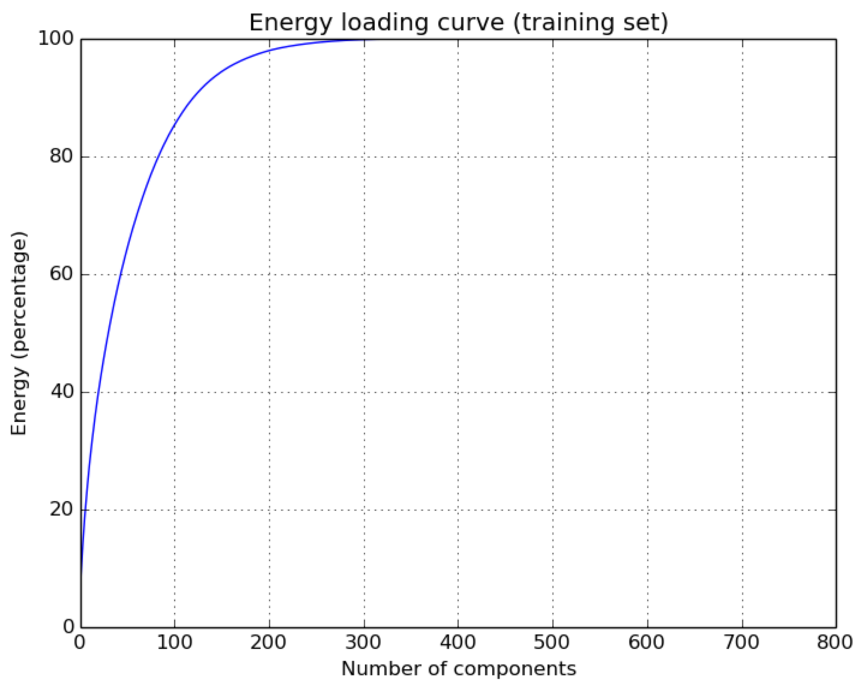
Lausanne, 22.05.2015.

Pen Digit Recognition

- Objective: development of a system for recognizing hand-written digits using SPR techniques
- Data: collected using a track-pad and a stylus
  - 3748 examples for training, 1873 examples for development, 1873 examples for testing
- Methodology:
  - k-Nearest Neighbors & Principal Component Analysis
  - Gaussian mixture model & Principal Component Analysis
- Open source machine learning package: scikit-learn

Principal Component Analysis (PCA)

- PCA for dimensionality reduction
- Selected configuration: PCA=10 (25.91% of the energy)
  - Is sufficient information preserved?
- Projection matrix: 10 x 784



kNN & PCA

- Simple strategy
  - Training phase: storing the feature vectors and class labels of the training samples (capacity=0)
  - Classification phase: a test point is assigned to the class most common amongst its *k* nearest neighbors measured by a distance function
- PCA for dimensionality reduction
- Selected configuration: k=9 and PCA=10 (25.91% of the energy)

kNN & PCA - Results

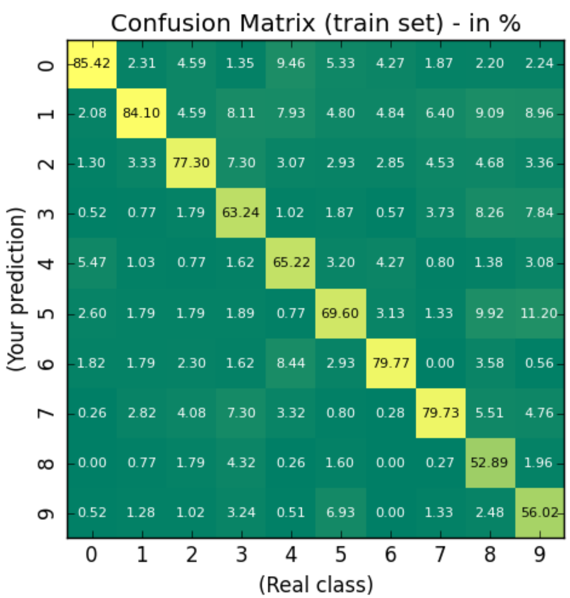
- Good tradeoff between dimensionality and CER with 10 PC

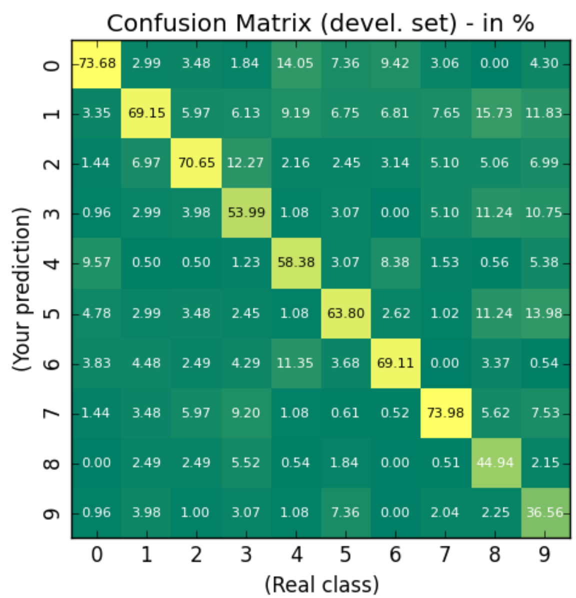
$ python mini_project.py -PCA -c 10 -kNN -nn 9

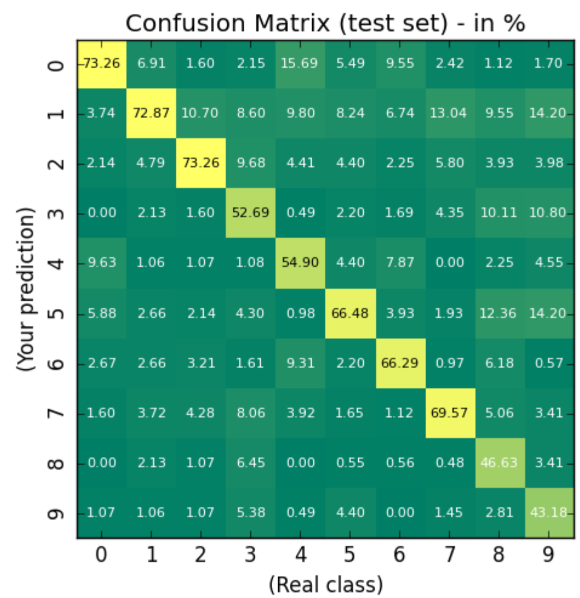
CER_{Train}: 28.50%

CER_{Devel}: 38.07%

CER_{Test}: 37.91%







kNN & PCA - Remarks

- Choice of the number of principal components (PC) to keep
- kNN - parameter *k* selection
  - Sensitivity analysis
- kNN advantages
  - The cost of the learning process is zero
  - No assumptions have to be done
- kNN drawbacks
  - May be computationally expensive to find the *k* nearest neighbors and to calculate the corresponding distances when the dataset is very large
  - The model can not be interpreted

GMM & PCA

- Generative approach to model the digits
- PCA for dimensionality reduction
- Data points and their labels are used for training
- One GMM to model all digits (the means automatically "move" to the digits)
- The whole training set is modeled using 150 gaussian components (capacity=450)
- Classification:
    1. Calculate the probability for a given point and for all labels based on the estimated GMM
    2. Select the class/label corresponding to the highest probabilility

GMM & PCA - Results

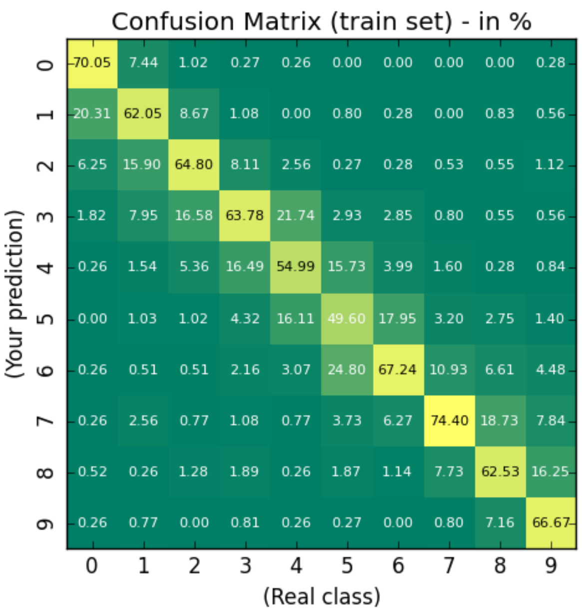
- Clear overfitting (**using the FULL covariance matrix**)

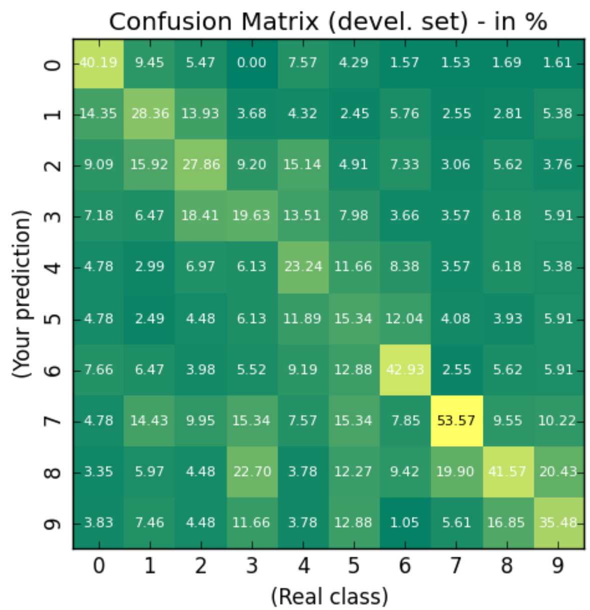
$ python mini_project.py -PCA -c 10 –GMM –nb_gaus 150

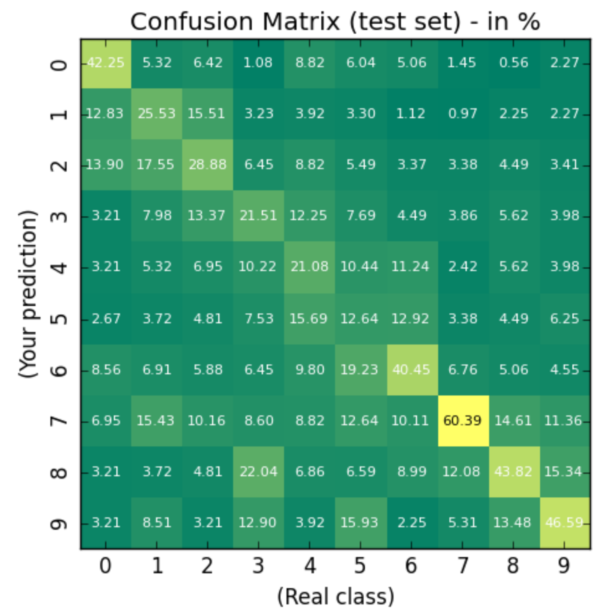
CER_{Train}: 36.45%

CER_{Devel}: 66.68%

CER_{Test}: 65.62%







GMM & PCA - Results

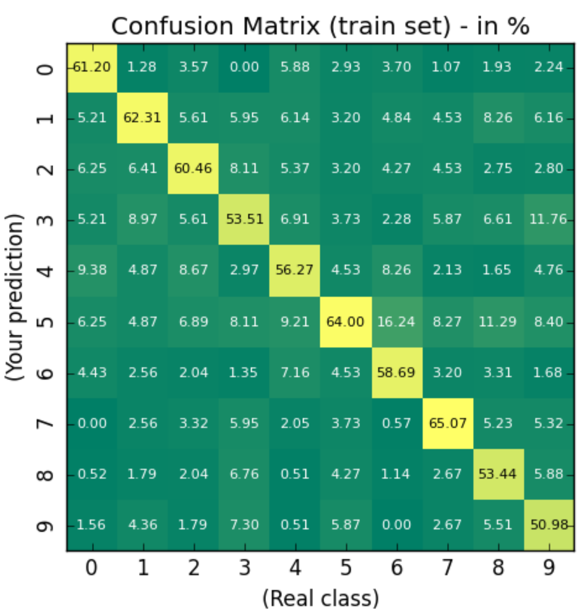
- Using Diagonal Covariance matrix

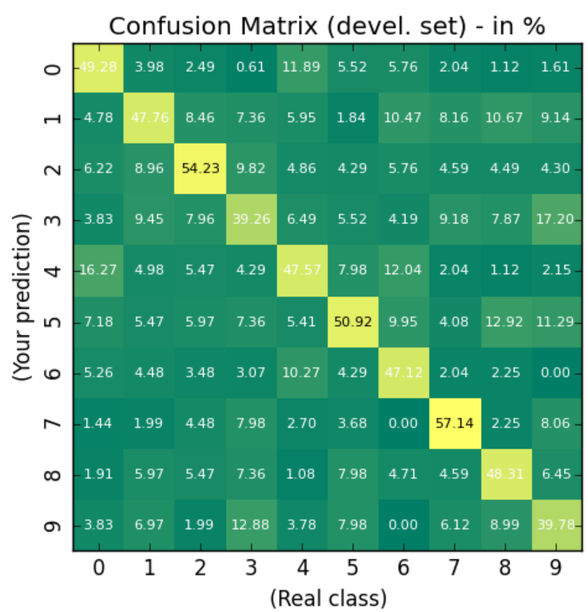
$ python mini_project.py -PCA -c 10 –GMM –nb_gaus 150 --cov_type diag

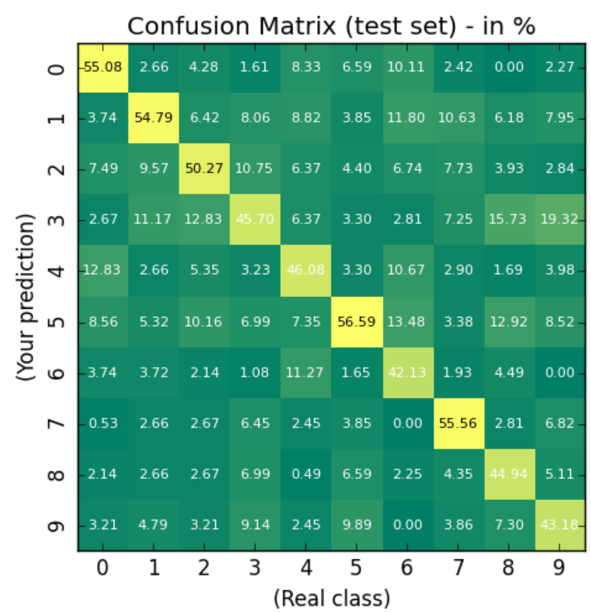
CER_{Train}: 41.33%

CER_{Devel}: 51.68%

CER_{Test}: 50.45%

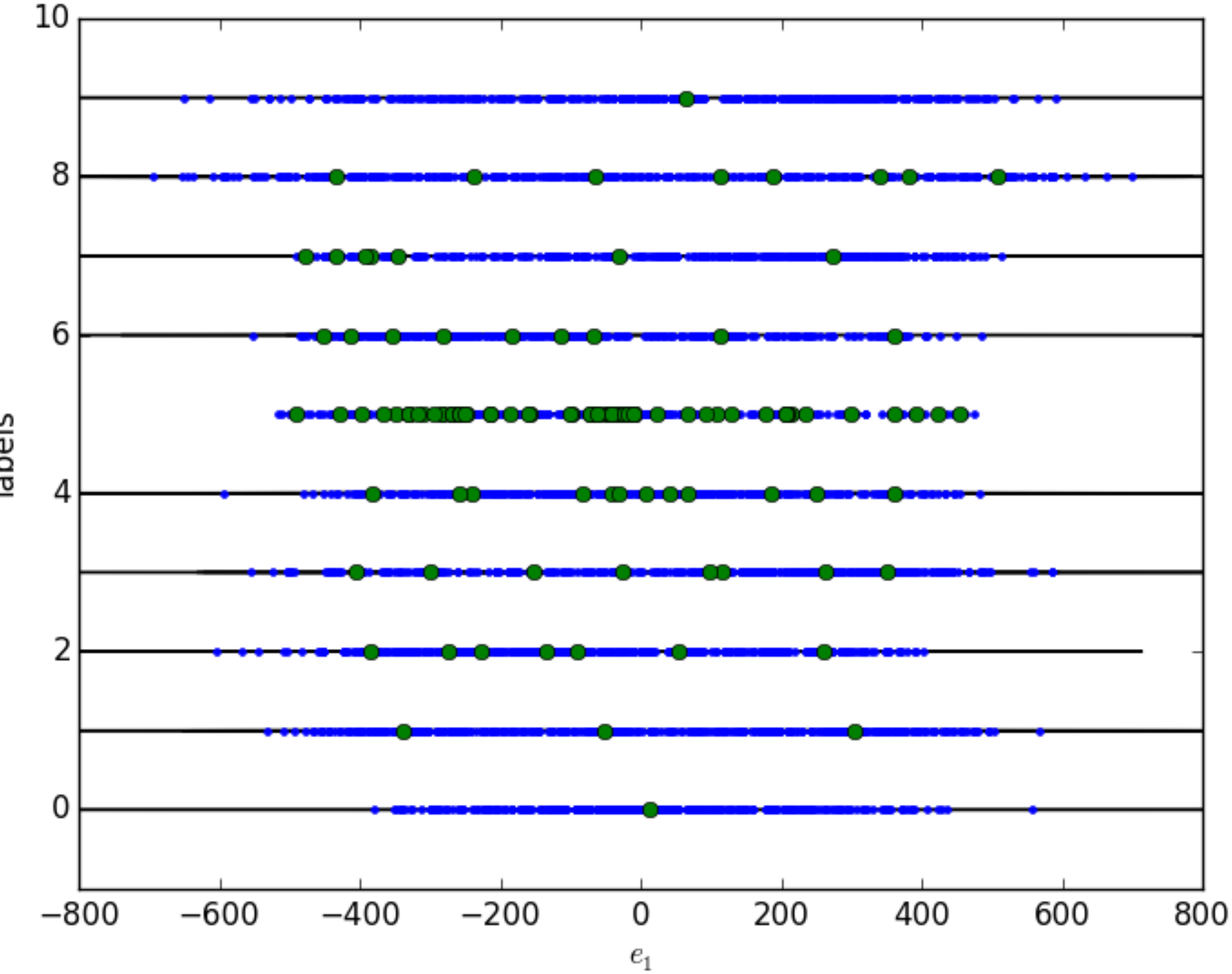






GMM & PCA - Remarks

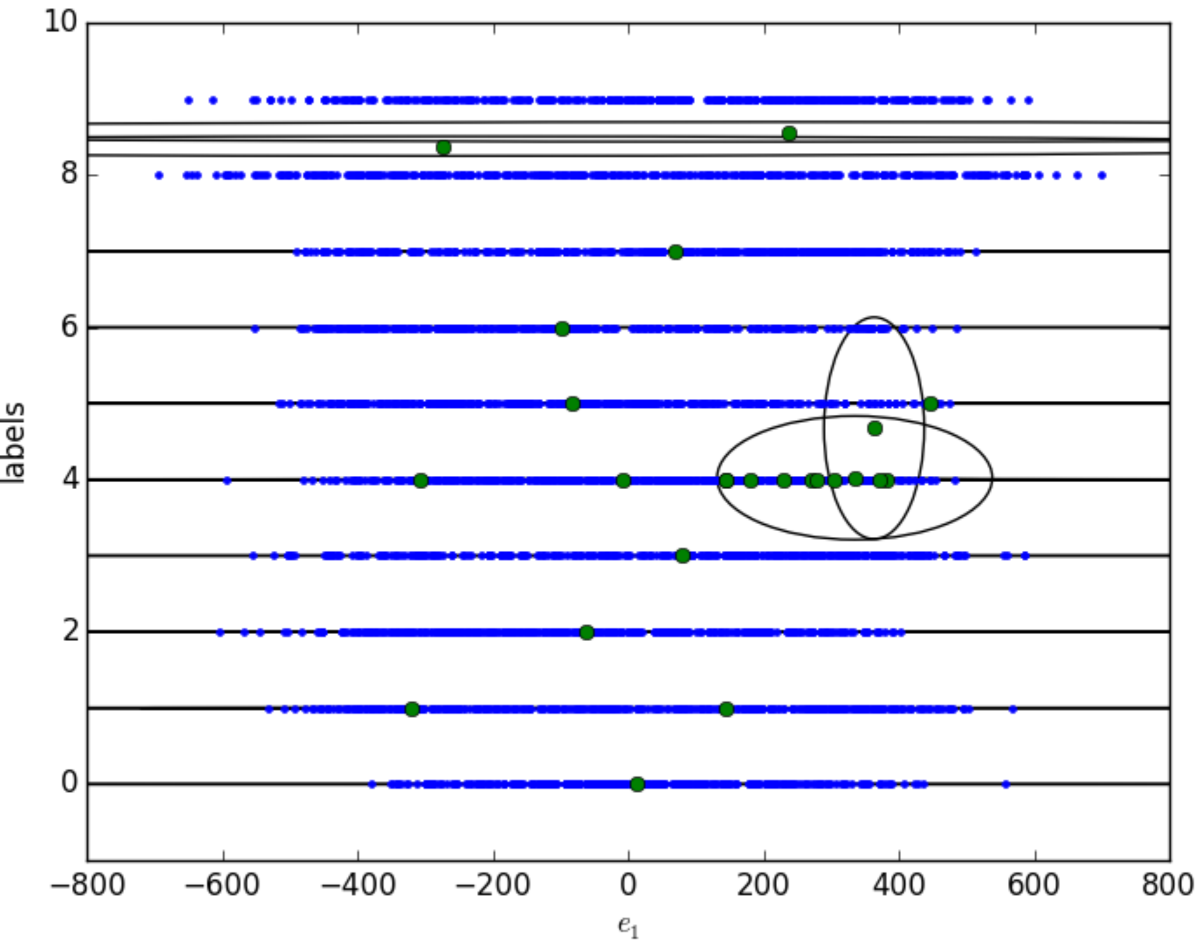
- How the means move along the space? Example with 150 gaussians.



- Possible overfitting on the number 5
- Possible underfitting on the numbers 9,0 and 1

GMM & PCA - Remarks

- How the means move along the space? Example with 24 gaussians.

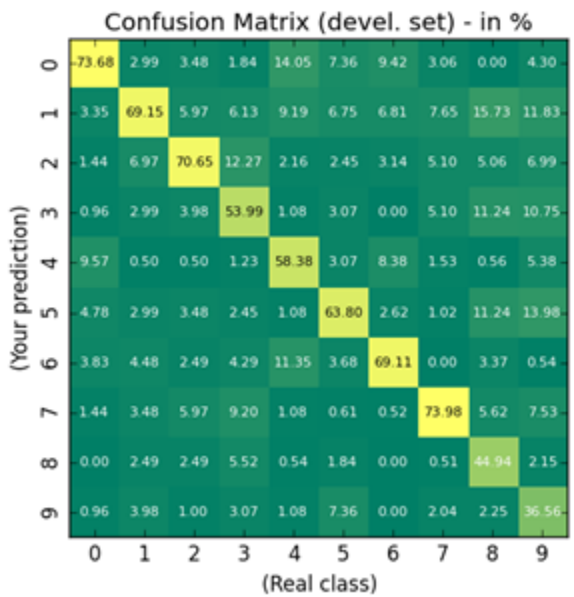


Diagonal covariance

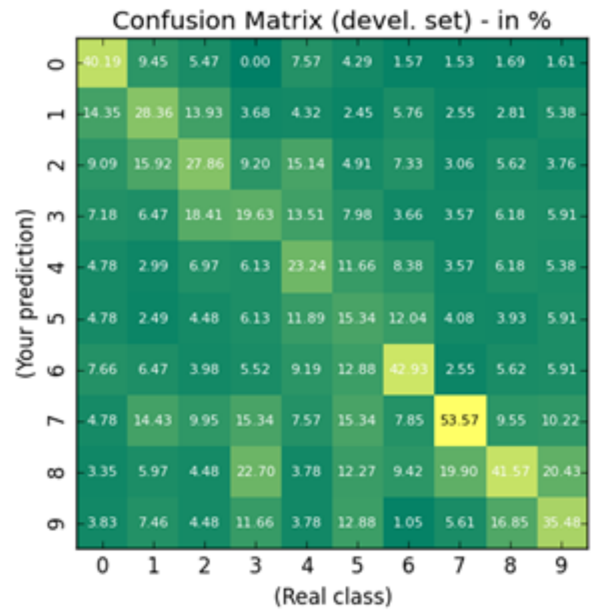
```
[[   7.71279485e+03   1.00000123e-02]
 [   1.99321693e+03   1.00000000e-02]
 [   7.21106645e+03   1.00000000e-02]
 [   1.92641293e+02   1.00000000e-02]
 [   3.29071183e+04   1.00000000e-02]
 [   1.59875255e+04   1.00000000e-02]
 [   6.09943336e+03   1.00000000e-02]
 [   3.12765518e+04   2.56880915e-01]
 [   2.70854969e+03   1.00000000e-02]
 [   3.61342145e+04   2.59261314e-01]
 [   7.52141798e+04   1.00000000e-02]
 [   1.38324272e+04   1.00000000e-02]
 [   4.97549309e+04   1.00000000e-02]
 [   8.02827841e+03   1.00000000e-02]
 [   3.25641184e+04   1.00000000e-02]
 [   1.48142400e+03   2.91374230e+00]
 [   6.41723388e+04   1.00000000e-02]
 [   6.02484636e+03   1.00000001e-02]
 [   1.16635171e+04   1.00000036e-02]
 [   5.53486069e+04   1.00000000e-02]
 [   5.66339402e+03   1.00000000e-02]
 [   4.30077826e+04   1.00000000e-02]
 [   4.06755640e+03   1.62449588e+00]
 [   6.01247404e+04   1.00000000e-02]]
```

kNN & PCA vs. GMM & PCA

- Aggregated level: CER
  - CER_{Devel} (kNN & PCA): 38.07%
  - CER_{Devel} (GMM & PCA): 66.68%
- Disaggregated level: Confusion matrices



kNN & PCA



GMM & PCA

Conclusion

- Simple solution is better (kNN & PCA)
- The results are reproducible
- Suggestion (GMM & PCA): Model one GMM per digit

GMM & PCA - Remarks

- Modelling with less gaussian components (#components=16, capacity=48):

$ python mini_project.py -PCA -c 10 –GMM –nb_gaus 16

CER_{Train}: 62.54%

CER_{Devel}: 67.70%

CER_{Test}: 65.46%

- Correctness of the adopted strategy
- Our suggestion:
  - Split the training data into classes according to their labels
  - Assume a mixture of Gaussian distributions for each class
    - Estimation of the model parameters using only training data without their labels
  - Assign the class labels for test points by comparing the posterior densities of all classes