Machine Learning I
Practice Session III – Classification

1 Goals

The goal of this assignment is to get you to practice the *classification* techniques presented during the class. These are:

- **k-Nearest Neighbor (kNN)**
- **GMM and Bayes**
- **SVM**

During the practical you will learn how to apply kNN, GMM and SVM for classification. Also you will select the optimal hyper-parameters for each one of the algorithms and compare the models' performance in order to decide which set yields the best performance on your data-set.

Furthermore, you will learn how to create a dataset and train the aforementioned algorithms for image classification in videos. This will be performed using an additional software, the MLDetect. You can download MLDetect alongside with short video tutorials here.

1.1 Datasets:

During this practical we will use the following datasets:

- Wine classification
- Grasp clasifcation
- Human activity recognition
- Autonomous driving

You can download the datasets here

2 The User-Interface

2.1 Classification panel

Before starting with the assignment you should familiarize yourself with the user interface. Starting with the classification panel, we will use the functionalities that appear in Fig. 1:

1. **Classification panel**: Enables the classification options
2. **Classify**: Runs the selected classification algorithm with the defined hyper-parameters

3. **Train/Testing ratio**: Specify the ratio between training and testing. For example, if the selected ratio is 25%, MLDemos will use the a random 25% partition of your data for training and the remaining 75%, will be used for testing.

4. **Compare**: Adds the selected algorithm and its hyper-parameters to the comparison tool (Box 7, see Sec.3.3 for more details) where it can be compared with other algorithms on the same training/testing sets.

5. **Classification algorithm**: You can specify the classification algorithm. In this practical we will use k-NN, GMM and SVMs for classification.

6. **Hyper-parameters**: You can specify the algorithm's hyper-parameters.

7. **Comparison Tool**: This opens the comparison environment, where we can run evaluations for multiple algorithms with various hyper-parameters (see Sec. 3.3 for more details).

8. **Information Window**: This displays information about the classification algorithm's performance including the confusion matrices for both train and testing sets (see Sec. 2.3 for more details)

9. **Restrict to visible dims**: If selected, the algorithm is applied only on the dimensions that currently appear in 2D view. Otherwise, the algorithm is applied on all the dimensions that the samples currently have

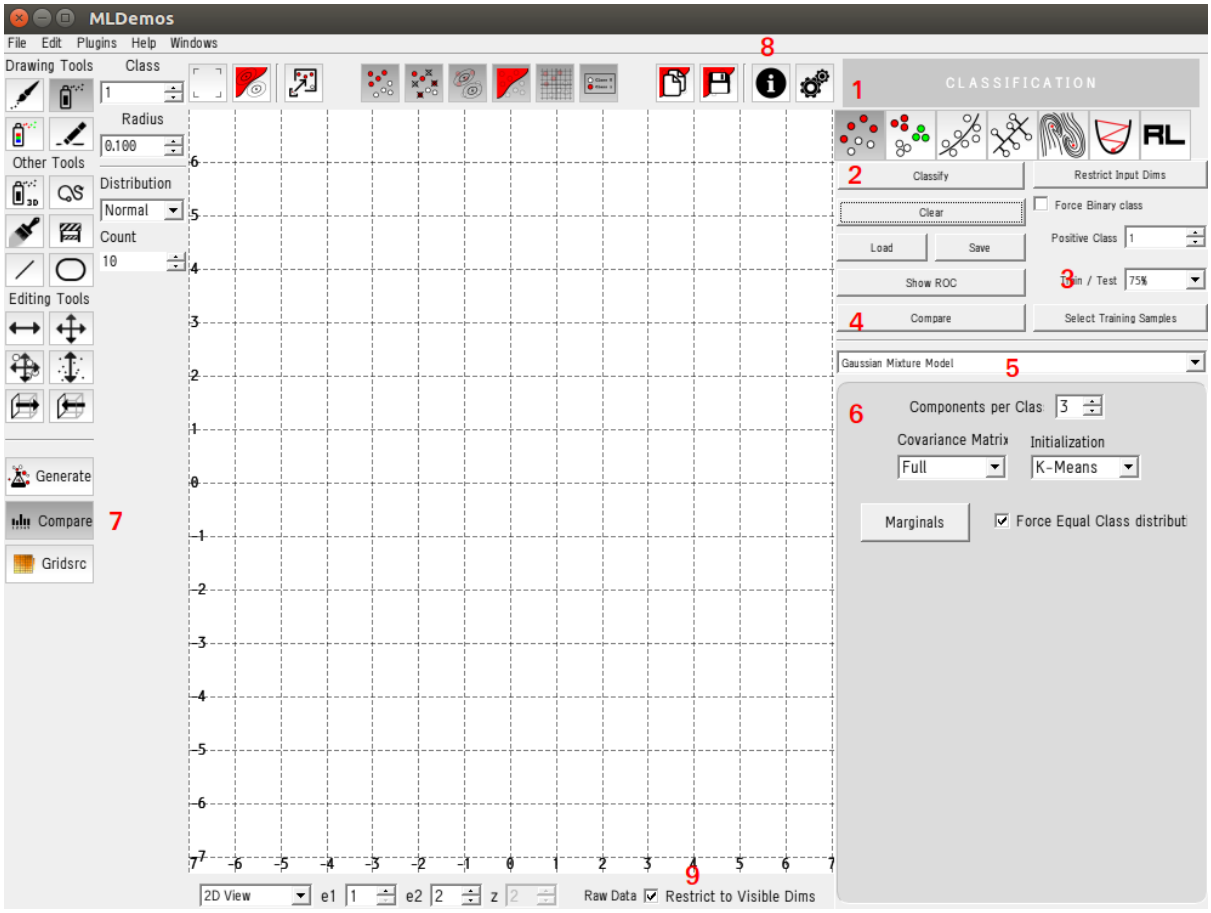

Figure 1: Classification panel.

2.2 The comparison tool

The comparison tool allow as to compare the performance of multiple algorithms on the same training/testing sets and get useful concludes regarding their performance. In order to bring-up the comparison window, click on the compare button (number 7 in Fig.1). The options you have are as follows (see Fig. 2):

1. **Compare** Runs the comparison for the introduced algorithms.
2. **Cross-Validation** You can set the number of validations that will be performed.
3. **Train-Test Ratio** Specifies the percentage of samples that will be used for training and testing.
4. **Algorithms** This window provides the list of compared algorithms alongside with the values of hyper-parameters.
5. **Results** of the evaluation for the specified metric (F-Measure)
6. **Performance metrics** include the F-Measure and Classification Error for both the training and testing sets.
7. **Visualization of performance** You can choose between box plots and histograms.

8. **To Clipboard** copy the evaluation results to clipboard in order to extract them out of MLDemos. The copied results include the mean and variance of both the F-measure and classification error for all the compared algorithms.

In order to add a classification algorithm with specific values of hyper-parameters in the comparison window follow the instructions in Sec. 3.3

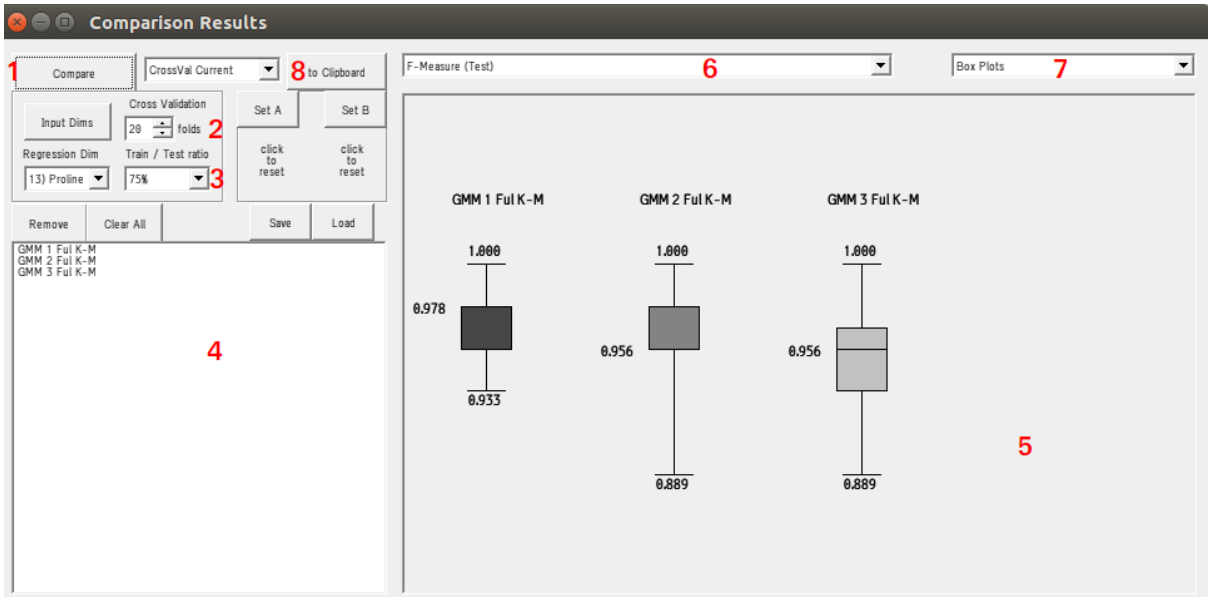

Figure 2: The Comparison window.

2.3 The information panel and Confusion matrices

The information panel provides a color-coded representation of confusion matrices for a classifier on both the training (Matrix 1 in Fig. 3)and testing (Matrix 2 in Fig. 3) sets. The classifier has to be trained (Box.2 in Fig.1) before the confusion matrices appear.

The confusion matrix is a $c \times c$ matrix where c is the number of classes. It illustrates which classes are mixed with which by a classification algorithm. You can see the general structure of a confusion matrix in Table1. For example, in the entry n_{11} is the number of samples which actually belong to class 1 and classified in class 1 by the classifier (correct classification). The entry n_{12} is the number of samples which actually belong to class 1 but classified in class 2 from the classifier, which corresponds to missclassification. Thus, if the classes are easily separable a fine-tuned classifier would produce only correct classifications for all the classes. This would result to a diagonal confusion matrix.

MLDemos provides a color-coded visualization of the confusion matrix Fig.3. The density of the red color represents the amount of samples at each entry which is represented by a tile. The higher the amount of samples in an entry, the highest the color density of the tile. The order of classes in the confusion matrix is the same as they appear in classes' legend.

3 How to:

3.1 Classify data

For this example we will use the wine dataset. First import the dataset, perform PCA and keep the first two eigenvectors.

Table 1: Structure of a confusion matrix

Actual Class \ Predicted Class	C_1	C_2	$\cdots$	C_c
C_1	n_{11}	n_{12}	$n_{1\ldots}$	n_{1c}
C_2	n_{21}	n_{22}	$n_{2\ldots}$	n_{2c}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
C_c	n_{c1}	n_{c2}	$\cdots$	n_{cc}

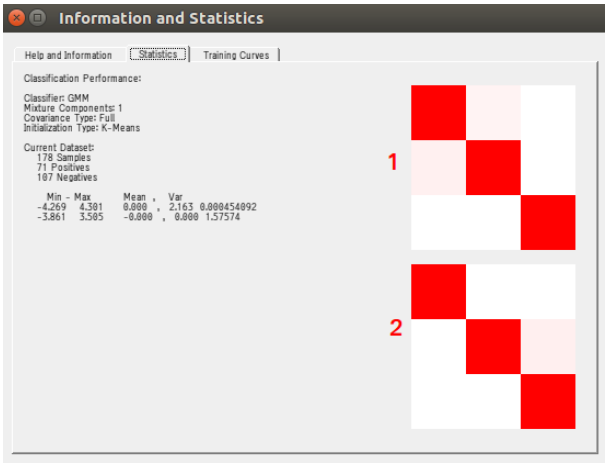

Figure 3: Confusion matrices. The top (1) corresponds at the training set and the bottom (2) at the testing set

- Go at the classification tab (Box 1 in Fig. 1) and select the k-NN algorithm from the drop-down menu (Box 5 in Fig.1)
- Set the value of k at Box 6. of Fig. 1 and click on the classify button (Box 2 in Fig. 1)

MLDemos will create a visualization of the boundaries between the classes as illustrated in Fig. 4. Also miss-classified points of the testing set will be denoted by an x symbol.

Besides kNN, in this practical, we will use the following algorithms with the corresponding hyper-parameters:

- GMM: with hyper-parameters the number of Gaussian components per class and the type of covariance matrices.
- C-SVM: with hyper-parameters the miss-classification penalnty C, the type of kernel (Linear or RBF) and the kernel's parameters (the width for the RBF kernel)

The visualization of SVM decision boundary includes the samples that are support vectors and illustrated with a circle. The circle is colored black for support vectors located on the margin and white for support vectors located within the margin (see Fig.5)

3.2 Get the confusion matrices

Clicking on the information panel button (Box 8 in Fig. 1), you will get the confusion matrices on the training and testing set for the classifier and the hyper-parameters which were set at

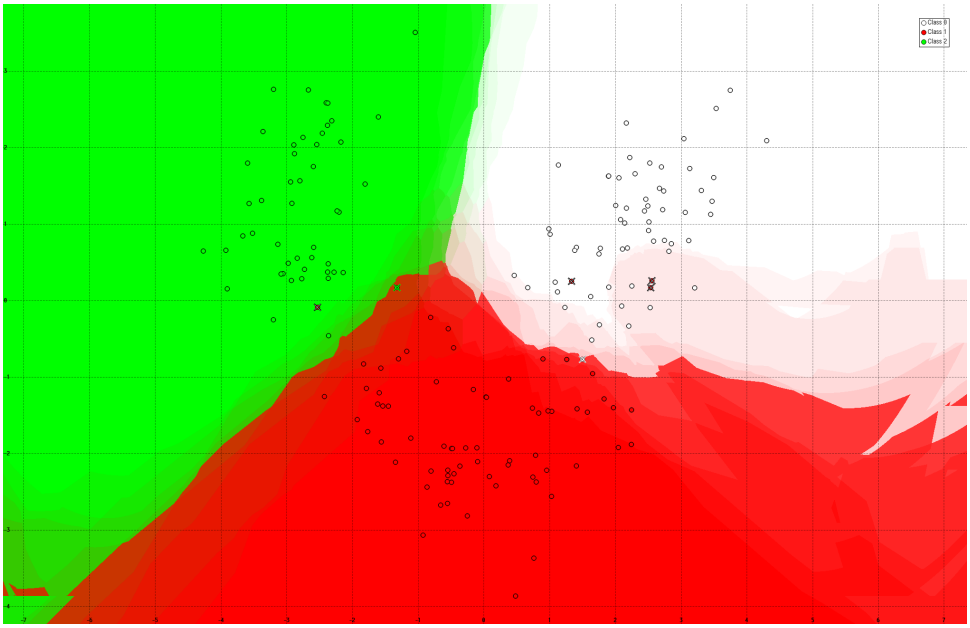

Figure 4: Decision boundaries and miss-classifications of the k-NN classifier at the *Wine* dataset.

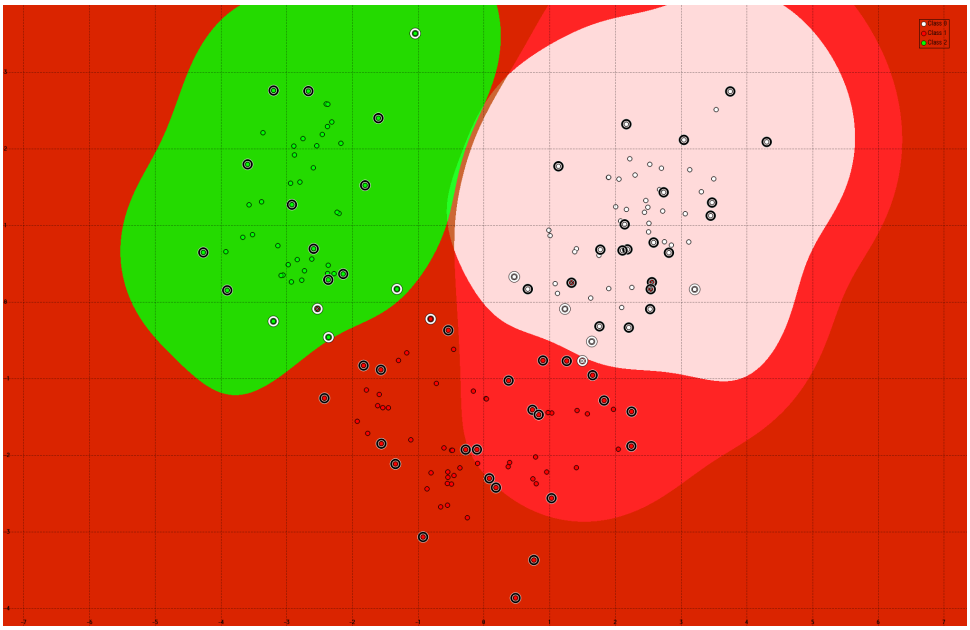

Figure 5: Visualization of decision boundaries and support vectors of the SVM classifier at the *Wine* dataset.

Sec. 3.1. In order to get confusion matrices of different algorithms and/or hyper-parameters, repeat the steps of Section 3.1 but this time choose another classifier and/or value of the hyper-parameters and perform classification.

3.3 Compare performance

In order to compare the performance of multiple classifiers with various hyper-parameters on the same training/testing sets, you need to add them at the comparison tool by clicking the

compare button of the classification menu (Box 4 in Fig. 1). The following steps describe the process which has to be done in order to compare:

- GMMs with one component per class with Full Covariance.
- k-NN with k=6.
- k-NN with k=10 and
- C-SVM with C=500, RBF kernel with width=1

Steps:

1. Select the GMM classifier from the drop-down list (Box. 5 in Fig. 1), set Components per Class to one and Full Covariance Matrix.
2. Click the Compare button, this will add GMMs with the specified parameters at the Comparison tool.
3. Repeat the above steps for the remaining three cases.
4. Open the Comparison tool by clicking the compare button (Box. 7 in Fig. 1). You should be able to see all the four classifiers (Box 4 in Fig. 2)
5. Specify the desired training/testing ratio (Box 3 in Fig. 2) and number of cross-validation folds (Box 2 in Fig. 2)
6. Click the compare button to perform the comparison (Box 1 in Fig. 2). You should be able to see box-plots of the F-measure for each of the cases you compared similarly to Fig. 6)

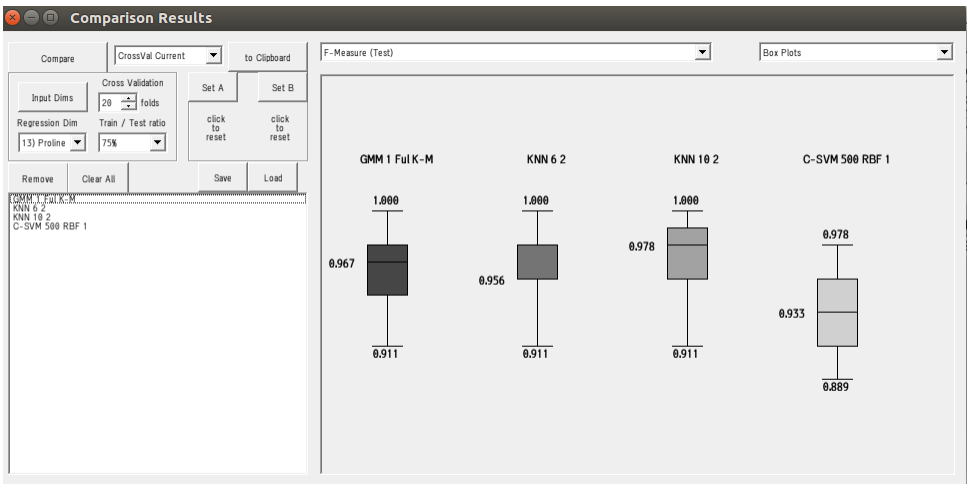

Figure 6: Comparison of classification algorithms on the *Wine* dataset.

4 Recap of important information

Training/testing ratio:
The training/testing ratio decides the amount of data which is used to train and test your classifier respectively. The less training data you use the higher the variance of your estimated

model parameters across folds (i.e. across training runs during crossvalidation). But you will have small variance in the test error. The opposite leads to the same effect, if you use a high percentage of your data for training you may have a low variance associated with the estimated parameters of your classifier. However the variance on your testing error may be high. To recall high variance implies that you cannot trust your estimate. Ideally you would want to trust both the estimate of your classifiers parameters (e.g. coefficients of the covariance matrices for GMM) and its accuracy on the test set. It is all about finding the right trade-off between the two by choosing the appropriate training/testing ratio. Hence, the optimal ratio depends on the complexity of the dataset. The choice of ratio must also take into account the total number of datapoints in your dataset to avoid training or testing on a ridiculously small amount of datapoints.

Cross-validation:
Cross-validation (CV) is a method for evaluating the variance of the test accuracy of a chosen **model** and **training/testing ratio**. To perform CV, you have to choose the number of folds, or repetitions, of the training and testing rounds. This allows you to build statistics on these two values.

Model selection:
The performance you get for particular choices training/testing ratio will also depend on the type of model you chose. By model we refer to the choice of the values of the hyper-parameters for a classifier (number of Gauss functions for GMM, C and kernel width in SVM, k in Nearest Neighbors). In model selection you have to find the right trade-off between the model's complexity and the confidence you have in both the estimated parameters and testing error.

5 Questions

Q1: Load the *Wine* dataset, perform PCA and keep the projections on the first two eigenvectors. Apply classification with k-NN, GMM and C-SVM algorithms using 100% as training testing ratio[1] and vary their hyper-parameters .

Answer the following questions based solely on the illustrations of the decision boundaries (qualitative analysis):

- K-NN hyper-parameter: k and similarity metric
  - How does the value of k affect the shape of the decision boundary?
  - For which value of k do you obtain a good classification model and why?
  - For which value of k do you obtain over-fitted and under-fitted classification models and why?
- GMM hyper-parameters: Number of components and type of covariance matrix
  - How does the type of covariance matrix affect the shape of the decision boundary?
  - How does the number of components affect the shape of the decision boundary?

[1] Using such a training/testing ratio for evaluating the performance of algorithms is not recommended. Nevertheless, in this question we want to focus on how the hyper-parameters affect the boundaries. Therefore, we set 100% to eliminate the effect of different training data on the decision boundary

    - For which values of hyper-parameters do you obtain a good classification model ?
    - For which values of hyper-parameters do you obtain an over-fitted and under-fitted classification model and why?
  - C-SVM hyper-parameters:Miss-classification penalty C, type of kernel (linear or RBF) and kernel hyper-parameters.
    - Use a linear kernel and vary the values of miss-classification penalty. How the shape of the decision boundary is affected by C?
    - Use a RBF kernel and vary its width. How the kernel width affects the number of support vectors and the decision boundary?
    - For which values of hyper-parameters you obtain a well-fitted decision boundary between the classes?
    - For which values of hyper-parameters you obtain an over-fitted and under-fitted classification model and why?

Q2: Load the *Grasp classification* dataset, perform PCA and keep the projections on the first six eigenvectors.

- What appears to be a reasonable selection of training/testing ratio and number of cross-validations for classifying the dataset with k-NN? Support your answer based on the amount of samples and the strengths/weaknesses of the k-NN classifier.
- Using the training/testing ratio and number of cross-validations you choose previously, determine quantitatively (using the comparison tool) the value of k for which you obtain a well-fitted model.
- Provide for which values of k you obtain over-fitted and under-fitted models. In order to support your answer, create a line plot (using Excel) of both the training and testing F-Measure w.r.t the value of k.

Q3: Load the *Activity recognition* dataset, perform PCA and keep the projections on the first five eigenvectors.

- What appears to be a reasonable selection of training/testing ratio and number of cross-validations for classifying the dataset with GMM? Support your answer based on the amount of samples and the strengths/weaknesses of the GMM classifier.
- Using the training/testing ratio and number of cross-validations you choose previously. Evaluate different values of the hyper-parameters (number of components and type of covariance matrices) and determine quantitatively (using the comparison tool) the value of hyper-parameters for which you obtain a well-fitted model.
- Provide a set of hyper-parameters for which you obtain an over-fitted model and a set for which you obtain an under-fitted model. Support your answer based on the mean and variance of the F-measure at the testing set for the specific models.

Q4: Load the *Autonomous navigation* dataset, perform PCA and keep the projections on the first 10 eigenvectors.

- What appears to be a reasonable selection of training/testing ratio and number of cross-validations for classifying the dataset with C-SVM? Support your answer based on the amount of samples and the strengths/weaknesses of the C-SVM classifier.

- Using the training/testing ratio and number of cross-validations you choose previously. Evaluate different values of the hyper-parameters (C and width of the RBF kernel) and determine quantitatively (using the comparison tool) the value of hyper-parameters for which you obtain a well-fitted model.
- Provide a set of hyper-parameters for which you obtain an over-fitted model and a set for which you obtain an under-fitted model. Support your answer based on illustrations of the confusion matrices both on training and testing set.

Q5: For this question you will need to use MLDetect. You can download MLDetect allongside with some video tutorial from here. Once you download MLDetect, extract it and execute the *MLDetect.exe* file located in the MLDetect folder. Watch the video tutorials before proceeding to the question's objectives. The goal is to train ML algorithms for object detection in videos.

- Load the *Ballons1.mp4* video located in the *example video* folder and create a data-set with balloons as demonstrated at the video tutorials. Don't forget to add enough negative samples. Train GMM, SVM and kNN models with the created dataset and use the cross-validation procedure to find acceptable values for the hyper-parameters.
- Once the models are trained, test their ability to detect balloons on the video you used for training. Report on any objects that may be miss-classified and cases where balloons failed to be recognized. Explain why this happens. Which method seems to perform better and why?
- At the next step, load the *Ballons4.mp4* video and test the trained models. Do not retrain the algorithms with images from the new video. Which algorithm performs better at this case? Give any examples of miss-classification and/or failures to detect balloons. Comment on why this happens.
- Load a video and create a data-set of your own choice. Train the algorithms and evaluate their performance. They perform better or worse compared to the balloons videos? Provide an explanation for their good or bad performance considering the differences between the video you used and the balloons video.