classifyData

Mathieu Lagrange

November 18, 2015

Introduction

This report documents the third demonstration of the use of the expLanes framework to conduct a computational experiment. The project classifyData is about the comparison of two classifiers over synthetic data.

Design

The project is divided into four processing steps:

1. **generate**: generation of synthetic datasets
2. **train**: learning models
3. **probe**: probing models
4. **test**: predicting labels

***Generate step**: generation of the synthetic datasets*

The training and testing datasets are both of 3 classes each modeled as mixture of 3 Gaussians.

Figure 1: The training dataset.

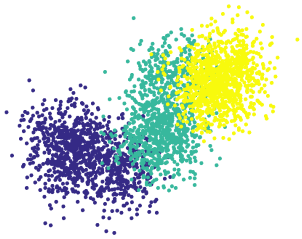

Figure 2: The test dataset.

***Train step**: Learning models*

2 types of classifiers are considered. The nearest neighbors classifier is the simplest one. In this case, the classifier is entirely defined by the training data. Thus, no explicit modeling is performed. On contrary, the Gaussian mixture based approach needs to estimate a model for each class. In order to manipulate those models, the netlab[1] toolbox is used. This process is iterative and it can be interesting to see the impact of the number of iterations on the likelihood and also on the resulting classification accuracy.

Indeed, if increase and convergence of likelihood are mathematically ensured, the relation between the number of iterations and the accuracy is less straightforward.

By default, explanes process the settings in unspecified order, especially in parallel mode. In order ot ensure the sequentiality of several settings related to a factor, one can set the said factor to be sequential:

```
nbIterations =s2:=1/2= 50:50:200
```

At the first setting of the sequential run, explanes initializes `config.sequentialData` to an empty value. This variable can be used to store sequential data, in our case the GMM models.

The code for learning the GMMs thus reads:

[1] `http://www.mathworks.com/matlabcentral/fileexchange/2654-netlab`

```
if isempty(config.sequentialData)
    % first step of the sequential run
    mix = gmm(setting.nbDimensions, setting.nbGaussians, 'diag');
    options(14) = 100;
    % initialize gmm model using kmeans
    mix = gmminit(mix, trainingData, options);
    options(14) = setting.nbIterations;
else
    % continuing step of the sequential run
    options(14) = setting.nbIterations-config.sequentialData.nbIterations;
    % get model from the sequential data of the previous run
    mix = config.sequentialData.gmm;
end
% EM training of the model
[model options] = gmmem(mix, trainingData, options);
% record training likelihood
obs.trainLikelihood = options(8);
obs.trainLikelihood = mean(gmmprob(mix, trainingData));
% store model for the next step
store.model = model;
% save model and number of iterations already done for the next
% step of the sequential run
config.sequentialData.nbIterations = setting.nbIterations;
```

Probe step: *Probing models*

Once GMMs models are learned, one needs to compute their likelihood on some data, which can be done using the `gmmprobe` function:

```
% conpute and store likelihood
store.likelihood = gmmprob(data.model, samples).';
```

Test step: *Predicting labels*

Predicting labels using the nearest neighbors approach is done using the `knn` and `knnfwd` functions:

```
load = expLoad(config, [], 1);
% get testing samples
```

With the GMMs approach, one needs to collect the likelihoods of the GMMs each one modeling a given class and consider the GMM with the highest likelihood as the prediction:

```
% get the likelihhod of the different models
for k=1:nbClasses
    likelihood(k, :) = data(k).likelihood;
end
% select the model with the highest likelihood
[mh prediction] = max(likelihood);
class = data(1).class.';
```

This explanes experiment is contracting, that is the number of settings for the third step is higher than the one for the fourth step, because the factor `class` is no longer needed at this last step. Thus the `data` structure is in this case an array with length equal to the

number of classes which can be obtained by the following command:

```
% get number of classes
nbClasses = length(expFactorValues(config, 'class'));
```

Definition of factors

Those factors and their corresponding modalities are defined in the file named `cldaFactors.txt` whose content is the following:

```
method =2:== {'knn', 'gmm'}
class =2:3=1/2= 1:3
set =1,3:== {'train', 'test'}
nbIterations =s2:=1/2= 50:50:200
nbGaussians =2:=1/2= 1:5
nbNeighbors =4:=1/1= 1:2:9
spread === 25
nbElementsPerClass === 1000
nbDimensions === 2
nbGaussiansInData === 3
```

Most the factor design discussed above is compactly displayed in Figure 3.

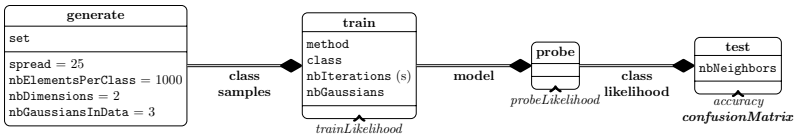

Figure 3: Factor and data flow graph.

Results

The analysis the performance of a classifier can be done using the confusion matrix. For this specific display, the observation is no longer an array but a structure that is filled as follow:

```
% record information for the display of the confusion matrix
obs.confusionMatrix.prediction = prediction;
obs.confusionMatrix.class = class;
obs.confusionMatrix.classNames = expFactorValues(config, 'class');
obs.confusionMatrix.setting = setting;
```

On display, one needs to specify a given setting:

The resulting display shown on Figure 4 is in line with the discrepancy of the training and testing dataset where the latter exhibit a stronger intrication between class 2 and 3, see Figures 1 and 2.

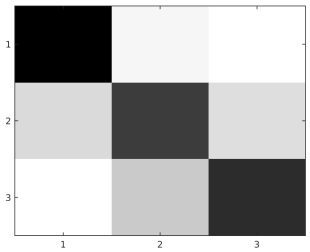

Figure 4: Confusion matrix for the method: gmm, set: test, nbIterations: 200, nbGaussians: 5.

The KNN classifier

The main parameter of this classifier is the number of neighbors. As can be seen on Figure 5, the performance of the classifier on the testing dataset reaches a plateau around 5. As expected, the accuracy decreases with respect to the number of neighbors on the training dataset.

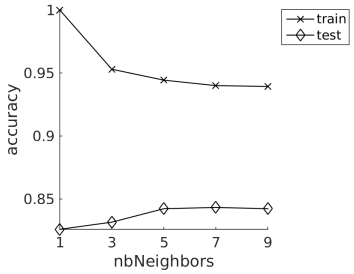

Figure 5: Performance of the KNN classifier on the training and testing dataset.

The GMM classifier

As can be seen on Figure 6, the quality of fit of each GMM increases with respect to the number of observations on the training dataset. On contrary, eventhough the difference is in this toy example negligible, the accuracy decreases with respect to the number of iterations, see Figure 7. This is a common case with iterative methods suggesting a reduced number of iterations to avoid an overfit.

`https://en.wikipedia.org/wiki/Early_stopping`

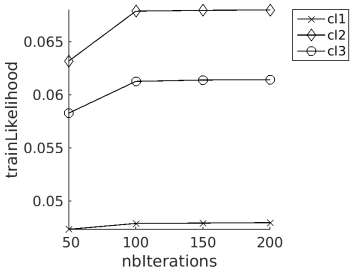

Figure 6: Likelihood for the 3 GMMs models.

Overall

On this toy example, the GMM and KKN classifiers achieve equivalent performance, see Figure 8.

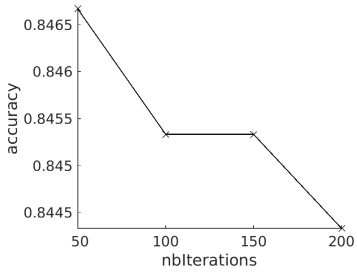

Figure 7: Classification accuracy of the GMM clasifier.

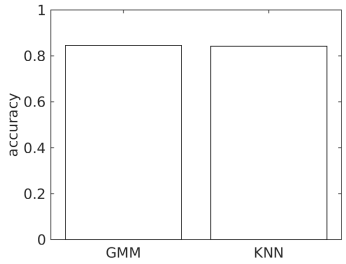

Figure 8: Overall results.