INF582 - DATA SCIENCE AND MINING

ÉCOLE POLYTECHNIQUE

Lab 5: Machine Learning applied to Handwritten digits automatic classification

Jean-Baptiste Bordes, Balázs Kégl, Fragkiskos Malliaros and Michalis Vazirgiannis

January 23, 2015

1 Introduction

The goal of this lab is to train a classifier for automatic handwritten digit recognition using *Gaussian Mixture Models (GMM)*. A mixture distribution can contain a variable number of Gaussians, making it possible to tune the complexity of the model. We will study the impact of this complexity on the performance of the classifier.

1.1 Description of the MNIST digit dataset

The MNIST dataset of handwritten digits [1] contains 60000 samples of training data and 10000 samples of test data. It is a subset of a larger ”real world” database of NIST (National Institute of Standards and Technology) which has been centered and size-normalised on a 28×28 pixels image. The database is provided on moodle through 4 files:

- `train-images-idx3-ubyte` contains the dataset of training images
- `train-labels-idx1-ubyte` contains the corresponding labels of the training set
- `t10k-images-idx3-ubyte` contains the dataset of test images
- `t10k-labels-idx1-ubyte` contains the corresponding labels of the test set

The `main.py` Python script provides the code for reading these files and for storing it in numpy arrays.

To extract relevant features for automatic recognition on this dataset, a SIFT vector is computed at the center of each image [2]. SIFT is a state-of-the-art feature which proved to be particularly efficient for all kind of object recognition. It is a vector of size 128 which describe local gradient properties around a given point. SIFT vectors have already been computed on each image of the dataset, we then have applied PCA on the obtained vectors and saved it in the files ”features”. The code to read this file and store it in numpy arrays is provided as well in `main.py`. The constant `size_features` which is defined at the beginning of `main.py`, is the number of components that we keep for each feature vector; we set it at 8 here in order to have fast computation.

1.2 Gaussian mixture model (GMM)

A probabilistic model is introduced **for every digit** to describe the feature distribution of the data as a Gaussian mixture. Formally, three random variables X, Y, Z are introduced. X stands for the observation, ie the feature vector. Y stands for the corresponding label (the digit). Z stands for an additional hidden variable which takes values into $\{1, \ldots, K\}$ where K is the complexity of the model. The role of this additional variable Z is to take into account for different types of writing of the same digit which could not be captured by only one model.

A Gaussian distribution is then defined for every couple value of (Y, Z):

$$P(X = x|Y = i, Z = j) = \frac{\exp(-(x-\mu_{ij})^t \Sigma_{ij}^{-1}(x-\mu_{ij}))}{\sqrt{(2\pi)^n |\Sigma_{ij}|}} = g_{ij}(x)$$

.
In this lab, the covariance matrix will be assumed to be diagonal for easier computation.

The likelihood of a handwritten digit to be annotated by a particular label is thus obtained by marginalisation on Z:

$$P(X = x|Y = i) = \sum_{j=1}^{K} P(Z = j|Y = i) P(X = x|Y = i, Z = j)$$

.
Let us denote $P(Z = j|Y = i) = \alpha_{ij}$, which can be interpreted as the relative importances of the Gaussians in the mixture. The previous sum can be rewritten as a mixture of Gaussian distributions:

$$P(X = x|Y = i) = \sum_{j=1}^{K} \alpha_{ij} g_{ij}(x)$$

For one given label i, a mixture model of K Gaussians has as parameters:

- K covariance matrices Σ_{ij} of size $n \times n$. In this lab, only the diagonal of the matrices are kept and K vectors of size n are retained.
- K mean vectors μ_{ij} of size n.
- K weight coefficients α_{ij}.

In the file `gmm.py`, the function `loggmm()` provides the log likelihood (the opposite of the logarithm of the probability) of a Gaussian mixture model for a particular feature vector and given parameters.

2 Creation of digit recognition module

We will now implement in Python a digit recognition module using GMMs. Two steps have to be implemented: the training step and the classification step.

2.1 Training step

Considering several Gaussians instead of one to describe the distribution of the features for every digit, makes it possible to describe it more accurately. However, it makes the training step more challenging. We propose to use the *Expectation Maximization* algorithm (EM) which fits particularly well the models containing a hidden variable. The principle is to compute an iterative loop on two consecutive steps:

the *E-step* which performs a guess on the value of the hidden variable Z for every sample of the training, and the *M-step* which performs the training of a Gaussian model for every possible value of Z.

Here is the pseudo code for every digit i in $\{0, \dots, 9\}$:

- Initialize Σ_{ij}, μ_{ij} and α_{ij} for $j \in \{1, \dots, K\}$
- Until convergence, compute:
  - E-step: For every sample x of class i in the training dataset, find $j \in \{1, \dots, K\}$ verifying $max(g_{ij}(x))$. Store this value in an array denoted W
  - M-step: For every $j \in \{1, \dots, K\}$, estimate Σ_{ij} and μ_{ij} from the samples x stored verifying $W(x) = j$.
  - For $j \in \{1, \dots, K\}$, set $\alpha_{ij} = \frac{\#(W(x)==j)}{\#training\, samples\, of\, class\, i}$ (# stands for the number of elements)

2.2 Classification step

Given a test feature vector x, the classification will be performed using the maximum likelihood criterion:

$$C(x) = argmax_{i \in \{0,\dots,9\}}(\sum_{j=1}^{K} \alpha_{ij} g_{ij}(x)),$$

where $C(x)$ stands for the output classification digit of the module for a feature vector x.

2.3 Pipeline of the task

The pipeline of the task is in the file `main.py`. Initially, the parameters are defined and the array to store the accuracy of the algorithm is initialized.

```
# parameters for the lab (can be changed for the bonus question)
size_features=8 # number of features retained from the PCA
size_training=1000; #number of samples retained for training
size_test = 1000; #number of samples retained for testing
K_max = 10 # maximum complexity of the mixture - number of GDs / digit (class)

#arrays to store results
results = zeros((K_max,2))
```

Then, the data is loaded. The data is composed of the raw images, the labels and the features for the training dataset and the test dataset.

```
#arrays to store results
results = zeros((K_max,2))
#reading of the dataset
images, labels_training, images_test, labels_test = read_dataset(size_training, size_test,
size_features);

#reading of the features extracted from dataset
features_test=numpy.array(list(csv.reader(open("test_data.csv","rb"),delimiter=',')))
        .astype('float') #loading the PCA features of the test data set
features_training=numpy.array(list(csv.reader(open("training_data.csv","rb"),delimiter=',')))
        .astype('float') #loading the PCA features of the training data set
features_test = features_test[:size_test,:size_features]
#only "size_features" first features are kept for training set
features_training = features_training[:size_training,:size_features]
     #only "size_features" first features are kept for test set
```

Then, the parameters of the model are initialized. We have to store the parameters of the Gaussian for every digit and every component of the mixture, as well as the weight.

```
# arrays containing the model for the mixture
mean_mix = zeros((10,K_max,size_features)) # mean values for the Gaussian distributions
var_mix = zeros((10,K_max,size_features)) # variance for the Gaussian distributions
alpha_mix = zeros((10,K_max)) # mixture weights for the Gaussian distributions
```

2.4 Tasks to be done

***Question 1*:** Fill the function `mix_gmm_em()` in `mix_gmm_em.py` file performing the EM algorithm to train a Gaussian mixture model. The initialization step is written already by bagging of the training set.

***Question 2*:** Fill the code of the function `classify_gmm()` in `classifier.py` file. Build the confusion matrix and compute the error rate for a mixture of $K = 1$ and $K = 2$ Gaussians on:

- the training set
- the test set

***Question 3*:** For K varying from 1 to 10, plot the error rate of the model on:

- the training dataset
- the test dataset

Comment your results.

3 Bonus question

In [1], various researchers have tried their classifiers to compete for optimal accuracy. Tune the parameter `size_features`, the size of the training and the complexity of the model (parameter K) to increase the performance of your classifiers and compare to the results obtained by other methods.

References

[1] Yann LeCun, Corina Cortest, Christopher J.C. Burges. ”The MNIST database of handwritten digits”. `http://yann.lecun.com/exdb/mnist/` Dash, Manoranjan, and Huan Liu. ”Feature selection for classification”. Intelligent data analysis 1.3 (1997): 131-156.

[2] David G. Lowe. ”Distinctive Image Features from Scale-Invariant Keypoints”. International Journal of Computer Vision (2004): 91-110