**SGN-16006 Bachelor's Laboratory Course in Signal Processing**
**Statistical Pattern Recognition Assignment**
**Recognition of Handwritten Digits**
**Heikki.Huttunen@tut.fi, January 22, 2015**

# 1 General information

In order to pass this exercise, the following prerequisites are helpful for the exercise.

- Good knowledge of MATLAB,
- Course SGN-13000/13006 Introduction to Pattern Recognition and Machine Learning (or equivalent).

This exercise studies the pattern classification methods for recognizing hand-written digits in MATLAB. Return your reports by e-mail to heikki.huttunen@tut.fi by 20.3.2015. The report should contain the following:

1. Name, student number and email
2. Explanation of the methods you have used, results, conclusions, and answers to the questions (if any). There are tasks and questions in the exercise instructions that help you while writing, but try to avoid just listing the answers; instead, write a full report (in pdf format). This is the main deliverable of this exercise.

   Also add a table describing the test data classification accuracies of three methods: 1) Naive Bayes, 2) Logistic Regression with small $\lambda$ and 3) Logistic Regression with cross-validated $\lambda$.
3. Commented MATLAB code (as m-files or a report appendix).

There are four sessions, where the instructor will be present for your questions. You can enter and leave at any time during these sessions. The exercises are in a computer room, so you can work on the assignment for the whole time with supervision if you wish. The times are the following.

- Thu Jan 29, 2015 14:00 - 16:00 TC303: QA session # 1
- Thu Feb 12, 2015 14:00 - 16:00 TC303: QA session # 2
- Thu Feb 26, 2015 14:00 - 16:00 TC303: QA session # 3
- Mon Mar 16, 2015 14:00 - 16:00 TC303: QA session # 4

## 2 Data

The data for the exercise can be downloaded from the following URL:

`http://www.cs.tut.fi/courses/SGN-16006/MNIST.zip`

The file contains the famous MNIST dataset[1], which consists of 60000 training images and 10000 test images representing hand-written numbers between 0 and 9. Additionally, the true labels for each image are available as well. Figure 1 shows example images from the training dataset and also the true labels for each file.

**WARNING**: the zip file size is 94MB, so do not extract it into your TUT home directory, because it may hit the quota limit and slow down your login.

More specifically, the zip file contains the following items.

- Training images are in the subdirectory `train_images`. There are altogether 60000 images in BMP format with resolution $28 \times 28$.
- There is also another set of 10000 images for testing. These files are in the subdirectory `test_images`. There are altogether 10000 images in BMP format with resolution $28 \times 28$.
- The class labels for the training images are listed in the text file `TRAIN.csv`. Each row contains one filename and the corresponding class ID (number 0...9) separated by comma:

```
train_images/TRAIN_00001.bmp,5
train_images/TRAIN_00002.bmp,0
train_images/TRAIN_00003.bmp,4
train_images/TRAIN_00004.bmp,1
train_images/TRAIN_00005.bmp,9
...
```

- The class labels for the test images are listed in the text file `TEST.csv`. The file format is the same as that of `TRAIN.csv`.

## 3 Theory

Any pattern is a pair comprising of an observation and a label. Pattern recognition is, by definition, inferring labels from observations. Designing a pattern recognition system means establishing a mapping from measurement space into the space of potential labels. The basic operations (components) in pattern recognition are feature extraction and classification. To understand the task of designing such a system, we briefly introduce the problems that each of these components must solve. Additional information can be found for example in the book [1].

[1] `http://yann.lecun.com/exdb/mnist/`

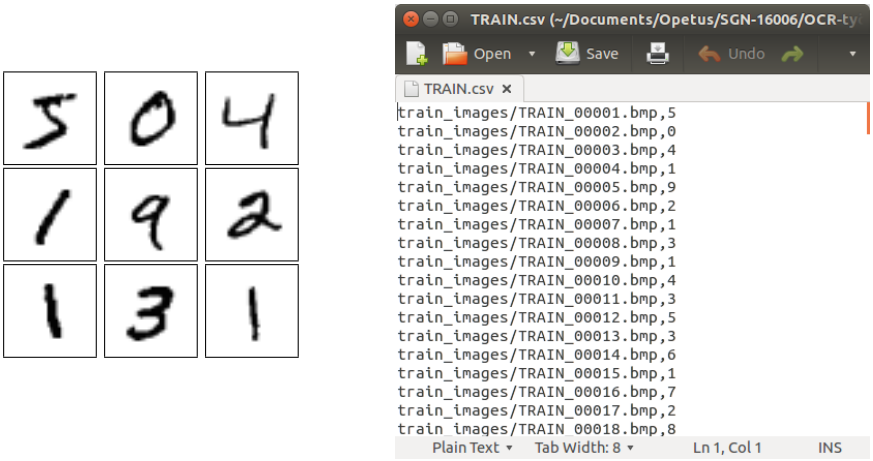


Figure 1: Left: Example handwritten images from the dataset. Right: Class labels opened in a text editor. The numbers on the left correspond to first rows of the label text file.

### 3.1 Feature extraction

The traditional goal of feature extractor is to characterize an object to be recognized by measurements whose values are very similar for objects in the same category, and very different for objects in different categories. This leads to the idea of seeking distinguishing features that are invariant to irrelevant transformations. Feature extraction aims to select from the data set the optimal subset of features that can achieve the highest accuracy. In other words feature extraction step consists of mapping the M-dimensional vector of observations $\mathbf{x} = (x_1, x_2, \ldots, x_M)$ (e.g., a $10 \times 10$ bitmap would be represented by a $M = 100$-dimensional vector) into a new K-dimensional feature description $\mathbf{z} = (z_1, z_2, \ldots, z_K)$, suitable for the given task. Note that the extraction step is not always necessary: in some cases, the original representation is well suited to solve the recognition task—possibly with some normalization step.

### 3.2 Pattern classification

The task of the classifier is to use the feature vector provided by the feature extractor to assign the object to a category. In this exercise you will concentrate on designing a 10-category classifier to recognize between different digits represented in binary bitmap form. The studied classifiers are described below. Each method assumes that the input is an *observation matrix* $\mathbf{X} \in \mathbb{R}^{N \times P}$:

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1P} \\ x_{22} & x_{22} & \cdots & x_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NP} \end{pmatrix},$$

where each row represents one training image. As an example, our data could consist of $N = 60000$ rows with $P = 28 \times 28$ grayscale values each (assuming the $28 \times 28$ images are simply vectorized to $P = 784$ numbers).

The class labels corresponding to the 60000 images are stored in a vector as $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ with $y_1 \in \{0, \ldots, 9\}$.

**Naive Bayes Classifier**

The second classifier used in this exercise is the Naive Bayes Classifier. The method makes the naive assumption that all the features would statistically independent from each other. Although this is not true for our data (neighboring pixels have high correlation), the NB classifier can still be relatively accurate.

Naive Bayes classifier is a statistical method in the sense that it attempts to model the probabilities of a sample belonging to a class. This time, the model can be written as follows:

$$P(y = c \mid \mathbf{x}) = \frac{1}{Z} P(c) \prod_{n=1}^{P} P(x_n \mid y = c),$$

for $\mathbf{x} = (x_1, x_2, \ldots, x_P)$. In other words, the probability is simplified to P terms, each trying to predict the class based on a single feature (i.e., pixel) only.

The significance of the factorization is that now each term in the product can be estimated using a lookup table. The table is built at training time by storing the histogram of grayscale values for each pixel. As an example, Figure 3 shows the histogram of all upper-left pixels of all bitmaps from class zero. In comparison, the corresponding histogram for class "one" is shown as well. If this were all the information we have, the decision rule between the two classes would be straightforward: The class is "one" if center pixel is greater than (about) 180, otherwise class is "zero".

**Logistic Regression**

Similar to the Naive Bayes method, the logistic regression classifier is also a statistical classification method, whose decision boundary is linear [2]. Despite its simplicity, it is widely used and can achieve good accuracy in selected problems. Due to its mathematical simplicity it is often a method of choice with big data sets[2].

For a two-class case, logistic regression models the probability of class membership of a sample $\mathbf{x}$ by the logistic function:

$$f(x) = \frac{1}{1 + \exp(\mathbf{a}^T \mathbf{x} + b)},$$

where $\mathbf{a} \in \mathbb{R}^P$ is a coefficient vector and $b$ a constant, both learned during training.

For a multinomial case, logistic regression for C classes models the probability $P(y = c \mid \mathbf{x})$ of the sample $\mathbf{x}$ belonging to class $c \in \{1, 2, \ldots, C\}$ as

$$P(y = c \mid \mathbf{x}) = \frac{\exp(\mathbf{a}_c^T \mathbf{x} + b_c)}{\sum_{k=1}^{C} \exp(\mathbf{a}_k^T \mathbf{x} + b_k)},$$

where the model parameters $\mathbf{a}_c \in \mathbb{R}^P$ and $b_c \in \mathbb{R}$ are also learned from training data.

[2]See, e.g., the open sourced machine learning platform from LinkedIn: https://engineering.linkedin.com/large-scale-machine-learning/open-sourcing-ml-ease

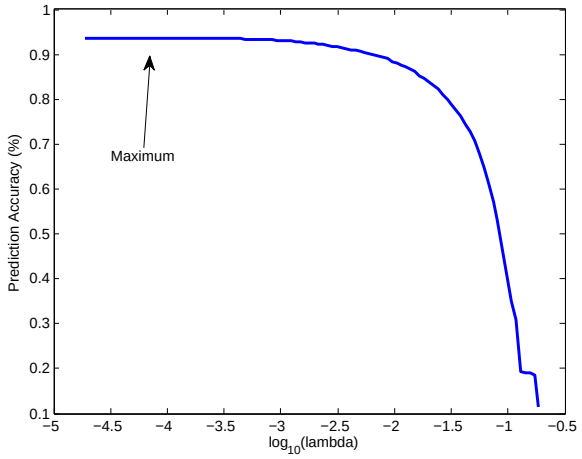


Figure 2: The accuracy of regularized logistic regression as a function of the regularization parameter $\lambda$.

In our case, there are altogether 10 classes, so we wish to learn the following 10 functions:

$$\begin{aligned}
\mathrm{P}(\text{"bitmap is a zero"} \mid \mathbf{x}) &= \frac{\exp(\mathbf{a}_0^T\mathbf{x} + b_0)}{\sum_{k=1}^{C}\exp(\mathbf{a}_k^T\mathbf{x} + b_k)} \\
\mathrm{P}(\text{"bitmap is a one"} \mid \mathbf{x}) &= \frac{\exp(\mathbf{a}_1^T\mathbf{x} + b_1)}{\sum_{k=1}^{C}\exp(\mathbf{a}_k^T\mathbf{x} + b_k)} \\
&\vdots \\
\mathrm{P}(\text{"bitmap is a nine"} \mid \mathbf{x}) &= \frac{\exp(\mathbf{a}_9^T\mathbf{x} + b_9)}{\sum_{k=1}^{C}\exp(\mathbf{a}_k^T\mathbf{x} + b_k)}
\end{aligned} \tag{1}$$

The functions are thus parameterized by the ten vectors $\mathbf{a}_0, \ldots, \mathbf{a}_9$ and ten constants $b_0, \ldots, b_9$, and their discovery is the task of the training step.

The training of a logistic regression classifier is straightforward in Matlab. We will use the GLMNET package as described below. In the the GLMNET framework, the training of the logistic regression model consists of estimating the unknown parameters by maximizing the penalized log-likelihood function

$$\max_{\{\mathbf{a}_k, b_k\}_1^K} \left[ \sum_{i=1}^{N} \log\left[\mathrm{P}(y_i = k \mid \mathbf{x}_i)\right] - \lambda \|\mathbf{a}_k\|_1 \right].$$

In other words, the goal is to find 10 parameter vectors $\{\mathbf{a}_k, b_k\}$ for $k = 1, 2, \ldots, 10$ that together maximize the log-likelihood of classes given the data. The latter term is a *regularizer*, whose role is to prevent over-fitting the model to the training data by giving additional penalty for large coefficient values. The penalization has been shown to improve accuracy for test data, and in particular the L1-norm penalty has been of great interest recently.

The amount of penalty is determined by the coefficient $\lambda$. If $\lambda$ is large, the weights are forced to small (close to zero) values, while a small $\lambda$ allows more freedom to choose larger coefficients. The classification accuracy on the test set may typically look like the plot on Figure 2.

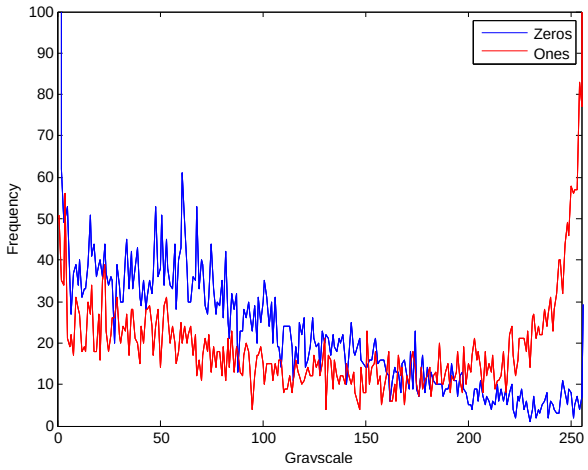


Figure 3: Histogram of the center pixel of all zeros and ones in the training set.

For this assignment, it is important to note that the GLMNET returns a set of classifiers instead of a single one.

## 4 Task 1: Use a Naive Bayes Classifier to Classify Test Images

The first task is to train and apply a naive Bayes classifier. To this aim, you can use Matlab's implementation documented here:

`http://se.mathworks.com/help/stats/naivebayes-class.html`

However, the preprocessing requires the following steps:

1. Load all training images and their class labels into Matlab. The file names and labels can be parsed from `TRAIN.csv` using `textscan` function (or your own loop with functions `fopen`, `fgetl` and `strsplit`).
2. While loading, remove the blank borders of each image. That is, most images have full rows and columns at the borders that are all white (all pixels have value 255). They should be cropped away.
3. After cropping to a tight bounding box, resize all images to size $10 \times 10$.

As a result of loading the data, you should have a matrix `X` of size $60000 \times 100$ holding the images. Also, there should be a label vector of size $60000 \times 1$ with the class labels (0 through 9).

Load the test data in a similar manner. Train the classifier with training data, and apply it to the test image data. Estimate the classification accuracy by

```
accuracy = mean(yHat == yTest);
```

where `yTest` is the 10000-element vector with true labels for the test samples and `yHat` are the labels predicted by the NB classifier. The accuracy of the classifier should be above 80 % if all went well.

Also plot the confusion matrix to your report (Matlab: `confusionmat`).

# 5 Task 2: Use a Logistic Regression Classifier to Classify Test Images

The second task is to use a logistic regression classifier instead of the naive Bayes of task 1. Matlab has its own LR classifier (`lassoglm`), but it is a lot slower than a state-of-the-art implementation available at

http://web.stanford.edu/~hastie/glmnet_matlab/

Use this package to train the LR model. The training is done using the `glmnet` function as follows

```
model = glmnet(X_train, y_train, 'multinomial');
```

Note, however, that the glmnet package assumes that the labels start from 1. Since our labels are 0...9, you will have to add one to the labels both at the training and testing stage.

Run the training. This may take up to 10 minutes depending on the speed of the computer.

As a result, the trained classifier will be stored in variable `model`. The variable contains the (among others) the following fields:

- `model.lambda`: a 100-element vector containing all used values for the penalty parameter $\lambda$.
- `model.beta`: a cell of 10 matrices of size $100 \times 100$. Each 100-element column represents a set of coefficients $\mathbf{a}_k$ (see Eq. 1) for the $10 \times 10$ pixels. There are altogether 100 such vectors, one for each value of parameter $\lambda$.
- `model.a0`: a vector containing the constants $b_k$ for the logistic regression model of Equation 1; one for each value of parameter $\lambda$.

Predict the class labels for all samples in the training set using the trained classifier. Use the coefficients that correspond to the smallest $\lambda$ value. The prediction can be done using the glmnetPredict function:

```
yHat = glmnetPredict(model, X_test, lambda, 'class');
```

The accuracy should be well over 90 %.

Also plot the confusion matrix to your report (Matlab: `confusionmat`).

## 6 Task 3: Select the $\lambda$ parameter optimally using cross-validation

In Task 2, the prediction was done using the smallest $\lambda$ parameter corresponding to least amount of regularization. A better approach is to test the suitability of each and select the most accurate solution. This could be done by randomly splitting the training set further to two parts: *e.g.,* 80 % for training and 20 % for testing. However, a more standardized approach is to use K-fold cross-validation, where each training sample serves as a test sample.

More specifically, K-fold cross-validation does the following:

1. Split the data randomly into K parts. These parts are called *folds*.
2. For each fold $k = 1, 2, \ldots, K$:
   (a) Train with all samples except those on fold $k$
   (b) Predict the class labels for all samples on fold $k$
   (c) Calculate the accuracy of predictions
3. Average the accuracies of the $k$ folds.

In this task, you have to estimate the accuracy of classification for all values of lambda using 5-fold cross-validation and plot a curve similar to Figure 2. A Matlab template for doing this is below.

```
numSamples = size(X_train, 1);
numFolds = 5;

% The crossvalind function returns a vector (of length numSamples)
% telling which fold each sample belongs to.
cvIdx = crossvalind('Kfold', numSamples, numFolds);

% We will store the estimated accuracies here:
cvAccuracies = [];

% For each fold...
for fold = 1:numFolds

    % TODO: Form 2 vectors here: one containing the train indices
    % ("trainIdx") and one containing the test indices ("testIdx")

    % TODO: Train the classifiers with data in X_train(trainIdx, :)

    % For each value of lambda:
    for k = 1:length(model.lambda)

        % TODO: Predict the test fold classes for this lambda;
        % Store the predictions to "yHat"

        % Calculate the accuracy for this test fold:
        accuracy = mean(yHat == y_train(testIdx, :));
```

```
        % Add to the accuracy matrix:
        cvAccuracies(fold, k) = accuracy;
    end

end

% Calculate the average of each column.
cvAccuracies = mean(cvAccuracies, 1);

% Plot graph
plot(log10(model.lambda), cvAccuracies);
```

Also plot the confusion matrix to your report (Matlab: `confusionmat`).

## References


[1] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification*, Wiley Interscience, 2002.

[2] T. Hastie, R. Tibshirani, J. Friedman, *The elements of statistical learning*, (Vol. 2, No. 1), 2009. Available: http://statweb.stanford.edu/~tibs/ElemStatLearn/.