DD2431 Machine Learning
Lab 3: Bayes Classifier & Boosting

Staffan Ekvall
based on a lab by Frank Hoffmann

September 26, 2010

1 Introduction

In this lab you will implement a Bayes Classifier and the Adaboost algorithm that improves the performance of a weak classifier by aggregating multiple hypotheses generated across different distributions of the training data. Some predefined functions for visualization and basic operations are provided, but you will have to program the key algorithms yourself. Predefined functions and data files for this lab are available in the directory `/info/ml10/labs/bayesboost/`.

In this exercise we will work with two images. Start by looking at the images.

```
>>hand = imread('hand.ppm', 'ppm');
>>book = imread('book.ppm', 'ppm');
>>imagesc(hand);
>>figure;
>>imagesc(book);
```

One image shows a hand holding a book and the other shows the hand alone. This example is taken from a computer vision application, in which the user demonstrates a book to the computer that has to learn to recognize the book in new images. The best object recognition performance is achieved when the training image shows the book alone. We would like to remove the hand holding the book from the image.

The idea is to look at the color of each pixel and then estimate whether it is more likely to belong to the hand-image than the book-image. If so, we will set the pixel to black and thereby remove all hand-pixels.

Each pixel is described by two discrete features (x_1, x_2), corresponding to normalized red and green color intensities. Each pixel belongs to one of two classes $\{0, 1\}$, corresponding to the classes *hand* and *book*.

This exercise consists of two parts. In the first part, you will classify the data using a simple Bayes classifier. In the second part, you will implement a boosting algorithm. Begin by viewing the data using the following matlab code:

```
>>data1 = normalize_and_label(hand, 0);
>>data2 = normalize_and_label(book, 1);
>>test_data = [data1; data2];
>>figure;
>>hold on;
>>plot(data2(:,1), data2(:,2), '.');
>>plot(data1(:,1), data1(:,2), '.r');
>>legend('Hand holding book', 'Hand');
>>xlabel('green');
>>ylabel('red');
```

The function *normalize_and_label* computes the normalized red and green intensities.

$$r_{norm} = \frac{r}{r+g+b} \qquad g_{norm} = \frac{g}{r+g+b} \tag{1}$$

It also labels each pixel telling which image it belongs to. All black pixels are discarded. Note that the hand pixels in the book-image are incorrectly labeled as the class *book*. We will rely on our classifier to distinguish these pixels from the correctly labeled pixels.

2 Bayes classifier

In Bayesian learning we are interested in determining the most probable hypothesis h given the observed training data D. Let us introduce a little notation. With $P(h)$ (later in the text $p(c_i)$) we denote the prior probability of a hypothesis/class, before we observe any data. with $P(D)$ (later $p(\vec{x})$) we denote the prior probabilty that training data D will be observed. Next we write $P(D|h)$ (later $p(\vec{x}|c_i)$) to denote the probability of observing data D given that the hypothesis h holds. In Bayesian learning $P(D|h)$ is also called the *likelihood* of the data, as it states how likely it is to observe data D given h. In Bayesian learning we are primarily interested in computing $P(h|D)$, namely the probability that hypothesis h holds given the observed data D. $P(h|D)$ (later $p(c_i|\vec{x})$) is called the *posterior probability* of h, as it reflects our confidence that h holds after we have seen D. The *maximum a posteriori* (MAP) hypothesis is the most probable among all possible hypotheses.

$$h_{MAP} = argmax_{h \in H} P(h|D) \tag{2}$$

The *maximum likelihood* hypothesis (ML) is one that maximizes the likelihood $P(D|h)$ of the data.

$$h_{ML} = argmax_{h \in H} P(D|h) \tag{3}$$

Bayes theorem provides a way to calculate the posterior probability $P(h|D)$, from the prior probability $P(h)$ together with $P(D|h)$ and $P(D)$.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{4}$$

If there is only one thing that you have to understand and remember about Bayesian learning, it is that equation.

The structure of a Bayes classifier is determined by the likelihoods $p(\vec{x}|c_i)$ as well as the prior class probabilities $p(c_i)$, where $\vec{x}$ is the feature vector and c_i a given class. The most prominent density function is the multivariate normal or Gaussian density.

We will approximate the data with gaussian functions, and we will assume that the features $x_1, ..., x_N$ are uncorrelated. You will derive the maximum likelihood estimates for class-conditional Gaussians. We start with a model for a discrete class c_i and a real-valued vector of N features $\vec{x} = \{x_1, ..., x_N\}$. In the following we use the lower index n to denote the n-th feature x_n, and the upper index m to denote the m-th training instance $\vec{x}^m$ in the dataset. The prior α_i is simply the frequency of class c_i in the dataset.

$$p(c_i) = \alpha_i \tag{5}$$

The likelihood density functions for a model (hypothesis) with diagonal covariance matrix are given by

$$p(\vec{x}|c_i) = \Pi_{n=1}^{N} \left(\frac{1}{\sqrt{2\pi\sigma_{in}^2}}\right) \cdot e^{-\sum_{n=1}^{N} \frac{(x_n - \mu_{in})^2}{2\sigma_{in}^2}} \tag{6}$$

Here, σ_{in} denotes σ for class i, and feature n. We apply Bayes rule to compute the posterior probabilties from the likelihoods and the prior probabilties:

$$p(c_i|\vec{x}) = p(\vec{x}|c_i)\frac{p(c_i)}{p(x)} = \Pi_{n=1}^{N}(\frac{1}{\sqrt{2\pi\sigma_{in}^2}}) \cdot e^{-\sum_{n=1}^{N}\frac{(x_n-\mu_{in})^2}{2\sigma_{in}^2}} \cdot \frac{\alpha_i}{p(\vec{x})} \tag{7}$$

This is the posterior for a single feature vector $\vec{x}$. Let us now assume that we have several independent feature vectors $D = \{\vec{x}^1, \ldots, \vec{x}^{M_i}\}$ belonging to class c_i. M_i is the number of instances that belong to class c_i.

The posterior probability of c_i given the entire data D is proportional to the product of the individual posteriors

$$P(c_i|D) \sim \Pi_{\{m|c_m=c_i\}}^{M_i} p(c_i|\vec{x}^m) \tag{8}$$

The maximum a posteriori hypothesis (MAP) $\vec{\mu_i^*}, \vec{\sigma_i^*}$ that best explains the data is computed by maximizing equation 8 with respect to $\vec{\mu_i}, \vec{\sigma_i}$.

$$\{\vec{\mu_i^*}, \vec{\sigma_i^*}\} = \mathrm{argmax}_{\vec{\mu_i},\vec{\sigma_i}} \mathrm{p(c_i|D)} \tag{9}$$

We now apply a common transformation, namely rather than maximizing the above expression we maximize the less complicated logarithm of equation 8.

$$\log p(c_i|D) = -\frac{M_i}{2}\sum_{n=1}^{N}\log 2\pi\sigma_{in}^2 - \sum_{\{m|c_m=c_i\}}^{M_i}\sum_{n=1}^{N}\frac{(x_n^m-\mu_{in})^2}{2\sigma_{in}^2} + \log\alpha_i - \log p(\vec{x}^m) \tag{10}$$

To find the maximum we equate the partial derivatives of $\log p(c_i|\vec{x})$ with respect to μ_{in} and σ_i with zero. Since the prior of the data $\log p\vec{x^m}$ does not depend on our model the last term in equation 10 disappears.

$$\frac{\partial \log p(c_i|D)}{\partial \mu_{in}} = \sum_{\{m|c_m=c_i\}}^{M_i} -\frac{(x_n^m-\mu_{in})}{\sigma_{in}^2} = 0 \tag{11}$$

$$\frac{\partial \log p(c_i|D)}{\partial \sigma_{in}} = -\frac{M_i}{\sigma_i} + \sum_{\{m|c_m=c_i\}}^{M_i}\frac{(x_n^m-\mu_{in})^2}{\sigma_{in}^3} = 0 \tag{12}$$

From these equations we obtain the maximum posterior model (hypothesis)

$$\mu_{in}^* = \frac{\sum_{\{m|c_m=c_i\}}^{M_i} x_n^m}{M_i} = E[x_n] \tag{13}$$

$$\sigma_{in}^{*\,2} = \frac{\sum_{\{m|c_m=c_i\}}^{M_i}(x_n^m-\mu_{in}^*)^2}{M_i} = E[(\vec{x_n}-\vec{\mu_{in}^*})^2] \tag{14}$$

Assignment 1: Write a function `bayes(data)` that computes the maximum posterior (MAP) parameters μ^*_{in} and σ^*_{in} for a given dataset D. Assume the usual data format for the parameter `data`, namely that the first N columns contain the features $x_1, ...x_N$, and the last $N+1$-th column contains the classification c. The signature of the function `bayes(data)` in Matlab looks like

```
function [mu, sigma] = bayes(data)
```

with return values `mu` and `sigma` (CxN-vectors). C denotes the number of classes and N is the number of features.

Apply your function to the data, set `bayes(test_data)`, where `test_data` is the combined data set `[data1; data2]`. Compute
$\mu^*_{01} = \ldots$
$\mu^*_{02} = \ldots$
$\sigma^*_{01} = \ldots$
$\sigma^*_{02} = \ldots$
$\mu^*_{11} = \ldots$
$\mu^*_{12} = \ldots$
$\sigma^*_{11} = \ldots$
$\sigma^*_{12} = \ldots$

Verify that your estimations match the data by plotting a 95%-confidence interval:

```
>> [mu sigma] = bayes(test_data);
>> theta = [0:0.01:2*pi];
>> x1 = 2*sigma(1,1)*cos(theta) + mu(1,1);
>> y1 = 2*sigma(1,2)*sin(theta) + mu(1,2);
>> x2 = 2*sigma(2,1)*cos(theta) + mu(2,1);
>> y2 = 2*sigma(2,2)*sin(theta) + mu(2,2);
>> plot(x1, y1, 'r');
>> plot(x2, y2);
```

The next step is to compute the discriminant functions for predicting the class of a unseen instance $\vec{x}$. The most likely classification is the one that maximizes the posterior $p(c_i|\vec{x})$ from equation 7.

$$c^* = argmax_{c_i}\, p(c_i|\vec{x}) \tag{15}$$

As before we will find it easier to work with the logarithm of this expression

$$log(p(c_i|\vec{x})) = \log \alpha_i - \frac{N \log(2\pi)}{2} - \sum_{n=1}^{N} \log \sigma_{in} - \sum_{n=1}^{N} \frac{(x_n - \mu_{in})^2}{2\sigma_{in}^2} - \log p(\vec{x}) \tag{16}$$

We can drop the terms $-\frac{N \log(2\pi)}{2}$ and $\log p(\vec{x})$ since they do not depend on $\alpha_i, \vec{\mu_i}, \vec{\sigma_i}$ and are therefore identical for all classes. We obtain the discriminant functions

$$g_i(\vec{x}) = \log \alpha_i - \sum_{n=1}^{N} \log \sigma_{in} - \sum_{n=1}^{N} \frac{(x_n - \mu_{in})^2}{2\sigma_{in}^2} \tag{17}$$

Write a function `discriminant(data, mu, sigma, p)` that computes the discriminant functions g_i. The parameter `data` is the MxN-matrix of M feature vectors $\vec{x}^m$, `mu` and `sigma` are the CxN-matrices of means and standard deviations, respectively, and `p` is an $Cx1$-vector of prior class probabilities α_i. Again, C denotes the number of different classes. Each class c_i is associated with one Gaussian described by the pair $\vec{\mu_i}, \vec{\sigma_i}$ and its prior class probability $p(c_i)$. The signature of `discriminant(data, mu, sigma, p)` in Matlab looks like

```
function g = discriminant(data, mu, sigma, p)
```

where the return value `g` is a MxC-matrix.

Assignment 2: Train the Bayes classifier with the dataset `test_data`, and compute the classification error. The prior class probabilities are computed by using the Matlab function `prior(data)`. We use the Matlab function `max` to compute the maximum discriminant value and the index `class` of the optimal class, which we compare with the known classification `test_data(:,3)`.

```
>>[M N] = size(test_data);
>>p = prior(test_data);
>>g = discriminant(test_data(:,1:2), mu, sigma, p);
>>[dummy class] = max(g, [], 2);
>>class = class - 1;
>>error_test = 1.0-sum(class == test_data(:,end))/M
```

error_test = ...

Note that since the image *hand holding book* also shows the hand, we can never achieve *error_test* = 0. This is not a problem. In fact, it is the misclassified pixels that we later will set to black, in order to remove the hand from the image *hand holding book*.

Overlay the decision boundary $X = \{\vec{x} : g_0(\vec{x}) = g_1(\vec{x})\}$ that separates both classes using the following Matlab code. If you have closed the figure window, plot the data again with the code from page 2.

```
>>ax = [0.2 0.5 0.2 0.45];
>>axis(ax);
>>x = ax(1):0.01:ax(2);
>>y = ax(3):0.01:ax(4);
>>[z1 z2] = meshgrid(x, y);
>>z1 = reshape(z1, size(z1,1)*size(z1,2), 1);
>>z2 = reshape(z2, size(z2,1)*size(z2,2), 1);
>>g = discriminant([z1 z2], mu, sigma, p);
>>gg = g(:,1) - g(:,2);
>>gg = reshape(gg, length(y), length(x));
>>[c,h] = contour(x, y, gg, [0.0 0.0]);
>>set(h, 'LineWidth', 3);
```

Finally, we will see how well our classifier serves its purpose. Begin by normalizing the image *hand holding book*. This time we keep the black pixels, so that we later may display the image.

```
>>book_rg = zeros(size(book,1), size(book,2), 2);
>>for y=1:size(book,1)
>>  for x=1:size(book,2)
>>    s = sum(book(y,x,:));
>>    if (s > 0)
>>      book_rg(y,x,:) = [double(book(y,x,1))/s double(book(y,x,2))/s];
>>    end
>>  end
>>end
```

Then we classify all pixels in the image, and create a *mask* of all pixels that are classified to belong to the other image (`hand.ppm`).

```
>>tmp = reshape(book_rg, size(book_rg,1)*size(book_rg,2), 2);
>>g = discriminant(tmp, mu, sigma, p);
>>gg = g(:,1) - g(:,2);
```

```
>>gg = reshape(gg, size(book_rg,1), size(book_rg,2));
>>mask = gg < 0;
>>mask3D(:,:,1) = mask;
>>mask3D(:,:,2) = mask;
>>mask3D(:,:,3) = mask;
```

Finally, we apply the mask to the original image, so that we can show the result in color.

```
>>result_im = uint8(double(book) .* mask3D);
>>figure;
>>imagesc(result_im);
```

3 Boosting

Boosting aggregates multiple hypotheses generated by the same learning algorithm invoked over different distributions of training data into a single composite classifier. Boosting generates a classifier with a smaller error on the training data as it combines multiple hypotheses which individually have a larger error. Boosting requires *unstable* classifiers which learning algorithm is sensitive to changes in the training examples.

The idea of boosting is to repeatedly apply a weak learning algorithm on various distributions of the training data and to aggregate the individual classifiers into a single overall classifier. After each iteration the distribution of training instances is changed based on the error the current classifier exhibits on the training set. The weight w^m of an instance (x^m, c^m) specifies its relative importance, which can be interpreted as if the training set would contain w^m identical copies of the training example (x^m, c^m). The weights w^m of correctly classified instances (x^m, c^m) are reduced, whereas those of incorrectly classified instances are increased. Thereby the next invocation of the learning algorithm will focus on the incorrect examples.

In order to be able to boost the Bayes classifier, the algorithm for computing the MAP parameters and the discriminant function has to be modified such that it can deal with fractional (weighted) instances. Assume, that ω^m is the weight assigned to the m-th training instance. Without going into a straight forward detailed derivation the equations 14 for the MAP parameter with weighted instances become:

$$\mu_{in}^* = \frac{\sum_{\{m|c_m=c_i\}}^{M_i} \omega^m x_n^m}{\sum_{\{m|c_m=c_i\}}^{M_i} \omega^m} = E_\omega[x_n]$$

$$\sigma_{in}^{*\,2} \quad = \quad \frac{\sum_{\{m|c_m=c_i\}}^{M_i} \omega^m (x_n^m - \mu_{in}^*)^2}{\sum_{\{m|c_m=c_i\}}^{M_i} \omega^m} = E_\omega[(\vec{x_n} - \vec{\mu}_{in}^*)^2] \tag{18}$$

Assignment 3: Extend the old `bayes` function to `bayes_weight(data, w)` that handles weighted instances. Again `data` is MXN-matrix of feature vectors and `w` is a $Mx1$-vector of weights. The signature in Matlab looks like

```
function [mu, sigma] = bayes_weight(data, w)
```

where the return parameters `mu` and `sigma` are identical to the old function `bayes`. The function computes the maximum posterior parameters μ_{in}^* and σ_{in}^* for a dataset D according to the equations in 18. Assume the usual data format for the parameter `data`, namely that the first N columns contain the features $x_1, ...x_N$, and the last (N+1)-th column contains the classification c. Test your function `bayes_weight(data, w)`, for a uniform weight vector with $\omega = 1/M$. The MAP parameters should be identical to those obtained with the previous version of `bayes`.

The Adaboost algorithm repeatedly invokes a weak learning algorithm, in our case `bayes_weight` and after each round updates the weights of training instances (x^m, c^m). Adaboost proceeds in the following manner.

- Initialize all weights uniformly $\omega_1^m = 1/M$.
- train weak learner using distribution ω_t
- get weak hypothesis h_t and compute its error ϵ_t with respect to the weighted distribution ω_t. In case of the Bayes classifier a single hypothesis h_t is represented by $\mu_{tin}^*, \sigma_{ti}^*$.

  $$\epsilon_t = 1 - \sum_{m=1}^{M} \omega_t^m \delta(h_t(x^m), c^m) \tag{19}$$

  where $h_t(x^m)$ is the classification of instance x^m made by the hypothesis h_t. The function $\delta(h_t(x^m), c^m)$ is 1 if $h_t(x^m) = c^m$ and 0 otherwise.
- choose $\alpha_t = 1/2 \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- update weights according to

  $$\omega_{t+1}^m = \frac{\omega_t^m}{Z_t} * \left\{ \begin{array}{ll} e^{-\alpha_t} & \mathrm{if\ h_t(x^m) = c^m} \\ e^{\alpha_t} & \mathrm{if\ h_t(x^m) \neq c^m} \end{array} \right. \tag{20}$$

where Z_t is a normalization factor, chosen such that ω_{t+1} becomes a distribution $\sum_m \omega_{t+1}^m = 1$.

The overall classification of the boosted classifier of an unseen instance x is obtained by aggregating the votes casted by each individual Bayes classifier $h_t = \mu_{tin}^*, \sigma_{ti}^*$. As we have higher confidence in classifiers that have a low error (large α_t), their votes count relatively more. The final classification $H(x)$ is the class c_{max} that receives the majority of votes

$$H(x) = c_{max} = argmax_{c_i} \sum_{t=1}^{T} \alpha_t \, \delta(h_t(x), c_i) \tag{21}$$

Assignment 4: Implement the Adaboost algorithm and apply it to the Bayes classifier. Design a Matlab function `adaboost(data, T)` that generates a set of boosted hypothesis, where the parameter `T` determines the number of hypotheses. The signature in Matlab looks like

```
function [mu, sigma, p, alpha, classes] = adaboost(data, T)
```

where `mu`, `sigma` and `p` contain the `T` sets of MAP parameters and priors. The return parameter α_t holds the classifier vote weights α_t and `classes` the set of unique classes. In particular `mu` and `sigma` are $TxCxN$-arrays, prior is a TxC-array, `alpha` is a $Tx1$-vector and `classes` is a $Cx1$-vector. Note that also the priors have to be weighted with w. Use `prior(data, w)`, this time with the optional argument, the weights w.

Design a function
`adaboost_discriminant(data, mu, sigma, p, alpha, classes, T)` that classifies the instances in `data` by means of the aggregated boosted classifier according to equation 21. The signature in Matlab looks like

```
function c = adaboost_discriminant(data, mu, sigma, p, alpha, classes, T)
```

where `c` is a $Mx1$-vector that contains the class predicted for the MxN-feature vector `data`. The other parameters have the same dimensions as in `adaboost`.

Notice, that you have to compute and store all the hypothesis generated with `bayes_weight(data, w)` for each of the different distributions ω_{t+1} and later aggregate their classifications to obtain the overall classification. Compute the classification accuracy of the boosted classifier and compare it with those of the basic classifier (see assignment 2).

```
>>T = 6;
>>[mu sigma p alpha classes] = adaboost(test_data, T);
>>class = adaboost_discriminant(test_data(:,1:N-1), mu,
                                sigma, p, alpha, classes, T);
>>boost_error_test = 1.0-sum(class == test_data(:,end))/M
```

boost_error_test = ...

Plot the decision boundary of the boosted classifier using the Matlab code.

```
>>figure;
>>hold on;
>>plot(data2(:,1), data2(:,2), '.');
>>plot(data1(:,1), data1(:,2), '.r');
>>legend('Hand holding book', 'Hand');
>>ax = [0.2 0.5 0.2 0.45];
>>axis(ax);
>>x = ax(1):0.01:ax(2);
>>y = ax(3):0.01:ax(4);
>>[z1 z2] = meshgrid(x, y);
>>z1 = reshape(z1, size(z1,1)*size(z1,2), 1);
>>z2 = reshape(z2, size(z2,1)*size(z2,2), 1);
>>g = adaboost_discriminant([z1 z2], mu, sigma, p, alpha, classes, T);
>>gg = reshape(g, length(y), length(x));
>>[c,h] = contour(x, y, gg, [0.5 0.5]);
>>set(h, 'LineWidth', 3);
```

Compare the decision boundary of the boosted classifier with that of the basic Bayesian classifier. Explain why the decision boundrary of the original classifier is much smoother than the decision boundrary of the boosted classifier.

View the performance of the boosted classifier on our hand removal problem. If `book_rg` somehow has vanished from the computer's memory, recompute it with the code described in the first part of the lab.

```
>>tmp = reshape(book_rg, size(book_rg,1)*size(book_rg,2), 2);
>>g = adaboost_discriminant(tmp, mu, sigma, p, alpha, classes, T);
>>gg = reshape(g, size(book_rg,1), size(book_rg,2));
>>mask = gg > 0.5;
```

```
>>mask3D(:,:,1) = mask;
>>mask3D(:,:,2) = mask;
>>mask3D(:,:,3) = mask;
>>result_im = uint8(double(book) .* mask3D);
>>figure;
>>imagesc(result_im);
```