

2D1431 Machine Learning

Lab 2: Bayes Classifier & Boosting

Frank Hoffmann
e-mail: hoffmann@nada.kth.se

November 18, 2002

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. During the examination with the lab assistant, you will present the results, answers to the questions and the code that you wrote.

It is assumed that you are familiar with the basic concepts of Bayesian learning and that you have read chapter 6 in the course book *Machine Learning* [2] and the papers by SCHAPIRE [4] and QUINLAN [3] on boosting. If you look for additional literature, chapters 2, 3 and 9 in the book by DUDA et al [1] are worthwhile reading.

In this exercise we will use a synthetic dataset with two continuous features (x_1, x_2) and two classes $\{0, 1\}$. Instances of class 0 are generated from a Gaussian distribution with mean $\mu = (0, 0)$ and covariance matrix $\sigma = (20, 0.2)$, Instances of class 1 stem from a Gaussian distribution with mean $\mu = (1, 1)$ and covariance matrix $\sigma = (0.5, 0.5)$. The training data and test data sets with 200 instances each are stored in the files `gauss_train.dat` and `gauss_test.dat`, with one instance $x_1 \ x_2 \ c$ per line. Load the data into Matlab and visualize the data, in both cases the first 100 instances belong to class 0, the second 100 instances to class 1.

```
>> gauss_train=load('gauss_train.dat');  
>> plot(gauss_train(1:100,1),gauss_train(1:100,2),'ro',...  
        gauss_train(101:200,1),gauss_train(101:200,2),'bx');
```

```
>> axis([-4 4 -4 4]);
```

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 probability 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.

$$h_{MAP} = \operatorname{argmax}_{h \in H} P(h|D) \quad (1)$$

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

$$h_{ML} = \operatorname{argmax}_{h \in H} P(D|h) \quad (2)$$

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 $P(D)$.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \quad (3)$$

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. In this exercise, 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 $\vec{x} = \{x_1, \dots, x_N\}$. 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.

$$p(c_i) = \alpha_i \quad (4)$$

The likelihood density functions for a model (hypothesis) with diagonal covariance matrix ($\sigma_{ij} = 0, \sigma_{ii} = \sigma_i$) are given by

$$p(\vec{x}|c_i) = (2\pi\sigma_i^2)^{-N/2} \exp - \frac{\sum_{n=1}^N (x_n - \mu_{in})^2}{2\sigma_i^2} \quad (5)$$

We apply Bayes rule to compute the posterior probabilities from the likelihoods and the prior probabilities:

$$p(c_i|\vec{x}) = p(\vec{x}|c_i) \frac{p(c_i)}{p(\vec{x})} = \alpha_i * (2\pi\sigma_i^2)^{-N/2} \exp - \frac{\sum_{n=1}^N (x_n - \mu_{in})^2}{2\sigma_i^2} / p(\vec{x}) \quad (6)$$

This is the posterior for a single feature vector \vec{x} . Let us now assume that we have several independent feature vectors $D = \{\vec{x}^1, \dots, \vec{x}^{M_i}\}$ 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 simply the product of the individual posteriors

$$P(c_i|D) = \prod_{\{m|c_m=c_i\}}^{M_i} p(c_i|\vec{x}^m) \quad (7)$$

The maximum a posteriori hypothesis (MAP) $\vec{\mu}_i^*, \sigma_i^*$ that best explains the data is computed by maximizing equation 7 with respect to $\vec{\mu}_i, \sigma_i$.

$$\{\vec{\mu}_i^*, \sigma_i^*\} = \operatorname{argmax}_{\vec{\mu}_i, \sigma_i} p(c_i|D) \quad (8)$$

We now apply a common transformation, namely rather than maximizing the above expression we maximize the less complicated logarithm of equation 7.

$$\log p(c_i|D) = -NM_i \log 2\pi\sigma_i + \sum_{\{m|c_m=c_i\}}^{M_i} \frac{-\sum_{n=1}^N (x_n^m - \mu_{in})^2}{2\sigma_i^2} + \log \alpha_i - \log p(\vec{x}^m) \quad (9)$$

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 9 disappears.

$$\frac{\partial \log p(c_i|D)}{\partial \mu_{in}} = \sum_{\{m|c_m=c_i\}}^{M_i} - \frac{(x_n^m - \mu_{in})}{\sigma_i^2} = 0 \quad (10)$$

$$\frac{\partial \log p(c_i|D)}{\partial \sigma_i} = -\frac{NM_i}{\sigma_i} + \sum_{\{m|c_m=c_i\}}^{M_i} \sum_{n=1}^N \frac{(x_n^m - \mu_{in})^2}{\sigma_i^3} = 0 \quad (11)$$

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

$$\mu_{in}^* = \frac{\sum_{\{m|c_m=c_i\}}^{M_i} x_n^m}{M_i} = E[x_n] \quad (12)$$

$$\sigma_i^{*2} = \frac{\sum_{\{m|c_m=c_i\}}^{M_i} \sum_{n=1}^N (x_n^m - \mu_{in}^*)^2}{NM_i} = \frac{E[(\vec{x} - \vec{\mu}_i)^2]}{N} \quad (13)$$

Assignment 1: Write a function `bayes(data)` that computes the maximum posterior (MAP) parameters μ_{in}^* and σ_i^* 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, \dots, 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` ($C \times N$ -matrix) and `sigma` ($C \times 1$ -vector). C denotes the number of classes and N is the number of features.

Apply your function to the artificial gauss data set `bayes(gauss_train)` and compute

$$\mu_{01}^* = \dots$$

$$\mu_{02}^* = \dots$$

$$\sigma_0^* = \dots$$

$$\mu_{11}^* = \dots$$

$$\mu_{12}^* = \dots$$

$$\sigma_1^* = \dots$$

Notice, that our univariate model in which both features (x_1, x_2) have the same variance σ (hyper-spheres), has fewer degrees of freedom than the original multivariate data distribution with covariance matrix σ_{ij} (hyper-ellipsoids). However, the optimal means μ_{ij} should match those of the underlying known model.

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 6.

$$c^* = \operatorname{argmax}_{c_i} p(c_i|\vec{x}) \quad (14)$$

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

$$\log(p(c_i|\vec{x})) = \log \alpha_i - N/2 \log(2\pi) - N \log \sigma_i - \frac{\sum_{n=1}^N (x_n - \mu_{in}^*)^2}{2\sigma_i^2} - \log p(\vec{x}) \quad (15)$$

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

$$g_i(\vec{x}) = \log \alpha_i - N \log \sigma_i - \frac{\sum_{n=1}^N (x_n - \mu_{in})^2}{2\sigma_i^2} \quad (16)$$

Write a function `discriminant(data,mu,sigma,p)` that computes the discriminant functions g_i . The parameter `data` is the $M \times N$ -matrix of M feature vectors \vec{x}^m , `mu` is the $C \times N$ -matrix of means, `sigma` is an $C \times 1$ -vector of standard deviations, and `p` is an $C \times 1$ -vector of prior class probabilities α_i . C again denotes the number of different classes. Each class c_i is associated with one Gaussian described by the pair $\vec{\mu}_i, \sigma_i$ and its prior class probability $p(c_i)$. The signature of `discriminant(data,mu,sigma,p)` in Matlab looks like

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

where the return value `g` is a $M \times C$ -matrix.

Assignment 2: Train the Bayes classifier with the dataset `gauss_train`, and compute the classification error for training and test data `gauss_train`. The prior class probabilities are computed by using the Matlab function `prior(data)`. Notice, that in this case the prior probabilities are both 0.5 as the dataset contains 100 instances of classes 0 and 1 each. We use the Matlab function `max` to compute the maximum discriminant value and the index `g_class` of the optimal class, which we compare with the known classification `gauss_train(:,3)` respectively `gauss_test(:,3)`.

```
>> [M,N]=size(gauss_train);
>> [mu,sigma]=bayes(gauss_train);
>> p=prior(gauss_train);
>> g_train=discriminant(gauss_train(:,1:2),mu,sigma,p);
>> gauss_test=load('gauss_test.dat');
>> [dummy g_class]=max(g_train,[],2);
>> error_train=1.0-length(find(g_class==gauss_train(:,3)+1))/M
>> g_test=discriminant(gauss_test(:,1:2),mu,sigma,p);
>> [dummy g_class]=max(g_test,[],2);
>> error_test=1.0-length(find(g_class==gauss_test(:,3)+1))/M
```

train error gauss = ...

test error gauss = ...

Plot the classification of the test data predicted by the Bayes classifier and mark the incorrectly classified instances in red with the following Matlab code.

```
>> class_0=find(g_class==1);
>> class_1=find(g_class==2);
>> false_class_0=class_0(find(class_0>M/2));
>> false_class_1=class_1(find(class_1<=M/2));
>> correct_class_0=class_0(find(class_0<=M/2));
>> correct_class_1=class_1(find(class_1>M/2));
>> plot(gauss_test(correct_class_0,1),gauss_test(correct_class_0,2),'bo',...
        gauss_test(correct_class_1,1),gauss_test(correct_class_1,2),'bx',...
        gauss_test(false_class_0,1),gauss_test(false_class_0,2),'rx',...
        gauss_test(false_class_1,1),gauss_test(false_class_1,2),'ro');
>> axis([-4 4 -4 4]);
```

Overlay the decision boundary $X = \{\vec{x} : g_0(\vec{x}) = g_1(\vec{x})\}$ that separates both classes on the previous test data plot using the following Matlab code.

```
>> x1=-4:0.1:4;
>> x2=-4:0.1:4;
>> [z1,z2]=meshgrid(x1,x2);
>> z1=reshape(z1,81^2,1);
>> z2=reshape(z2,81^2,1);
>> g = discriminant([z1 z2],mu,sigma,p);
>> gg=g(:,1)-g(:,2);
>> gg=reshape(gg,81,81);
>> hold on;
>> [c,h]=contour(x1,x2,gg,[0.0 0.0]);
>> set(h,'LineWidth',3);
>> axis([-4 4 -4 4]);
>> hold off;
```

What do you think is the functional form of the decision boundary?

3 Boosting

Boosting aggregates multiple hypotheses generated by the same learning algorithm invoked over different distributions of training data into a single composite classifier [4]. 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.

The training and test data sets for this assignment with 400 instances each are stored in the files `gauss_boost_train.dat` and `gauss_boost_test.dat`, with one instance x_1x_2c per line. Load the data into Matlab and visualize the data, in both cases the first 200 instances belong to class 0, the second 200 instances to class 1.

```
>> gauss_boost_train=load('gauss_boost_train.dat');
>> gauss_boost_test=load('gauss_boost_test.dat');
>> plot(gauss_boost_train(1:200,1),gauss_boost_train(1:200,2),'ro',...
        gauss_boost_train(201:400,1),gauss_boost_train(201:400,2),'bx');
>> axis([-2 3 -2 3]);
```

Apply the Bayes learning algorithm to the training data, obtain the maximum posterior parameters μ and σ of the Bayes classifier and compute the training and test error.

```
[mu,sigma]=bayes(gauss_boost_train);
[M,N]=size(gauss_boost_train);
p=prior(gauss_boost_train);
g_train=discriminant(gauss_boost_train(:,1:2),mu,sigma,p);
[V g_class]=max(g_train,[],2);
error_train=1.0-length(find(g_class==gauss_boost_train(:,3)+1))/M
g_test=discriminant(gauss_boost_test(:,1:2),mu,sigma,p);
[V g_class]=max(g_test,[],2);
error_test=1.0-length(find(g_class==gauss_boost_test(:,3)+1))/M

train error boost gauss = ...
test error boost gauss = ...
```

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 13 for the MAP parameter with weighted instances become:

$$\begin{aligned}\mu_{in}^* &= \frac{\sum_{\{m|c_m=c_i\}}^{M_i} \omega^m x_n^m}{\sum_{\{m|c_m=c_i\}}^{M_i} \omega^m} = E_\omega[x_n] \\ \sigma_i^{*2} &= \frac{\sum_{\{m|c_m=c_i\}}^{M_i} \sum_{n=1}^N \omega^m (x_n^m - \mu_{in})^2}{N \sum_{\{m|c_m=c_i\}}^{M_i} \omega^m} = \frac{E_\omega[(\vec{x} - \vec{\mu}_i)^2]}{N}\end{aligned}\quad (17)$$

Assignment 3: Extend the old `bayes` function to `bayes_weight(data,w)` that handles weighted instances. Again `data` is $M \times N$ -matrix of feature vectors and `w` is a $M \times 1$ -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_i^* for a dataset D according to the equations in 17. Assume the usual data format for the parameter `data`, namely that the first N columns contain the features x_1, \dots, 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`. In order to calculate the priors `p` for weighted instances use the function `prior(data,w)`. It computes the prior class probabilities considering weighted instances.

```
>> p = prior(data,w)
```

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 $\mu_{tin}^*, \sigma_{ti}^*$.

$$\epsilon_t = 1 - \sum_{\{m|h_t(x^m)=c^m\}} \omega_t^m \quad (18)$$

or in more compact notation

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

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

$$\alpha_t = 1/2 \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right) \quad (20)$$

- update weights according to

$$\omega_{t+1}^m = \frac{\omega_t^m}{Z_t} * \begin{cases} e^{-\alpha_t} & \text{if } h_t(x^m) = c^m \\ e^{\alpha_t} & \text{if } h_t(x^m) \neq c^m \end{cases} \quad (21)$$

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 $h_t = \{\mu_{tin}^*, \sigma_{ti}^*, p_t(c_i)\}$. 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 $v(c_i)$

$$v(c_i, x) = \sum_{\{t|h_t(x)=c_i\}} \alpha_t \quad (22)$$

$$H(x) = c_{max} = \operatorname{argmax}_{c_i} v(c_i, x) \quad (23)$$

or in more compact notation.

$$H(x) = c_{max} = \operatorname{argmax}_{c_i} \sum_{t=1}^T \alpha_t \delta(h_t(x), c_i) \quad (24)$$

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, priors. The return parameter α_t holds the classifier vote weights α_t and `classes` the set of unique classes. In particular `mu` is $T \times C \times N$ -array, `sigma` and prior are $T \times C$ -matrices, `alpha` is a $T \times 1$ -vector and `classes` is a $C \times 1$ -vector. Adaboost uses `bayes_weight(data,w)` to compute the MAP parameters `mu` and `sigma` and `prior(data,w)` to calculate the class priors. Adaboost computes the `alpha` using the error over the current distribution of instances according to equations 19 and 20.

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 24. The signature in Matlab looks like

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

where `c` is a $M \times 1$ -vector that contains the class predicted for the $M \times N$ -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 based on equation 24. Compute the classification accuracy for training and test of the boosted classifier and compare it with those of the basic classifier (see assignment 2).

```
>> [M,N]=size(gauss_boost_test);
>> T=6;
>> [mu sigma p alpha classes]=adaboost(gauss_boost_train,T);
>> class=adaboost_discriminant(gauss_boost_train(:,1:N-1),...
                                mu,sigma,p,alpha,classes,T);
>> error_train=1.0-length(find(class==gauss_boost_train(:,3)))/M
>> class=adaboost_discriminant(gauss_boost_test(:,1:N-1),...
                                mu,sigma,p,alpha,classes,T);
>> error_test=1.0-length(find(class==gauss_boost_test(:,3)))/M
```

boosted train error boost gauss = ...
boosted test error boost gauss = ...

Plot the classification of the test data set and the decision boundary of the boosted classifier using the Matlab code.

```
>> class_0=find(class==1);
>> class_1=find(class==2);
>> false_class_0=class_0(find(class_0>M/2));
>> false_class_1=class_1(find(class_1<=M/2));
>> correct_class_0=class_0(find(class_0<=M/2));
>> correct_class_1=class_1(find(class_1>M/2));
>> plot(gauss_boost_test(correct_class_0,1),...
        gauss_boost_test(correct_class_0,2),'bo',...
        gauss_boost_test(correct_class_1,1),...
        gauss_boost_test(correct_class_1,2),'bx',...
        gauss_boost_test(false_class_0,1),...
        gauss_boost_test(false_class_0,2),'rx',...
        gauss_boost_test(false_class_1,1),...
        gauss_boost_test(false_class_1,2),'ro');
>> axis([-2 3 -2 3]);
```

Overlay the decision boundary on the test data plot using the following Matlab code.

```
>> hold on;
>> x1=-2:0.2:3;
>> x2=-2:0.2:3;
>> [z1,z2]=meshgrid(x1,x2);
>> z1=reshape(z1,26^2,1);
>> z2=reshape(z2,26^2,1);
>> g = adaboost_discriminant([z1 z2],mu,sigma,p,alpha,classes,T);
>> gg=reshape(g,26,26);
>> hold on;
>> [c,h]=contour(x1,x2,gg,[0.5 0.5]);
>> set(h,'LineWidth',3);
>> axis([-2 3 -2 3]);
>> hold off;
```

Compare the decision boundary of the boosted classifier with that of the basic Bayesian classifier.

References

- [1] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification 2nd ed.* Wiley, 2001.
- [2] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [3] J.R. Quinlan. Bagging, boosting, and c4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 725–730. 1996.
- [4] R.E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. 1999.