Exercises
Machine Learning
AS 2012

Machine Learning Laboratory
Dept. of Computer Science, ETH Zürich
Prof. Dr. Joachim M. Buhmann
Web http://ml2.inf.ethz.ch/courses/ml/

Email questions to: Alexey Gronskiy
alexey.gronskiy@inf.ethz.ch

Series 3, Oct 24th, 2012 (Bayesian Learning and Perceptron)

Problem 1 (Bayesian Learning):

In the previous exercise series, we applied the frequentist maximum-likelihood approach to estimate the parameter(s) $\boldsymbol{\theta}$ of the model. Here, we consider the approach of Bayesian inference instead.

Bayes' rule relates the prior distribution $p(\boldsymbol{\theta})$ of the parameter and the likelihood $p(\mathcal{X}|\boldsymbol{\theta})$ of the dataset $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ to the posterior distribution:

$$p\left(\boldsymbol{\theta}|\mathcal{X}\right) = \frac{p\left(\mathcal{X}|\boldsymbol{\theta}\right) p\left(\boldsymbol{\theta}\right)}{p\left(\mathcal{X}\right)}.$$

In this exercise, we assume that the samples X_i are (pairwise) independent and exponentially distributed:

$$X_i \sim \mathrm{Exponential}(\lambda),$$

for all $i = 1, \ldots, n$. The exponential distribution is controlled by one *rate parameter* $\lambda > 0$, and its density is

$$p\left(x; \lambda\right) = \lambda e^{-\lambda x}$$

for $x \geq 0$. For $x < 0$, we let $p\left(x; \lambda\right) = 0$.

1. Draw the graph of $p\left(x; \lambda\right)$ for $\lambda = 1$ in the interval $x \in [0, 4]$. A qualitative sketch suffices, but give the precise function value for at least two points in the domain of $p\left(x; \lambda\right)$.
2. What is the visual representation of the likelihood of individual data points? Draw it into the graph above for the samples in a toy dataset $\mathcal{X} = \{1, 2, 4\}$ and $\lambda = 1$. How is the likehood of this toy dataset related to that of the individual data points?
3. Would a higher rate (e.g. $\lambda = 2$) increase or decrease the likelihood for the toy data set?

We introduce a prior distribution $p\left(\lambda\right)$ over the parameter of interest. Having determined the functional form of the prior and likelihood, we want to compute the posterior. Doing it analytically can be hard in general, but it is easy if the prior and likelihood form a *conjugate pair*. Then the posterior will have the same functional form as the prior, only the parameters differ.

We choose the *gamma* distribution for the prior, because it forms a conjugate pair with the exponential likelihood. Here we define $\mathrm{Gamma}(\alpha, \beta)$ in slightly different form than in the last exercise sheet:

$$p\left(\lambda; \alpha, \beta\right) = \lambda^{\alpha-1} \frac{\beta^{\alpha} e^{-\beta\lambda}}{\Gamma(\alpha)}$$

for $\lambda \geq 0$ and $\alpha, \beta > 0$. Both definitions are common, but this one will simplify the following calculations.

Show that if the prior is gamma distributed

$$p\left(\lambda\right) = \mathrm{Gamma}(\alpha_0, \beta_0)$$

with *hyperparameters* (α_0, β_0), and the data likelihood $p\left(\mathcal{X}|\lambda\right)$ is a product of n exponential densities (as defined above), then the posterior is again gamma distributed

$$p\left(\lambda|\mathcal{X}\right) = \mathrm{Gamma}(\alpha_n, \beta_n),$$

but with different parameters (α_n, β_n). To compute the parameters α_n and β_n:

1. Ignore multiplicative constants and normalization terms, such as the evidence term in Bayes' formula (which ensures that the posterior integrates to one).

2. Show that the posterior is proportional to a gamma distribution.

3. Infer the parameters by comparing your result for the posterior to the definition of the gamma distribution.

We turn to online estimation of the posterior, where the data is processed sequentially. For this purpose, we define $\mathcal{X}_n = \mathcal{X}_{n-1} \cup \{x_n\}$. The update is defined as

$$p(\lambda|\mathcal{X}_n) = \frac{p(x_n|\lambda)p(\lambda|\mathcal{X}_{n-1})}{\int p(x_n|\lambda)p(\lambda|\mathcal{X}_{n-1})d\lambda}.$$

We know that after $(n-1)$ samples have been observed, the posterior $p(\lambda|\mathcal{X}_{n-1})$ is gamma distributed with parameters $(\alpha_{n-1}, \beta_{n-1})$. Give a definition of the parameters (α_n, β_n) of the posterior $p(\lambda|\mathcal{X}_n)$ in terms of $(\alpha_{n-1}, \beta_{n-1})$.

To visualize the gradual change of shape of the posterior $p(\lambda|\mathcal{X}_n)$ with increasing n:

1. Generate $n = 256$ exponentially distributed samples with the true parameter $\lambda = 1$. You can use the function `exprnd` from the statistics toolbox.

2. Use the values $\alpha_0 = 2, \beta_0 = 0.2$ for the hyperparameters of the prior.

3. Visualize the updated posterior distribution after $n = \{4, 8, 16, 256\}$, in the range `x = 0:0.01:4`. Plot all curves in the same figure (using `hold all`), and use the `legend` command to label each curve. Unfortunately, using Matlab's `gamma` runs into nummerical problems. One trick is to first compute the log-likelihood, then take the exponential of this, i.e., compute $\exp(\log(p(\lambda; \alpha, \beta)))$. You can make use of Matlab's `gammaln` function.

4. What can you observe about the posterior distribution as n increases?

Finally, show that for large n (i.e. many data points), the maximum of the posterior

$$\arg\max_{\lambda} p\left(\lambda|\mathcal{X}_n\right)$$

agrees with the ML estimator

$$\hat{\lambda}_n = \frac{1}{\frac{1}{n}\sum_{i=1}^{n} x_i}$$

for the rate parameter of the exponential distribution:

1. Start with the fact that the gamma distribution attains its maximum value at $\frac{\alpha-1}{\beta}$.

2. Plug in the values α_n and β_n that you have obtained for the posterior distribution.

3. Take the limit $n \to \infty$ and compare to $\hat{\lambda}$.

Problem 2 (K-fold Cross Validation and Bootstrap):

In the lecture, we have looked at some model assessment methods. In this exercise, we will implement K-fold cross validation and bootstrap. These two methods will come in useful for future exercises, so it is good to have them handy.

We first present a simple classification problem. Assume that we are given a data source consisting of C classes, each of which is normally distributed. We train a simple classifier in the following way:

- Each class c is represented by samples (training data) $\mathcal{X}_c \sim p(x|Y = c) := \mathcal{N}(\Theta_c)$.
- We fit a Gaussian distribution into each class using maximum likelihood estimation, and obtain the model parameters $\hat{\Theta}_c$ for $c = 1, ..., C$.
- To classify a test data value, we assume uniform priors, i.e., $p(Y = 1) = ... = p(Y = C) = 1/C$, and can use Bayes theorem to compute the posterior: $p(y|x) \propto p(x|y)p(y)$. For a data value x, we assign it to the class corresponding to the largest posterior.

Please download the functions provided: `train.m` and `classify.m`.

In the function `[mdl] = train(X,Y)`, `X` is the input data, where each column is a vector corresponding to a single data point. The class of each datapoint is stored in the corresponding column in the row vector `Y`. The output `mdl` is a structure array, where `mdl(i).mu` and `mdl(i).Sigma` are the estimated statistics for class `mdl(i).class` .

The classification function is given by `[Y] = classify(X,mdl)`. Here, `Y` is the predicted class label (i.e., `mdl(i).class` for which the model density value is largest).

Here's what we ask you to do:

1. Implement the following functions:
   - `[err,var] = gauss_cv(X,Y,K)`,
     which performs K-fold cross validation for the model described above. To keep things simple[1], call the methods `train` and `classify` directly from your code, rather than handing them over as a function handle. The return value `err` is the estimated error, with its associated variance denoted by `var` as discussed in the class.
   - `[err,var] = gauss_bootstrap(X,Y,B, type)`,
     which now performs B rounds of bootstrap in place of cross validation. The parameter `type` is an integer that indicates the type of bootstrap error computed (Lecture slides page 121).
     - `type=1`: Original bootstrap estimation (too optimistic).
     - `type=2`: e_0 bootstrap estimator.
     - `type=3`: $.632$ bootstrap estimator.
2. Now assume that $X_1 \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $X_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$, where

$$\boldsymbol{\mu}_1 = (1,1)^t,\ \boldsymbol{\Sigma}_1 = \begin{pmatrix} 8 & 1 \\ 1 & 1 \end{pmatrix},\ \boldsymbol{\mu}_2 = (-4,0)^t,\ \boldsymbol{\Sigma}_2 = \begin{pmatrix} 3 & 1 \\ 1 & 1 \end{pmatrix}.$$

   Draw $n = 10$ samples from each class using your `gsample` method. On the sampled dataset, apply both cross validation ($K = 2$ and $K = 10$) and bootstrap methods ($B = 200$) to estimate the error and variance. Report the error and variance obtained for each method.
3. Repeat the previous task, this time, use $n = 100$. Report the error and variance obtained for each model selection method. Comment on your results obtained on both tasks.

Please submit only your answers to part 2 and 3 of the question.

Problem 3 (Bootstrap):

In the lecture we have learnt about the bootstrap technique. In bootstrap, B datasets are sampled which are used to estimate the bias and the variance of an estimator. In non-parametric bootstrap, we sample N times from the original dataset of N elements *with replacement*. Here, let us study some of the properties of this sampling with replacement.

1. For $N \to \infty$, show that the probability that a sample is not observed in a single bootstrap draw is $e^{-1} \approx 0.368$..
2. For a given N, what is the expected number of distinct samples in a single bootstrap draw? Explain your answer.
3. Verify your answer by plotting a histogram of the #(distinct samples) over $B = 10000$ bootstrap draws. Use $N = 500$ here. Can you say something about the variance of the set?
4. We will now look at how many possible different bootstrap sets that we can potentially obtain. For a given value N, how many distinct sets can we possibly generate? Explain your solution. Compute the number of such distinct sets for $N = 10$. We see that the number of distinct sets are large.

[1] Of course, if you are an enthusiatic, do feel free to code a general version that works with function handles.

5. We can use bootstrap to give us an estimate on the variance of the mean statistic. Prove that in the limit of $B \to \infty$, the bootstrap estimate of the variance of the mean is the same as the standard estimate of the variance of the mean.

Problem 4 (Perceptron):

Our perceptron implementation will use the homogeneous coordinate representation of vectors, i.e. vectors $\mathbf{x} \in \mathbb{R}^d$ are represented by $\mathbf{y} = (x_1, \ldots, x_d, 1)^t \in \mathbb{R}^{d+1}$. This representation turns affine hyperplanes $\{\mathbf{x} \in \mathbb{R}^d | \mathbf{w}^t\mathbf{x} + w_0 = 0\}$ with normal vector $\mathbf{w} = (w_1, \ldots, w_d)^t$ into hyperplanes through the origin of the form $\{\mathbf{y} \in \mathbb{R}^{d+1} | (w_1, \ldots, w_d, w_0)^t \mathbf{y} = 0\}$.

To work with a perceptron, we need a two-class set of linearly separable data. As we have always done so far, we will just generate the data by ourselves. A sample set of n data values in d-dimensional space will be represented as a $n \times (d+1)$-matrix `S`, so each row is the (homogeneous coordinate) representation of a single data point. This matrix will contain the data of both classes, so in addition, we need class labels. Class labels will be represented as a vector `c` of length n, with entries in $\{-1, +1\}$.

Here is what we want you to do:

1. Write a matlab function `[S,c] = fakedata(a, n_samples)`, which takes a $(d+1)$-dimensional hyperplane normal vector `a` as its argument and returns a randomly generated, linearly separable two-class data set `S` with class labels `c`. Which method (and which type of distribution) you use to generate the data is up to you. Remember to generate only d random components for each vector, as the last component is 1. The matlab function `null` might be useful: Calling `null` for a normal vector returns a set of vectors spanning its nullspace, i.e. the corresponding hyperplane.

2. To evaluate a perceptron solution (a hyperplane classifier trained by a perceptron algorithm), write a function `c = classify(S,w)`. `S` is a sample data set and `w` the perceptron weight vector (which will be returned by the perceptron training algorithm). `c` is a class label vector as described above.

3. Implement two versions of the perceptron training algorithm (cf. slide 110, or chapter 5 of the Duda book):

   - The batch version with variable step size $\eta(k) = \frac{1}{k}$, where k is the number of the current iteration.
   - The fixed-increment single sample version.

   Both functions should be of the form

   `[w,W_history] = perceptrain(S,c),`

   where w is the normal vector of the hyperplane. `W_history` is a matrix containing the history of the normal vector throughout the training run, i.e. a (number of iterations) $\times (d+1)$ matrix, the rows of which are the interim results for `W`.

   A recommended sanity check is to train your `perceptrain` function on a `fakedata` sample, and to make sure that the classifiers returned by your perceptron algorithms correctly classify their own training data.

4. Generate a 3D random vector `a`, do `[S,c]=fakedata(a,100)`, and train both your perceptrons on this set. Rerun `fakedata` to produce a test set, and check whether it is classified correctly.

5. Convert the data and the vectors into their corresponding 2D representation to make it more suitable for plotting.

And here is what we want you to hand in as a solution:

1. A hardcopy of your source code (at least the essential parts).

2. For both implementations, hand us one plot showing the test data set and the classifier hyperplane, and one showing the training data and the trajectory of the algorithm by visualizing `W_history`.