1 Perceptron

Implement functions for simulating perceptron:

- `p = perc_create(n)` creates and initializes randomly a perceptron with `n` inputs. The perceptron will be represented by an extended weight vector — a row vector of length `n+1`, where the last component is the bias of the perceptron with the opposite sign.
- `c = perc_recall(p,x)` returns the output of the perceptron `p` for an input (column) vector `x`.
- `pn=perc_update(p,x,c,lam)` performs a single step of the learning algorithm for the perceptron `p` on a set of input vectors, where

  `x` is a matrix with some k input vectors as columns,

  `c` is a row vector of length k containing the desired outputs for the corresponding input vectors,

  `lam` is a learning rate.

  The function `perc_update` performs the perceptron learning algorithm for each column `x(i)` with the desired output `c(i)`, for `i` $= 1, \ldots,$ `k`. Instead of adding or subtracting the vector `x(i)`, the function adds or subtracts `lam*x(i)`.
- `c = perc_err(p,x,c)` returns the number of input vectors – columns of the matrix `x` – misclassified by the perceptron `p` when the desired outputs are in (row) vector `c`.
- `pn=perc_learn(p,x,c,lam,maxit)` makes at most `maxit` epochs of learning the perceptron `p`, where

  `x` is a matrix of input vectors – columns of the matrix,

  `c` is a row vector of desired outputs of the perceptron,

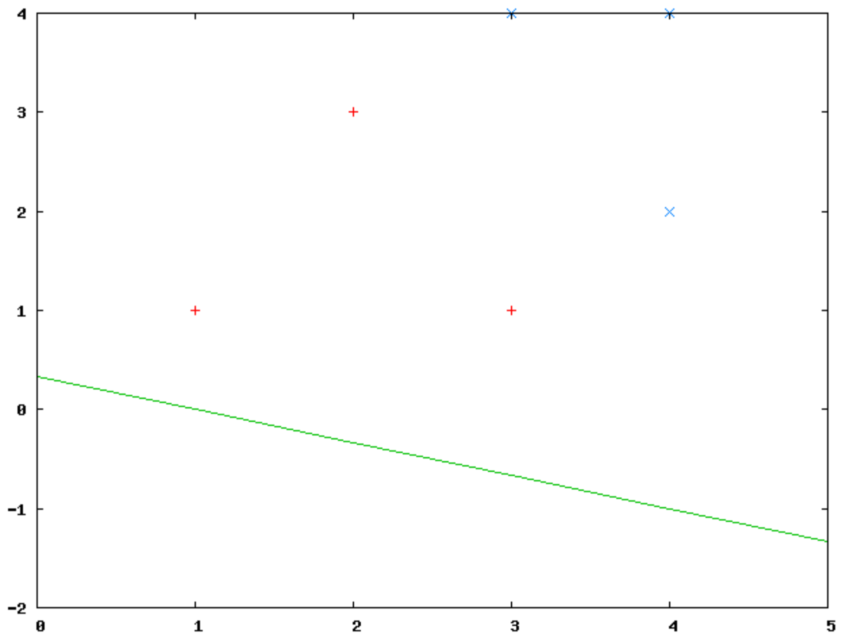
  `lam` is a learning parameter.

  The learning stops when the perceptron has a zero error on `x` or the number of epochs reaches `maxit`. One epoch consists of one step of perceptron learning performed on each input vector from `x`.

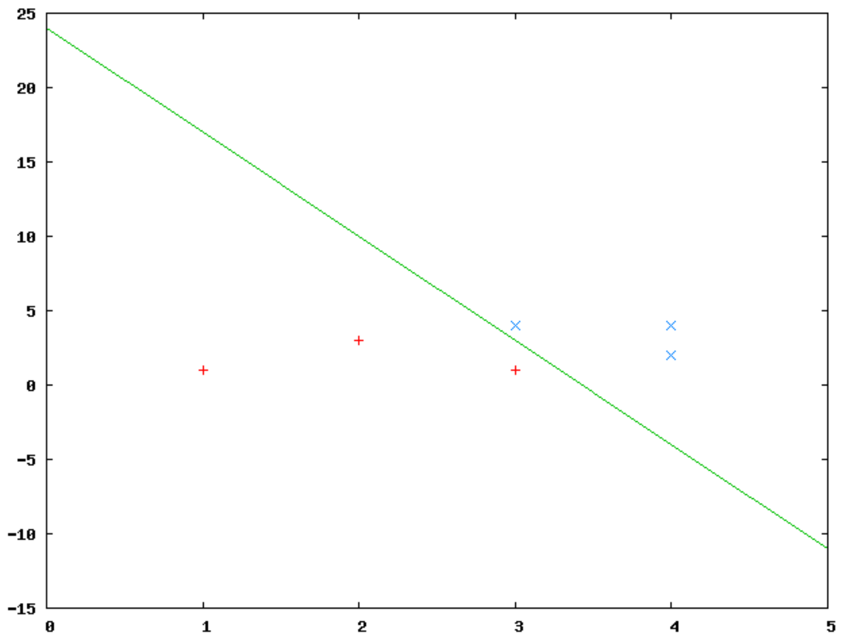
Test your implementation on the following example:

```
p=[1 3 -1];                    %manually initialized perceptron
xpos=[1 2 3; 1 3 1];           %training positive inputs, desired output 1
xneg=[3 4 4; 4 2 4];           %training negative inputs, desired output 0
x=[xpos xneg];
perm=[1 4 6 2 3 5];
x=x(:,perm)                    %permute the training smaples
c=[ones(1,3) zeros(1,3)];
c=c(perm)                      %permute also the desired outputs
xr=0:5;                        %plot the samples and the separating
                               %hyperplane of the perceptron
plot(xpos(1,:),xpos(2,:),'+r',xneg(1,:),xneg(2,:),'xb',xr,-(p(1)*xr+p(3))
  /p(2),'g')
pn=perc_learn(p,x,c,0.5,100)  %learn the perceptron and depict the result
plot(xpos(1,:),xpos(2,:),'+r',xneg(1,:),xneg(2,:),'xb',xr,-(pn(1)*xr+pn(3))
  /pn(2),'g')
```

The first `plot`-command illustrates the initial state of the perceptron.



The second `plot` draws the state of the learned perceptron.



What is the minimal number of epochs necessary to learn the perceptron to classify all training samples correctly?

2 Testing Machine Learning Algorithms

Task: Let X be a set, e.g., the set of all vectors of length d with entries from the interval $< 0, 1 >$. Let $f : X \rightarrow \{0, 1\}$ be a target function. A goal of a learning algorithm is to identify a function $h : X \rightarrow \{0, 1\}$, called hypothesis, from some class of functions $\mathcal{H}$ such that the function h is a good approximation of the target function f. The only information the algorithm can use is a sample $S \subset X$ called training set together with the correct value $f(x)$ for all $x \in S$. The sample S is a set of n elements from X randomly selected according to some probabilistic distribution D.

When the learning algorithm selects some function h, we can easily measure how well h approximates f on the training set by counting the number of elements from S for which both functions f and h have the same value. Then the error of the hypothesis h on the training set is

$$\mathsf{Error}_S(h) = \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x)) \ ,$$

where n is he size of S and δ is the classification error function

$$\delta(\alpha, \beta) = \begin{cases} 1 & \text{for } \alpha \neq \beta \\ 0 & \text{for } \alpha = \beta \end{cases}$$

Hence,

$$\mathsf{Error}_S(h) = \frac{r}{n} \ ,$$

where r is the number of elements from S, which were incorrectly classified by h.

Problems:

1. How to estimate accuracy of h on *any* sample taken from X according to the same probability distribution D?
2. How good is this estimator – what is the probable error of this estimator?

We want to know the error rate of the hypothesis h on the whole set X, i.e. the probability that an element $x \in X$ selected according to the probability distribution D is classified incorrectly:

$$\mathsf{Error}_D = Pr_{x \in D}[f(x) \neq h(x)] \ .$$

Such error is called *true error*. It has a binomial distribution. An estimator of the error can be computed as the number of incorrectly classified samples divided by the number of samples. However, this estimator must be computed on a sample which is independent from the training set S! Hence:

> **an estimator of the error of a learning algorithm must be computed on a test set T ($\subset X$) independent (disjunctive) from S.**

The maximum likelihood estimate of the true error rate of the learning algorithm can be computed from the number of errors r_T on a test set T of size $n_T = |T|$ as:

$$\mathsf{Error}_T(h) = \frac{r_T}{n_T} \quad .$$

Then the expected number of errors on n_T samples is r_T. Variance of the number of errors is according to the binomial distribution:

$$\mathsf{Var'}_T = n_T \cdot \frac{r_T}{n_T} \cdot \left(1 - \frac{r_T}{n_T}\right) = n_t \cdot \mathsf{Error}_T(h) \cdot (1 - \mathsf{Error}_T(h)) \quad .$$

Also the standard deviation of the expected number of errors is:

$$\sigma'_T = \sqrt{n_t \cdot \mathsf{Error}_T(h) \cdot (1 - \mathsf{Error}_T(h))} \quad .$$

Hence he standard deviation of the estimator of error rate of hypothesis h on the test set T is

$$\sigma_T = \frac{\sigma'_T}{n_T} = \sqrt{\frac{\mathsf{Error}_T(h) \cdot (1 - \mathsf{Error}_T(h))}{n_T}} \quad .$$

Using Matlab, it is easy to compute a confidence interval – the interval to which belongs the actual error rate of the hypothesis h on the whole set X with a prescribed probability. With the confidence α the hypothesis h makes at most `k=binoinv(`α`,`n_T`,`r_T/n_T`)` errors on a test set of size n_T. Therefore, the error of the hypothesis h on the whole X belongs to the interval $< 0, \mathtt{k}/n_T >$ with the confidence α.

For a sufficiently large n_T (e.g., $n_T \geq 30$, or $n_T \cdot \mathsf{Error}_T(h) \cdot (1 - \mathsf{Error}_T(h)) \geq 5$, respectively) the binomial distribution can be approximated by the normal distribution with mean $\mathsf{Error}_T(h)$ and standard deviation $\sigma_T(h)$. Consequently, the two-sided P%-confidence interval is given by the following formula

$$\mathsf{Error}_T(h) \pm z_P \cdot \sigma_T \quad ,$$

where the values of z_P are called quantiles and can be found in statistical tables. In Matlab it is possible to compute z_P directly:

```
>> a=[80,90,95,98,99]
a =
    80      90      95      98      99
>> b=(100+a)/2/100
b =
    0.9000    0.9500    0.9750    0.9900    0.9950
>> norminv(b,0,1)
ans =
    1.2816    1.6449    1.9600    2.3263    2.5758
```

Hence, e.g., $z_{95} = 1.96$ and $z_{99} = 2.5758$.

In a single graph, plot the binomial probability distribution for $n = 40$ samples with the probability of success $p = 0.1$ (as this is a discrete probability distribution, your task is to plot the distribution function for arguments

$1, \dots, n$) and the normal probability distribution with the same mean and standard deviation (in this case, it is a continuous probability distribution, hence you should plot its continuous density function).

A general guidance for deriving a confidence interval
A $N\%$ confidence interval for a parameter p is an interval that is expected with probability $N\%$ to contain p.
Method:

1. Identify the parameter p, which should be estimated – e.g., the error of a hypothesis $\mathsf{Error}_D(h)$.
2. Derive the estimate Y – in our case, $\mathsf{Error}_T(h)$. If possible, use an unbiased estimate with minimal variance.
3. Determine the probability distribution D_Y for Y, including its mean and standard deviation.
4. Establish $N\%$ confidence interval – i.e. find the limits L and U such that $N\%$ of samples selected according to the probability distribution D_Y are between L and U.

3 Comparing Learning Algorithms

Learning algorithms can be compared using methods for comparing statistical hypothesis, but better results can be obtained from pair tests or k-fold cross validation.

3.1 Difference in Error of Two Hypotheses

A hypothesis h_1 is tested on a set T_1 od size n_1, a hypothesis h_2 is tested on a set T_2 of size n_2. We want to estimate the difference $d = \mathsf{Error}_D(h_1) - \mathsf{Error}_D(h_2)$ between their errors when the samples were drawn from the same underlying probability distribution D. Similarly to Section 2, we assume that the range of the target function is discrete and the error of the hypothesis can be computed as the fraction of incorrectly classified samples from the tested samples. An unbiased estimator for the difference of errors is

$$\widehat{d} = \mathsf{Error}_{T_1}(h_1) - \mathsf{Error}_{T_2}(h_2) \ .$$

This method has an disadvantage that the hypotheses have not only different errors but also variances. Hence the difference of the errors can be approximated by a normal distribution with the mean $\widehat{d}$ and variance

$$\sigma^2_{\widehat{d}} = \frac{\mathsf{Error}_{T_1}(h_1) \cdot (1 - \mathsf{Error}_{T_1}(h_1))}{n_1} + \frac{\mathsf{Error}_{T_2}(h_2) \cdot (1 - \mathsf{Error}_{T_2}(h_2))}{n_2} \ .$$

Therefore $N\%$ confidence interval for the estimator $\widehat{d}$ is:

$$\widehat{d} \pm z_N \sqrt{\sigma^2_{\widehat{d}}} \ .$$

3.2 Paired Tests

We are interested in comparing not just two hypothesis, but two learning *algorithms*. We can decrease he variance of the difference in their errors by training both algorithms on the same training set a testing them on the same test set, The obtained confidence intervals are narrower, because the difference in performance of the learning algorithms are then caused by the algorithm themselves, not by different input data.

Let L_A and L_B be two learning algorithms learning the same target function f. For the comparison, we should compare L_A and L_B by learning on all subsets of X of size n selected according to the same probabilistic distribution D. Then the expected value of the difference of error rates for the learning algorithms L_A and L_B is

$$E_{T \subset D}[\mathsf{Error}_D(L_A(T)) - \mathsf{Error}_D(L_B(T))] \ ,$$

where $L_\alpha(T)$ denotes the hypothesis output by the learning algorithm L_α after learning from the training set T ($\alpha \in \{A, B\}$). The expected value is computed on subsets (of size n) of X selected according to the probability distribution D.

When comparing learning algorithms in practice, we have only a limited sample D_0. Therefore we split D_0 randomly into a training set S_0 and a test set T_0, which are disjoint. The training samples are used to train both L_A and L_B. The test data is used for comparing accuracy of the trained hypotheses – the obtained error rates are subtracted:

$$\mathsf{Error}_{T_0}(L_A(S_0)) - \mathsf{Error}_{T_0}(L_B(S_0)) \ . \tag{1}$$

Hence the difference in error rates for L_A and L_B is measured on a single training set S_0 and not as the expected value of the difference on all samples S that might be drawn from the distribution D. If we have enough data, we can use them better.

3.3 k-Fold Cross-Validation

One way for improving the estimator given by Equation 1 is to partition the data D_0 into disjoint training and test sets and to take the mean of the test errors. The procedure is as follows:

1. Partition the data D_0 into k pairwise disjoint sets T_1, $T_2, \ldots, T_k$ of equal size. **The size of T_i should be at least 30!**

2. For $i = 1, \ldots, k$ do:
   take T_i for the test set and the remaining data for the training set S_i
   $$\begin{aligned} S_i &:= D_0 \setminus T_i \\ h_A &:= L_A(S_i) \\ h_B &:= L_B(S_i) \\ \delta_i &:= \mathsf{Error}_{T_i}(h_A) - \mathsf{Error}_{T_i}(h_B) \end{aligned}$$

3. The resulting value of the difference of errors of L_A a L_B is
   $$\bar{\delta} = \frac{1}{k} \sum_{i=1}^{k} \delta_i \ .$$

The approximate $N\%$ confidence interval the real difference of errors is

$$\left\langle \overline{\delta} - t_{N,k-1} \cdot s_{\overline{\delta}},\ \ \overline{\delta} + t_{N,k-1} \cdot s_{\overline{\delta}} \right\rangle,$$

where $s_{\overline{\delta}}$ is an estimate of the standard deviation

$$s_{\overline{\delta}} = \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \overline{\delta})^2}$$

and $t_{N,k-1}$ is the $\frac{N+100}{2}\%$ quantile of the student's t-distribution with $k-1$ degrees of freedom. In Matlab, we can compute $t_{N,k-1}$ as

```
tinv((N+100)/200,k-1).
```