Assignment 2: *k*-Fold Cross-Validation

In Matlab implement a function, which compares two learning algorithms using *k*-fold cross validation.

Suppose that learning algorithms are functions of the form

`LPar = `*name*`(Tr,DTr,Par)`,

where

name	is the name of the learning function,
`Tr`	is a training set (training vectors are the columns of `Tr`),
`DTr`	is a row vector of desired outputs, and
`Par`	is a cell array containing parameters of the learning algorithm.

Such function returns learned parameters `LPar` (also in a cell array).

Further we will need a function which will execute the learned function

`Out = `*menoL*`(LPar,In)`,

which computes the learned function with parameters `LPar` on input vectors from a matrix `In` (each column of the matrix is an input vector). `Out` is a row vector of the results.

Then it is possible to implement the function

`E = Err(Name,NameL,Par,Tr,DTr,Ts,DTs)`,

which computes the error of the function `NameL` learned by the algorithm `Name` with parameters `Par` on a training set `Tr` and desired outputs `DTr`. The error is computed on a test set `Ts` with the desired outputs `DTs`. `Name` and `NameL` are strings containing names of the respective functions. The respective functions can be evaluated using the function `feval`.

Then it is easy to implement the following function

```
[delta,s] =
    CrossVal(Name1,Name1L,Par1,Name2,Name2L,Par2,Pat,DOut,k),
```

which estimates the difference between the errors of the hypothesis `Name1` learned by a learning algorithm `Name1L` with parameters `Par1` and the error of the hypotheses `Name2` learned by a learning algorithm `Name2L` with parameters `Par2` using *k*-fold cross-validation on patterns `Pat` with the desired outputs `DOut`. The function returns the difference of errors `delta` and the standard deviation `s` of this estimator.

For example, we can compare the errors of the perceptron learning algorithm limited to at most 10 epochs and the perceptron learning algorithm limited to at most 100 epochs. As the perceptron learning algorithm we will use the function `PLearn` obtained from the function `perc_learn` from the seminary and similarly for simulating perceptrons we will use `perc_recall` from the seminary:

```
function LPar = PLearn(x,c,Par)
   p = perc_learn(Par{1},x,c,Par{2},Par{3});
   LPar = {p}
end

function Out = PRecall(Par,x)
   Out = perc_recall(Par{1},x);
end
```

Another learning algorithm we will implement is `Memorizer`, which memorizes all training samples together with the desired outputs for them. Learned `Memorizer` answers correctly on the inputs from the training set and randomly otherwise:

```
function LPar = Memorizer(Tr,DTr,Par)
    % The learning algorithm which only remembers all training
    % samples and the desired answers for them
    %
    %     LPar = Memorizer(Tr,Dtr,Par)
    %
    % inputs:
    %   Tr   matrix with training samples in columns
    %   DTr  row vector of the desired outputs
    %   Par  is not used here, it is given here for
    %        compatibility only
    %
    % output:
    %   LPar a cell array containing the training samples and
    %         the desired outputs for them

    LPar = {Tr,DTr};
end

function Out = MemorizerRecall(LPar,In)
    % A function simulating Memorizer
    %
    %     Out = MemorizerRecall(LPar,In)
    %
    % inputs:
    %   Lpar  a cell array with training samples and
    %         the respective desired outputs
    %   In    a matrix of input vectors (as columns)

    % output:
    %   Out   a row vector; whenever the i-th input vector is
    %         contained within the memorized input samples,
    %         Out(i) equals the i-th remembered desired output,
    %         otherwise it will be randomly 0 or 1

    Known = LPar{1};
    KnownOut = LPar{2};
    j=1;
    Out = zeros(1,size(In,2));
    for i = 1:size(In,2)
        j=1;
        while j<=size(Known,2)
            if In(:,i)==Known(:,j)
                break
            else
```

```
                j=j+1;
            end
        end
        if j<=size(Known,2)
            Out(i) = KnownOut(j);
        else
            Out(i) = rand(1)>0.5;
        end
    end
end
```

The above algorithms can be compared using the above function `CrossVal`.

```
% at first we reset the random number generator; in this
% way we obtain in each run the same training data
stream = RandStream.getGlobalStream; reset(stream)

% generate random training samples
In = randn(2,600);
% generate random desired outputs for the training samples
c = In(1,: )-3*In(2,: ) >= 0.5

% set the learning parameters for the perceptron learning
% algorithm:
%     an extended weight vector,
%     a learning rate, and
%     a maximal number of epochs
Par1 = {[1 1 -1], 1, 10}
Par2 = {[1 1 -1], 1, 100}

#run 5-fold cross-validation
[d,s] = CrossVal('PLearn','PRecall',Par1,'PLearn','PRecall',
                  Par2,In,c,5)
```

Tasks:

Implement the function `CrossVal` and by running the above script estimate the difference in error rates of the perceptron learning algorithm with at most 10 epochs and of the same perceptron learning algorithm with at most 100 epochs. From the resulting error difference and the standard deviation of the estimate compute interval which contains the true error difference with the probability 95%. Is the error difference statistically significant? Then modify the above script to compare error rates of the algorithm `Memorizer` and perceptron with at most 50 epochs using 6-fold cross-validation on the following 300 samples:

```
% at first we reset the random number generator; in this
% way we obtain in each run the same training data
stream = RandStream.getGlobalStream; reset(stream)
% generate random desired outputs for the training samples
In = randi(20,4,300);

c = In(1,: )-3*In(2,: )+2*In(3,: )-In(4,: )>= 0
```

You should submit:

1. A Zip-file with commented source code of the function `CrossVal` and all the functions you have used for solving the above tasks.

2. A text file (PDF is the preferred format) describing your solution and containing the results of your experiments. You should analyze the obtained results and explicitly write a recommendation which algorithm is better to use. Eventually you can give an advice which experiments would be suitable for more thorough comparison of the considered learning algorithms.