Machine Learning	Prof. Dr. Bernt Schiele
	Dr. Paul Swoboda & Dr. Gerard Pons-Moll
Solution 6 - 26.11.2018	due: 03.12.2018, 14:15

Exercise 11 - Construction of Kernels

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive definite kernel. Prove that the following functions $k'(x, y)$ are again positive definite kernels:

a. **(2 points)** $k'(x, y) = \sum_{r=0}^{\infty} \alpha_r k(x, y)^r$ for $\alpha_r \geq 0, r \geq 0$.

b. **(2 points)** $k'(x, y) = e^{-\lambda(d_k^2(x,y))}$, where $\lambda > 0$ and $d_k(x, y)$ is the (semi-)metric induced by the kernel k, $d_k^2(x, y) = k(x, x) + k(y, y) - 2k(x, y)$.

c. **(1 point)** $k'(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$.

d. **(2 Bonus points)** $\mathcal{X} = \{x \in \mathbb{R}^d \mid \|x\|_2 < 1\}$ and $k'(x, y) = \frac{1}{1-\langle x,y\rangle}$

Solution:

a. First of all, we have to assume that the series $\sum_{r=0}^{\infty} \alpha_r k(x, y)^r$ is convergent.

Since the pointwise product of two kernels $k_1(x, y)k_2(x, y)$ is again positive definite, $k(x, y)^2$ is a kernel, and by induction also any higher power $k(x, y)^r, r \geq 2$. Furthermore, $k(x, y)^0 = 1$ as a positive constant is a kernel.

As positive definiteness is preserved by multiplication by a positive constant, it holds that $\alpha_r k(x, y)^r$ is a kernel $\forall \alpha_r \geq 0, r \geq 0$.

Moreover, the pointwise sum of two kernels again yields a kernel, and by induction one shows that also $\sum_{r=0}^{n} \alpha_r k(x, y)^r$ is a kernel for $n \geq 0$.

Finally, as it is the pointwise limit of a sequence of positive definite kernels, also $k'(x, y)$ is a kernel.

b. Note that we can rewrite k' as

$$k'(x, y) = e^{-\lambda d_k^2(x,y)} = e^{-\lambda k(x,x)} e^{-\lambda k(y,y)} e^{\lambda 2k(x,y)} \ .$$

Using the series expansion of the exponential function we can rewrite the last term as

$$e^{2\lambda k(x,y)} = \sum_{r=0}^{\infty} \frac{(2\lambda)^r}{r!} k(x, y)^r \ ,$$

and choosing $\alpha_r = \frac{(2\lambda)^r}{r!}$ we can conclude from the result of the previous exercise that this is a positive definite kernel. Finally, since

$$k'(x, y) = f(x)f(y)k(x, y)$$

is a kernel for any function $f : \mathcal{X} \to \mathbb{R}$, it follows with $f(x) = e^{-\lambda k(x,x)}$ that $k'(x, y)$ is a kernel.

c. This follows as a special case from part b) for $\lambda = \frac{2}{\sigma^2}$ and $k(x, y) = \langle x, y\rangle$, since

$$d_k^2(x, y) = \langle x, x\rangle + \langle y, y\rangle - 2\langle x, y\rangle = \|x - y\|^2 \ .$$

d. We realize that $\frac{1}{1-x} = \sum_{r=0}^{\infty} x^r$ for $|x| < 1$ and thus $\frac{1}{1-\langle x,y\rangle}$ is a positive definite kernel according to a) as $k(x, y) = \langle x, y\rangle$ is a positive definite kernel and we have $|\langle x, y\rangle| \leq \|x\|_2 \|y\|_2 < 1$ as $\|x\|_2 < 1$ for all x in the domain of the kernel.

Exercise 12 - Multiclass schemes for classification of handwritten digits

In this exercise we are doing handwritten digit classification using multi-class SVM with a Gaussian kernel. In order to solve the optimization problem for the SVM, we are using the MATLAB interface to the LIBSVM package (`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`). Download `DataEx6.zip` from the course webpage. Note: You do not need to download anything from the LIBSVM webpage - everything you need is contained in the zip-file.

- Extract the files in libsvm-3.14.zip somewhere in your home directory.
- Start Matlab and use addpath to add the directory where you have extracted LIBSVM to the MATLAB search path (use savepath in order to add it permanently).
- Go to the subfolder matlab and type make in the Matlab prompt. (Pre-built binary files for Windows 64bit already contained in the folder).
- The matlab function `getKernelSVMSolution` provides a nice interface to the LIBSVM package (use `help getKernelSVMSolution` to see how it works).
- **(4 Points)** The problem deals with the classification of handwritten digits (10 classes). You are supposed to use the SVM with the Gaussian kernel:

  $$k(x, y) = e^{-\lambda \|x-y\|^2}.$$

  The training and test data is in `USPSTrain.mat` and `USPSTest.mat`. The 16×16-images of the digits are represented as 256-dimensional column vectors. Write two matlab scripts:

  - one which solves the multi-class problem using **one-versus-all** (save it in `OneVersusAll.m`),
  - one which solves the multi-class problem using **one-versus-one** (save it in `OneVersusOne.m`),

  In both cases use $C = 100$ and $\lambda = \frac{3}{\gamma}$, where γ is the median of all squared distances between **training** points, as parameters for the binary SVM.

  Visually inspect the digits which have been misclassified. How do you judge the result? Compare the quality of the classification obtained by the two multi-class schemes. How do the two multiclass schemes compare in terms of runtime?

  Save your prediction on the test set in a file `USPSResults.mat` as `PredOneVersusOne` and `PredOneVersusAll` and report the test error for both cases. Also generate for both cases a figure (`ErrorsOneVersusOne.png` and `ErrorsOneVersusAll.png`) containing the misclassified images in the test set.

- **(2 Points)** Suppose the computation of a binary classifier has complexity $O(n^m)$, where n is the number of training points. Suppose we have k classes and the training set contains $\frac{n}{k}$ points of each class. What is the computational complexity of the one-versus-all and one-versus-one scheme? Which multi-class scheme is better in terms of complexity in terms of n? And in terms of k?

Hints:

- Use `dist_euclidean.m` to compute the squared Euclidean distances between two sets of points (warning: your own code may be too slow for this dataset), `getKernelSVMSolution.m` to obtain the dual variables α and the offset b of the SVM and `VecToImage.m` to plot the images of the digits which have been wrongly classified.
- Given a matrix D, with $D_{ij} = \|X_i - Z_j\|^2$, $i = 1, \dots, n, j = 1, \dots, m$, of the squared distances between two point sets $\{X_1, \dots, X_n\}$ and $\{Z_1, \dots, Z_m\}$ with $X_i \in \mathbb{R}^d$ and $Z_j \in \mathbb{R}^d$, you can compute the kernel matrix `K` in Matlab as: `K=exp(-lambda*D);`.

Solution:

a. The two different multi-class schemes:

- One versus all:

```
clear all;
addpath DataEx6/libsvm-3.14/matlab/;
load USPSTrain;

num=size(Xtrain,1); dim=size(Xtrain,2);
C=100;
lambda=3;
Classes = CheckLabelVector(Ytrain);

D=dist_euclidean(Xtrain,Xtrain);
K=exp(-lambda*D/median(D(:)));


% one - versus - all
for i=1:length(Classes)
  LabelVec = 2*(Ytrain==Classes(i))-1;
  [alpha(:,i),b(i)]=getKernelSVMSolution(K,LabelVec,C);
end


% compute the training error
for i=1:length(Classes)
 LabelVec = 2*(Ytrain==Classes(i))-1;
 OutputTrain(:,i) =K*(alpha(:,i).*LabelVec)+b(i);
end
[Max,PredTrain] = max(OutputTrain');
trainError = sum(Classes(PredTrain)~=Ytrain);
disp(['-----------------------------------------------------------------------------------------------------------']);
disp(['Number of training errors: ',num2str(trainError),' - Percentage: ',num2str(trainError/length(Ytrain))]);

load USPSTest;
D2=dist_euclidean(Xtest,Xtrain);
Ktest=exp(-lambda*D2/median(D(:)));


% compute the test error
for i=1:length(Classes)
 LabelVec = 2*(Ytrain==Classes(i))-1;
 OutputTest(:,i) =Ktest*(alpha(:,i).*LabelVec)+b(i);
end
[Max,PredTest] = max(OutputTest');
testError = sum(Classes(PredTest)~=Ytest);

disp(['Number of test errors: ',num2str(testError),' - Percentage: ',num2str(testError/length(Ytest))]);
```

- One versus one:

```
clear all;
addpath DataEx6/libsvm-3.14/matlab/;
load USPSTrain;

num=size(Xtrain,1); dim=size(Xtrain,2);
C=100;
lambda=3;
Classes = CheckLabelVector(Ytrain);

D=dist_euclidean(Xtrain,Xtrain); medD=median(D(:));
```

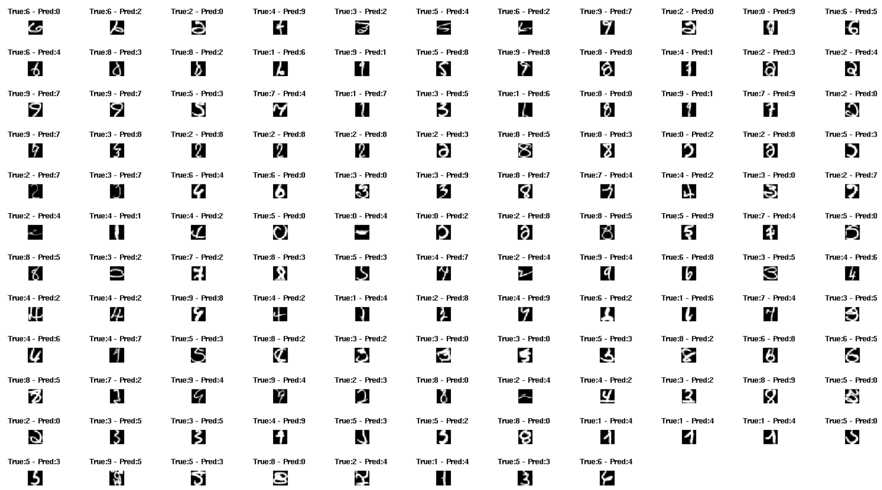
```
K=exp(-lambda*D/medD); clear D;
% one - versus - one
b     = zeros(length(Classes),length(Classes));
for i=1:length(Classes)
  for j=i+1:length(Classes)
     Indices = find(Ytrain==Classes(i) | Ytrain==Classes(j));
     LabelVec = 2*(Ytrain(Indices)==Classes(i))-1;
     [alpha{i,j},b(i,j)]=getKernelSVMSolution(K(Indices,Indices),LabelVec,C);
  end
end
% compute the training error
Votes = zeros(length(Ytrain),length(Classes),length(Classes));
for i=1:length(Classes)
 for j=i+1:length(Classes)
   Indices = find(Ytrain==Classes(i) | Ytrain==Classes(j));
   LabelVec = 2*(Ytrain(Indices)==Classes(i))-1;
   Votes(:,i,j) = K(:,Indices)*(alpha{i,j}.*LabelVec)+b(i,j)>0;
   Votes(:,j,i) = 1-Votes(:,i,j);
 end
end
MajVotes=sum(Votes,3);
[Max,PredTrain] = max(MajVotes');
trainError = sum(Classes(PredTrain)~=Ytrain);
disp(['Number of training errors: ',num2str(trainError),' - Percentage: ',num2str(
    trainError/length(Ytrain))]);
load USPSTest;
D2=dist_euclidean(Xtest,Xtrain);
Ktest=exp(-lambda*D2/medD);
% compute the test error
clear Votes; clear MajVotes;
for i=1:length(Classes)
 for j=i+1:length(Classes)
   Indices = find(Ytrain==Classes(i) | Ytrain==Classes(j));
   LabelVec = 2*(Ytrain(Indices)==Classes(i))-1;
   Votes(:,i,j) = Ktest(:,Indices)*(alpha{i,j}.*LabelVec)+b(i,j)>0;
   Votes(:,j,i) = 1-Votes(:,i,j);
 end
end
MajVotes=sum(Votes,3);
[Max,PredTest] = max(MajVotes');
testError = sum(Classes(PredTest)~=Ytest);
disp(['Number of test errors: ',num2str(testError),' - Percentage: ',num2str(
    testError/length(Ytest))]);
```

For the parameters $C = 100$ and $\lambda = \frac{3}{\gamma}$ we have 129 errors (error rate 6.43%) for the one-versus-all scheme and 131 errors (error rate 6.53%) for the one-versus-one scheme. We see that they both perform almost in the same way, which is also an observation which often holds in practice. In particular, more complicated multi-class schemes have up to now not shown to be systematically better than the simple one-versus-all and one-versus-one scheme. In practice, often the one-versus-one scheme is used. In terms of runtime, we observe that the one-versus-one scheme is slightly faster (on the given dataset it takes ∼13 seconds compared to ∼18 seconds).

Code to plot the misclassified digits:

```
handles=VecToImage(Xtest(Classes(PredTest) =Ytest,:),16,16,0,2,1);
Index=find(Classes(PredTest) =Ytest);
for i=1:length(Index)
titleString=['True:',num2str(Ytest(Index(i))),
' - Pred:',num2str(Classes(PredTest(Index(i))))];
subplot(handles(i)); title(titleString,'FontWeight','bold');
end
```

The digits which have been wrongly classified for the **one-versus-all** scheme:



and for the **one-versus-one** scheme:

b. Computational complexity:

- **One-Versus-All:**
  We have to solve k binary classification problems (one for each class), each with the full set of training data points n.

$$\text{Complexity:} \quad kn^m$$

- **One-Versus-One:**
  We have to solve $\binom{k}{2}$ binary classification problems (each class vs. each other), each has a training set of $\frac{2n}{k}$ data points.

$$\text{Complexity:} \quad \frac{k(k-1)}{2}\Big(\frac{2n}{k}\Big)^m = \Big(\frac{2}{k}\Big)^{m-1}(k-1)n^m$$

We observe that the computational complexity in terms of the number of training examples n is the same. However, the dependency on the number of classes k is worse for the one-versus-all scheme. We want to check when $2^{m-1}k^{1-m}(k-1) \leq k$. Note that

$$2^{m-1}k^{1-m}(k-1) \;\leq\; 2^{m-1}k^{m-1}k$$

And thus $\left(\frac{2}{k}\right)^{m-1} \leq 1$ is a sufficient condition. It is easy to check that this hold for all $k \geq 2$ irrespectively of m. However, note that for $m = 1$ the difference is negligible.