perClass 5.4 cheat-sheet

help: `doc perclass` http://perclass.com/doc

version & license info: `sdversion` **send feedback:** `sdfeedback`

Labels

create labels by specifying individual sample labels
```
lab=sdlab({'apple','banana','apple','pear'})
```
create labels by giving class list and **vector of class sizes**
```
lab=sdlab({'apple','banana'},'sizes',[10 5])
```
create labels **per-sample names and indices**
```
lab=sdlab({'apple','banana','lemon'},[1 1 2 3])
```
create labels by giving class **name and number of samples**
```
lab=sdlab('apple',10,'banana',5)
```
find samples of a class
```
ind=find(lab=='apple')
```
find given class index
```
ind=find(lab==2)
```
get class index by name
```
lab.list('banana')
```
get class name by index
```
lab.list(2)
```
number of classes
```
length(lab.list)
```
number of entries (samples)
```
length(lab)
```
class/category sizes
```
lab.sizes
```
class/category fractions/priors
```
lab.fractions
```
subset of samples by index
```
lab(1:10)
```
test if class is present
```
isempty(lab.list('A'))
```
- `lab.list` contains only classes present in lab

display label details (classes, sizes and fractions): `lab'` or `info(lab)`

Data sets

create data set
```
a=sddata(matrix,lab)
```
raw data matrix
```
+a ; double(a)
```
access class labels
```
a.lab
```
display **data set details**
```
a' ; info(a)
```
subset of samples
```
a(1:10)
```
remove samples
```
a(1:10)=[]
```
subset of features by indices
```
a(:,[1 2 5])
```
subset of features by names
```
a(:,{'Skew','Entropy'})
```
select classes by names
```
a(:,:,{'apple','pear'})
```
select classes by indices
```
a(:,:,1:2)
```
set **new data matrix** in a data set (optionally provide new feature labels)
```
b=setdata(a,data)
```
```
b=setdata(a,data,featlab)
```
create new labels
```
a.patient=sdlab('Pat1',10,'Pat2',30)
```
create numerical property
```
a.pixel=1:1024
```
select samples by property
```
a( a.pixel<50 )
```
test if property exists
```
isprop(a,'patient')
```
remove sample property
```
rmprop(a,'patient')
```
make patient labels current
```
a=setlab(a,'patient')
```
find current label set
```
name=setlab(a)
```
- `a.lab` now points to `a.patient`

label all samples in one class
```
a.lab='pear'
```
change label for a subset
```
a(1:10).lab='banana'
```

Subsets

sample subset by values
```
b=subset(a,'lab','cancer','patient',1:3)
```
- get cancer samples from the first three patients

return also the **rest of samples**
```
[b,rest]=subset(a,'patient',5)
```
select classes **matching regular expression**
```
b=subset(a,'/substring')
```
iterate over classes
```
for i=1:length(a.lab.list), b=a(:,:,i); end
```

Random subsets

100 samples **per class**
```
sub=randsubset(a,100)
```
select 30% **per class**
```
b=randsubset(a,0.3)
```
select 100 samples **per patient**
```
sub=randsubset(a,'patient',100)
```
select 100 samples **from complete data set**
```
[sub,rest]=randsubset(a,100,'all')
```
Bootstrap sampling (with replacement): `sub=randsubset(a)`

Classifiers

train a classifier (returns decisions by default)
```
p=sdlinear(a)
```
```
p=a*sdlinear
```
- if trained on one class, all training samples are accepted

get decisions
```
dec=b*p
```
get data with decision labels
```
c=b.*p
```
display pipeline details
```
p' ; info(p)
```
get type of pipeline output
```
p.output
```
remove decision step
```
-p
```
get soft outputs
```
out=b * -p
```
all classifier decisions
```
p.list
```
get output labels (soft outputs)
```
p.lab
```
- `dec.list` or `c.lab.list` contains **all classes present** in `dec` (may be less than in `p.list`)
- `p.inlab` gets input labels (features names of training data)

Relabeling

relabel data set (what is not *stone* becomes *fruit*)
```
a2=sdrelab(a,{'~stone','fruit'})
```
relabel first two classes into *fruit* **and** *stone* class in *other*
```
a=sdrelab(a,{1:2,'fruit','stone','other'})
```
relabel by **regular expression** (*apple* or *pear* becomes *fruit*)
```
a=sdrelab(a,{'/apple|pear','fruit'})
```
relabel label set, return label set (can be set as new in the data set)
```
a.gender=sdrelab(a.patient,
        {'/male','male','~male','female'})
```
add prefix to class names
```
a2=sdrelab(a,'add','new-')
```
add prefix to all class names **in all sets of labels**
```
a2=sdrelab(a,'add to all','new data-')
```
relabel classifier decisions
```
p=sdrelab(p,{'/apple|banana','fruit'})
```

Detection

train on one class, accept all training samples (models marked **d** in table)
```
sdparzen( a(:,:,'class_name') )
```
train a detector (data, target class, model)
```
pd=sddetect(a,'banana', sdknn)
```
- change decision names with 'target' and 'non-target' opts.

train a one-class detector rejecting 10% of bananas
```
pd=sddetect(a,'banana',sdparzen,'reject',0.1)
```
train a detector **on all samples** (by default accept all)
```
pd=sddetect(a,'all', sdknn)
```
- use target name that is not present in `a.lab.list`

provide a set to estimate ROC (does not split `tr`)
```
pd=sddetect(tr,'banana',sdgauss,'test',val)
```
- use `pd.roc` to access ROC curve (empty for one-class detectors)
- specify **ROC performance measures** with `measures` option

Classifier combining and cascades

Create **classifier cascade**
```
pc=sdcascade(pd1,'fruit',pd2)
```
- `pd1` is *fruit/non-fruit* detector, `pd2` *apple/banana* classifier

Combine classifiers: `p1=sdlinear(a);p2=sdgauss(a);`

soft-output fusion
```
pc=sdcombine([-p1 -p2])
```
crisp (decision) fusion
```
pc=sdcombine([p1 p2])
```

Available classifiers

d *... can be used for detection*

Nearest mean	**d**	`sdnmean`
Fisher linear discriminant		`sdfisher`
Gaussian model (for detection)	**d**	`sdgauss`
Linear discriminant		`sdlinear`
Quadratic discriminant		`sdquadratic`
Least mean square classifier (also regression)		`sdlms`
Logistic classifier		`sdlogistic`
Parzen	**d**	`sdparzen`
Mixture of Gaussians	**d**	`sdmixture`
k-NN	**d**	`sdknn`
k-means prototype extraction	**d**	`sdkmeans`
Support Vector Machine	**d**	`sdsvc`
Neural network (feed-forward or RBF)		`sdneural`
Deep convolutional network		`sddeepnet`
Naïve Bayes	**d**	`sdnbayes`
Decision tree		`sdtree`
Random forest		`sdrandforest`
Minimum distance classifier (on `sdprox` output)	**d**	`sdmindist`
Look-up table (2D classifier approximation)		`sdlut`

Rejection

Add reject option to a classfiier (reject 5% of samples in `a`)
```
p=sdparzen(a); pr=sdreject(p,a,'reject',0.05)
```
- to reject low confidence, make normalized outputs with `sdnorm`

Reject specific **number of samples**
```
pr=sdreject(p,a,'reject',10)
```
Find samples that are not rejected: `dec=a*pr; a(dec~='reject')`

Evaluation

find error on a test set (default mean error over classes)
```
err=sdtest(p,b)
```
compute **specific performance measures**
```
perf=sdtest(p,ts, 'measures',{'TPr','apple',
            'mean-error','precision','banana'})
```
confusion matrix (use `'norm'` option to normalize)
```
sdconfmat(a.lab,a*p)
```
specific order of classes (rows) **or decisions** (columns)
```
sdconfmat(a.lab,a*p,'classes',
{'lemon','apple'},'decisions',{'lemon','apple'})
```
confusion matrix **in a figure**
```
sdconfmat(a.lab,a*p,'figure')
```
get sample indices **in a specific entry** of confusion matrix
```
ind=sdconfmatind(a.lab,a*p,'apple','banana')
```
- `a(ind)` are *apple* samples, labeled as *banana*

cross-validate a classifier (10-fold rotation)
```
p=sdlinear; [s,res]=sdcrossval(p,a)
```
- `s` is a string summary, `res` struct with results

cross-validation by **randomization** (30-fold, 80% in training)
```
s=sdcrossval(p,a, 'method','random',0.8,...
             'folds',30,'seed',42)
```
retrieve training, test subsets and the classifier in fold 5
```
[s,res,e]=sdcrossval(p,a);
tr=gettrdata(e,a,5);  ts=gettsdata(e,a,5);
p=e(5); dec=ts*p
```
leave-one-out over patients
```
res=sdcrossval(p,a,'method','loo', 'over','patient')
```
simple leave-one-out **loop over patients**
```
for i=1:length(a.patient.list)
   [ts,tr]=subset(a,'patient',i);
   p=sdlinear(tr);
   err(i)=sdtest(ts,p);
end
```

Performance measures

		decisions		sum
		target	non-target	
true labels	target	**TP**	**FN**	**Nt**
	non-target	**FP**	**TN**	**Nn**

Accuracy	FN/Nt+FP/Nn
True positive ratio (recall,sensitivity)	TPr=TP/Nt
True negative rate (specificity)	TNr=TN/Nn
Precision (purity)	TP/(TP+FP)
Positive fraction (posfrac)	(TP+FP)/N
Detection rate (detrate)	(TP+FP)/Nt

measures: mean-error, class-errors, TP,TN,FP,FN, TPr,FPr,TNr,FPr, sensitivity, specificity, precision, posfrac, detrate

ROC analysis

estimate ROC characteristic (two- or multi-class)
```
[tr,ts]=randsubset(a,0.5); p=sdlinear(tr);
r=sdroc(ts,p)
```
estimate ROC from soft outputs (use `-p` to remove decision step)
```
out=ts*-p; r=sdroc(out)
```
draw interactive ROC plot
```
sddrawroc(r)
```
- select op.point by clicking; press 's' to save it back to workspace

show interactive confusion matrix (also via ROC plot by pressing 'c')
```
sdconfmat(r)
```
- click on confmat entries to define constraints (also precision)
- up/down or mouse scroll for direct error minimization

add ROC to classifier
```
p=p*r
```
access ROC stored in the classifier
```
p.roc
```
create ROC **with specific measures**
```
r=sdroc(out,'measures',{'FPr','apple','TPr','apple'})
```
get performances at op.point 10
```
r(10)
```
estimate ROC **for user-defined class weights**
```
pd=sddecide('w',rand(10000,3),a.lab.list)
r=sdroc(pd,out)
```
constrain ROC to a subset with error on apple<0.3
```
r2=constrain(r,'err(apple)',0.3)
```
set curent operating point by index
```
r=setcurop(r,100);          ind=getcurop(r);
```
set curent operating point by constraining TPr **and** minimizing FPr
```
r=setcurop(r,'constrain','TPr(apple)',0.99,...
             'min','FPr(apple)')
```
set operating point **minimizing the cost** (confmat is stored)
```
M=ones(3); M(1,2)=10; r=sdroc(out,'confmat');
r=setcurop(r,'cost',M)
```

Dimensionality reduction

Principal Component Analysis	`sdpca`
Fisher projection (class separation)	`sdlda`
Proximity representation (distance to prototypes)	`sdprox`
Scaling data	`sdscale`
Polynomial feature space expansion	`sdexpand`
Preprocessing/spectral indices	`sdprep`
Feature selection	`sdfeatsel`

Forward feature selection with 1-NN error as criterion
```
pf=sdfeatsel(data,'forward')
```
Backward selection **using error of a specific model**:
```
pf=sdfeatsel(data,'backward','model',sdfisher)
```
select fixed feature subset
```
pf=sdfeatsel(a,[1 5 10])
```
using regular expression
```
sdfeatsel(a,'/substr')
```
- see `pf.lab` for output features and `pf.inlab` for input features

Select features with non-zero variance `sdfeatsel(a,'var>0')`

Interactive visualization

interactive scatter plot: `sdscatter(a)` **keyboard commands:**

show all keyboard shortcuts	?
change feature	→/←/↑/↓
change z-order of classes	+/- or =
cycle through classes one at a time	</>
show/hide legend	l
rename class (also class merging)	r
hide current class	h
show only this class	o
invert filter on this label set	i
remove filter on this label set	R
tag sample under cursor (also via double-click)	t
label visible samples as...	L
switch to label set	1:9
switch between full data set and subset axes	a
switch axes limits between visible samples / space of total set	v
show/hide feature distributions	d
show/hide confusion matrix with visible/hidden samples	c
return to previous sample filter	f
show all samples (remove filter)	F

classifier decisions (also in multi-dimensional space)
```
p=sdlinear(a); sdscatter(a,p)
```
- change color of decision backdrop under cursor: c

open also connected **ROC plot for scatter/image view**
```
sdscatter(a,p*r,'roc');   sdimage(im,p*r,'roc')
```
show feature distributions
```
sdfeatplot(a)
```
- right-click to control interactive threshold selection

Cluster analysis

cluster data with k-means algorithm into 10 clusters
```
b=sdcluster(a,sdkmeans,10)
```
train k-means model returning cluster labels
```
p=sdkmeans(a,10,'cluster')
```
get labels
```
lab=a*p
```
protect clustering from outliers
```
pr=sdreject(p,a)
```
execute and set decisions as new labels `b=a.*p`

Execution out-of-Matlab

Export classifier for execution: `sdexport(p,filename)`
- Classifier can be run from any application via `perclass.dll` runtime
- For details see: http://perclass.com/doc/guide/deployment.html

Export classifier as a **C header file for static linking**:
```
sdexport(p,filename,'header')
```

Feature extraction

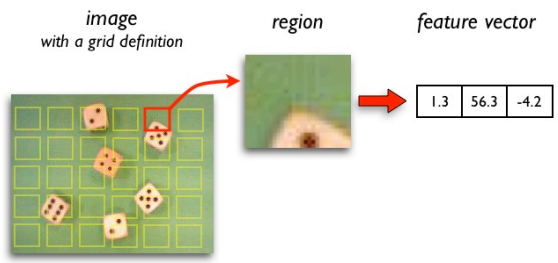
general extraction syntax

```
out=sdextract(data, domain, feat [,options] )
```

domains:

- `region`: Extract features from **local image neighborhoods**
- `object`: Extract one feature **vector per object** (via object labels)
- `bands`: Extract features for bands in spectra
- `color`: Transform color space

Region features



- `raw` image content (in: any, out: block*block*inbands)
- `moments` **mean and std** per neighborhood (in: 1, out: 2)

```
out=sdextract(im(:,1),'region','moments')
```

- `hist` Local **histogram** (in: 1, out: bins)
  - requires data `range`
  - optional number of `bins` (def: 8)

```
out=sdextract(im(:,1),'region','hist','range',[0 255])
```

- `histfeat` statistical **features of local histogram** (in: 1, out: 5)
  - requires data `range`
  - optional number of `bins` (def: 8)
- `cm` **Co-occurrence matrix** (in: 1, out: bins^2)
  - requires data `range`
  - optional number of `bins` (def: 8)
- `gauss` Convolution with **Gaussian kernel or derivative** (in: 1, out: 1)
  - `sigma` smoothing (def: 2.0)
  - `der` derivative (def: 0)

```
a=sdextract(im,'region','gauss','sigma',3,'der',[1 0])
```

- `sobel` Gradient magnitude and orientation (in:1, out:2) , always `block=3`
  - additional names: `sobel.x` , `sobel.y` , `sobel.mag` (magnitude only)

Grid definition

Neighborhood size with `block` (default: `block=8`)

Change grid `step` (default: `step=1`)

```
a=sdextract(im,'region','moments','block',16,'step',2)
```

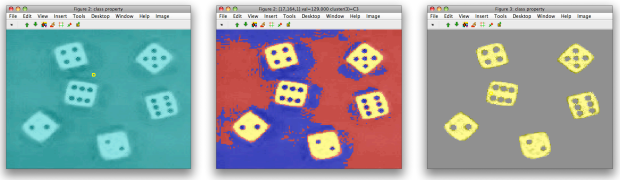
Remove grid from image (shrink to grid)

```
b=sdimage(a,'grid')
```

Image subsets

Subset of an image is still image

```
c=sdcluster(im,sdkmeans,3);
sub=c(:,:,1)   % choose cluster 1 (dice in our example)
sdimage(im);        sdimage(c);      sdimage(sub)
```



- use **rename class** ('r' key) to assign meaningful names and merge

Object segmentation (connected components)

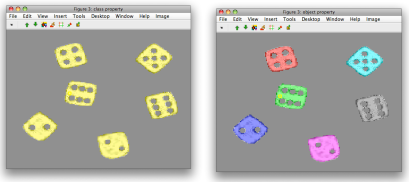
Find **connected components** using current label set in `sub`:

```
A=sdsegment(sub,'minsize',100)
```

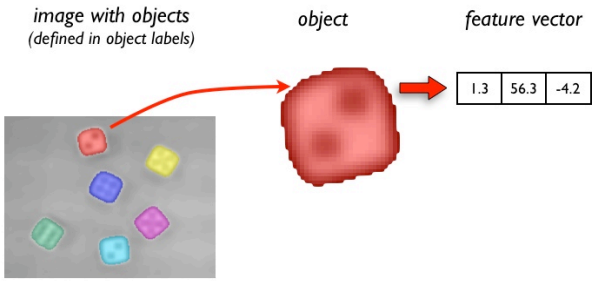
- segmentation is performed per-class and stored in `A.object` labels
- object labels are set as current (`A.lab` == `A.object`)
- objects of class `dice` are labeled as `'dice-object N'`
- regions smaller than `minsize` joined in `'dice-small objects'`

Subset of large objects from dice class: `A(:,:,'/^dice-obj')`

- regular expression: object name starts with `dice-obj`



Object features



Features working on object labels only:

- `size` Object size (in: any, out: 1)
  - works on only object labels

Features computed from object content (no use of spatial info):

- `mean` Mean vector for each object (in: any, out: same as input)
- `sum` Sum vector for each object (in: any, out: same as input)
- `hist` Histogram of one input feature per-object (in: 1, out: bins)
  - requires data `range`
  - optional number of `bins` (def: 8)

```
b=sdextract( im(:,1) ,'object','hist','range',[0 255])
```

Features computed from object content (use spatial info)

- `gray shape` Hu moments and shape eigenvalues (in: 1,out: 9)

Features computed from object shape (spatial info)

- `shape` Hu moments and shape eigenvalues (in: any, out: 9)
  - computed on object binary mask

Features computed from local labels inside objects

- `class fractions` Fraction of classes present in provided pixel/region labels. This is useful to find defects inside objects.

```
im=sdsegment(im .* pobjdet)    % separate objects/tray
dec=im * ppixel       % get per-pixel defect decisions
obj=sdextract(im,'object','class fractions',dec)
```

- `obj` contains one sample for each of `length(im.object.list)` objects

Back project object decisions to orig.image

Feature extraction

- `sdpca(a,frac)` Principal component analysis
  - frac <1 - fraction of preserved variance
  - frac >1 - number of output dimensions
- `sdlda` Linear discriminanc analysis
- `sdscale(data,method)` Feature scaling
  - 'variance'

Non-parametric classifiers

- `sdknn` **k-th nearest neighbour**
  - `sdknn(data,10)` - by default returns distance to k-th neighbour in each class (can be used as a detector)
  - `sdknn(data,'classfrac','k',10)` - returns confidence (fraction of samples of each class in k neighbors). Discriminant only.
  - *k* may be any integer (also even)
  - trained pipelines allow change of k: `p(1).k=7`
- `sdkmeans` **k-means as a classifier (detector or discriminant)**
  - `sdkmeans(data,10)` runs k-means clustering for each class and returns 1-NN classifier to the extracted prototypes

- `sddeepnet` Convolutional neural network
  -