

# コンピュータビジョン 画像認識-I

# 画像認識技術の現状

実用レベル

研究途上

1970

1980

1990

2000

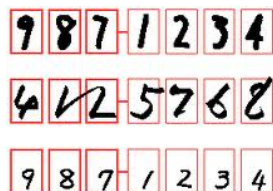
2010

???

バーコード



手書き文字



指紋



印刷文字

CHAPTER 1  
Eigenvalues, eigenvectors, and similarity

1.0 Introduction

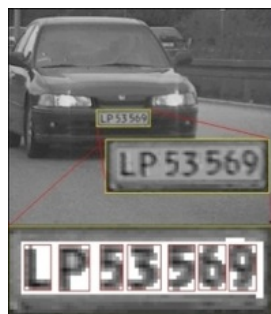
In this and all the following chapters, we motivate some key issues discussed in the chapter with examples of how they arise, either conceptually or in applications.

1.0.1 Change of basis and similarity. Every invertible matrix is a change-of-basis matrix, and every change-of-basis matrix is invertible (see Section 0.10). Thus, if  $B$  is a given basis of a vector space  $V$ , if  $T$  is a given linear transformation on  $V$ , and if  $A = a_{ij}T$  is the  $B$ -basis representation of  $T$ , the set of all possible basis representations of  $T$  is  $\{a_{ij}T \mid a_{ij}T \in GL_n(\mathbb{R})\}$ , which is a basis of  $V$ .

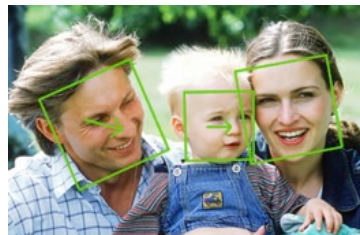
$M = (M_{ij})$  is an invertible matrix. This is just the set of all matrices that are similar to the given matrix  $A$ . Similar but not identical matrices are therefore just different basis representations of a single linear transformation. One would expect similar matrices to have many important properties - at least, those properties that are intrinsic to the underlying linear transformation - and this is an important theme in linear algebra. It is often useful to step back from a question about a matrix to a question about some intrinsic property of the linear transformation of which it is only one of many possible representations. The notion of similarity is a key concept in this chapter.

10

ナンバープレート



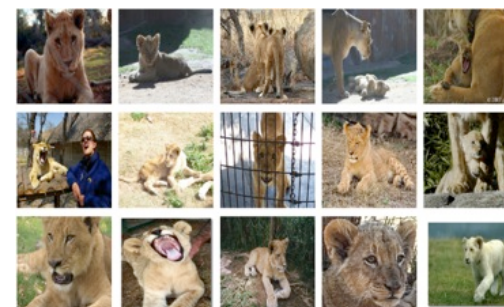
顔



道路標識



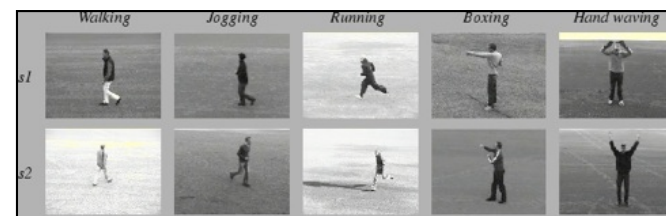
一般物体



材質・質感



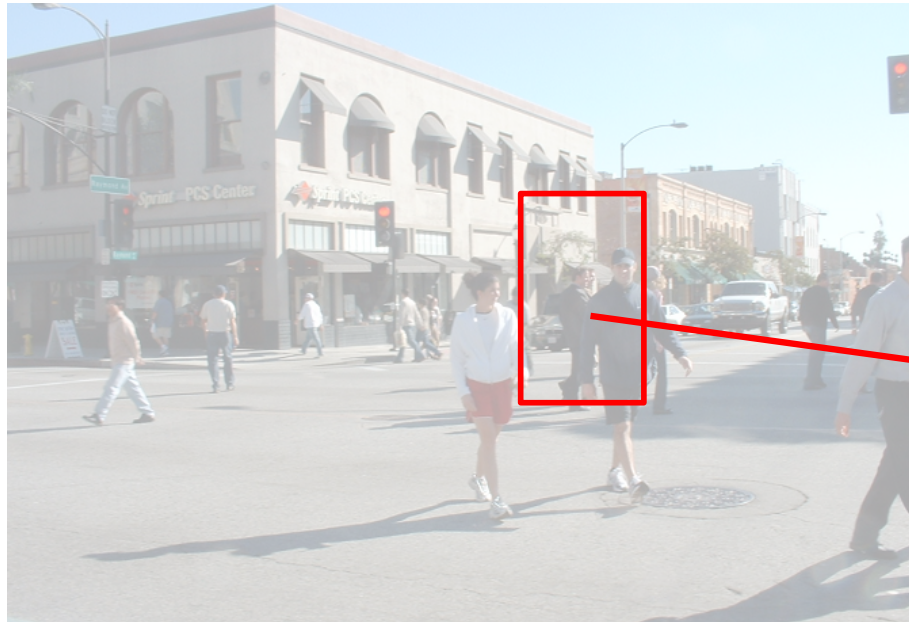
人の行動



# 画像認識—人物検出を例に



# 画像認識—人物検出を例に



小領域ごとに  
人かどうかを判断

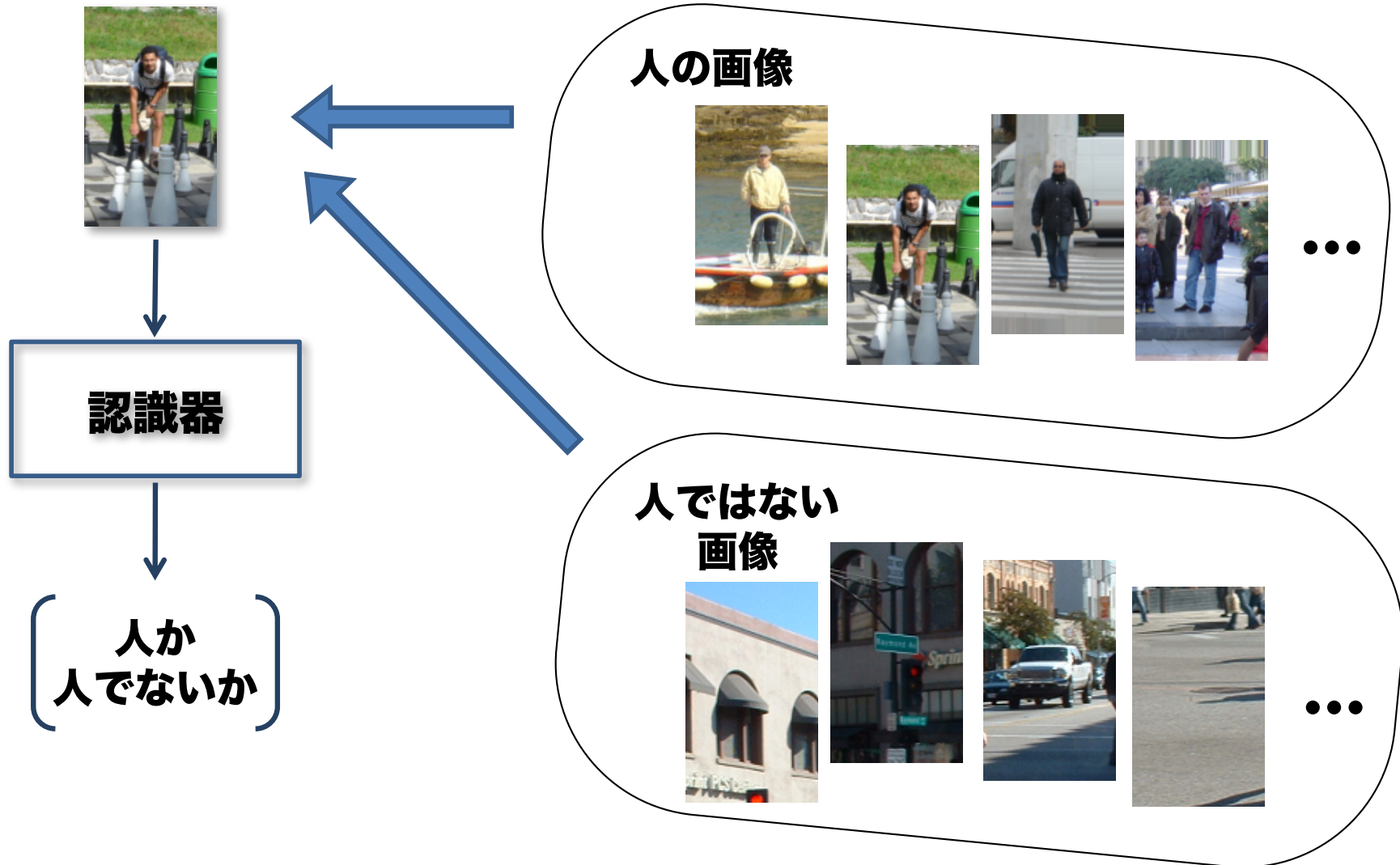


認識器

人か  
人でないか

# 画像認識—人物検出を例に

## 2クラス分類器をデータから学習



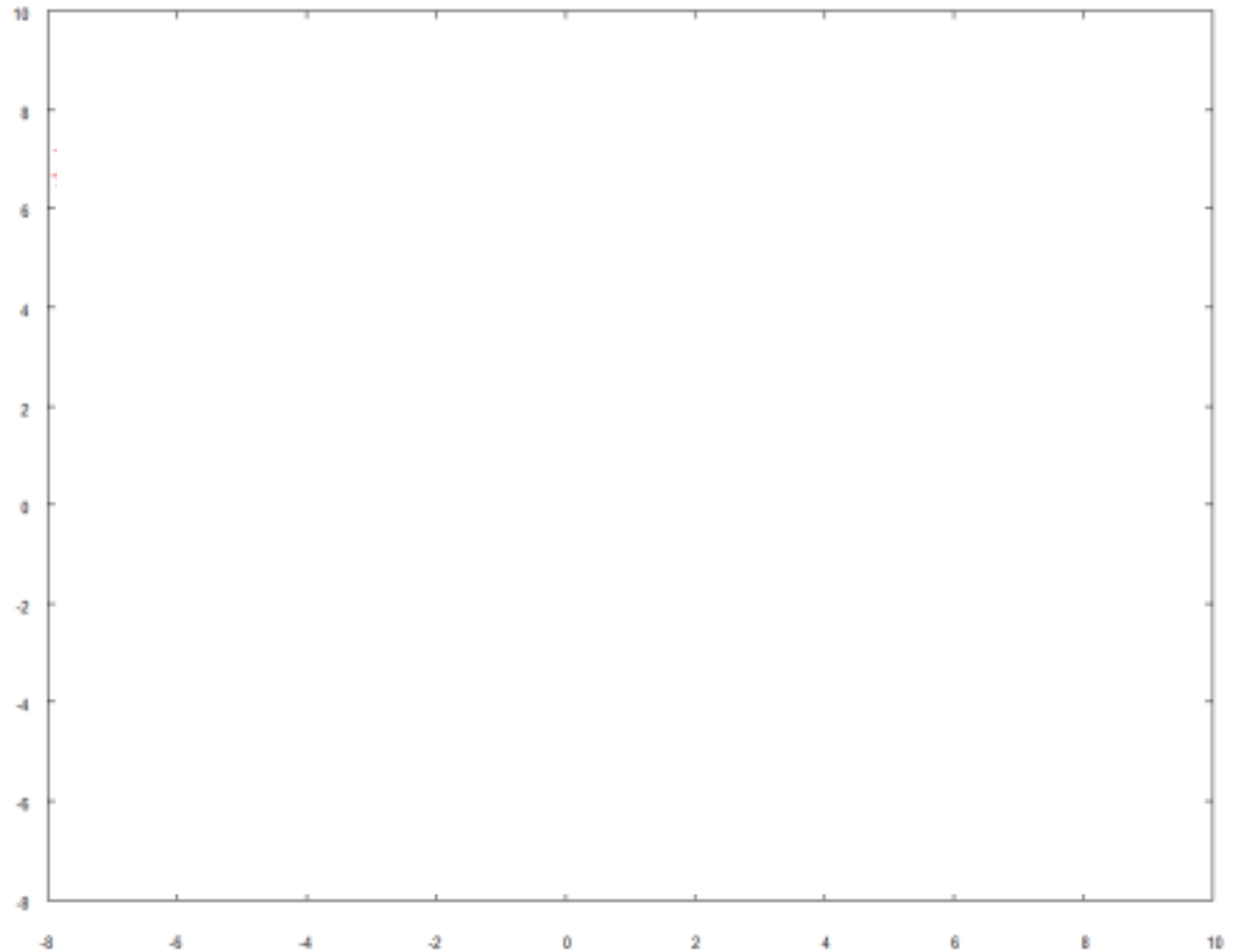
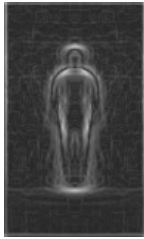


# 画像認識—人物検出を例に

特徴抽出：「人らしさ」を取り出す写像

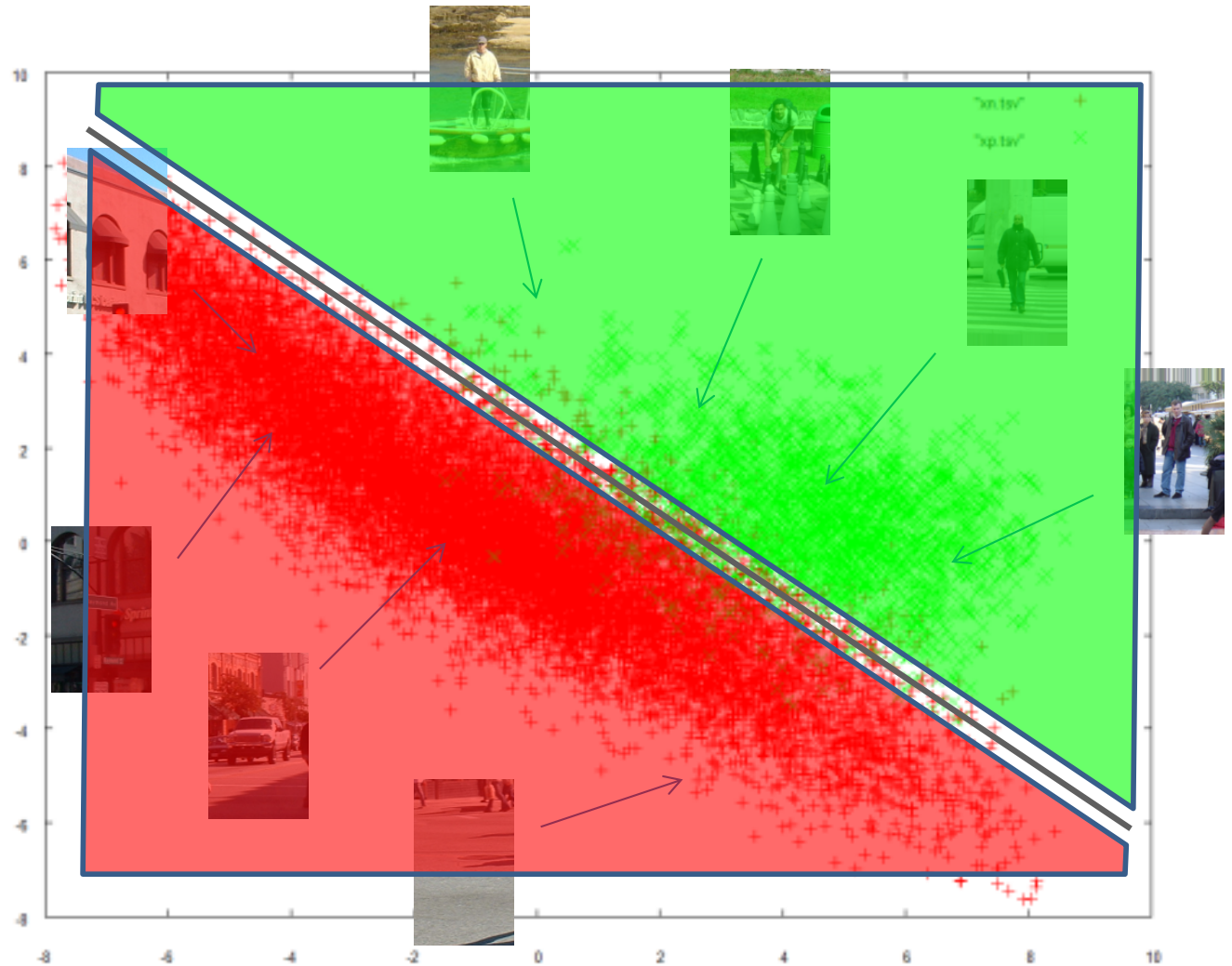
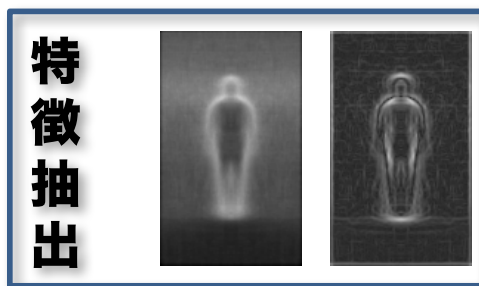


特徴抽出



# 画像認識—人物検出を例に

特徴抽出：「人らしさ」を取り出す写像

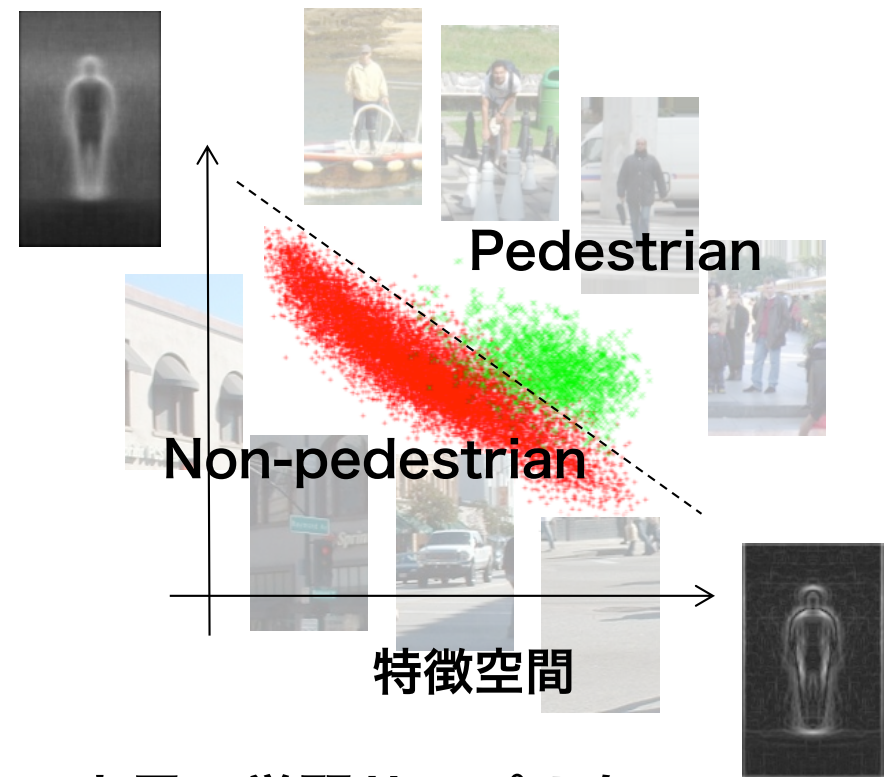
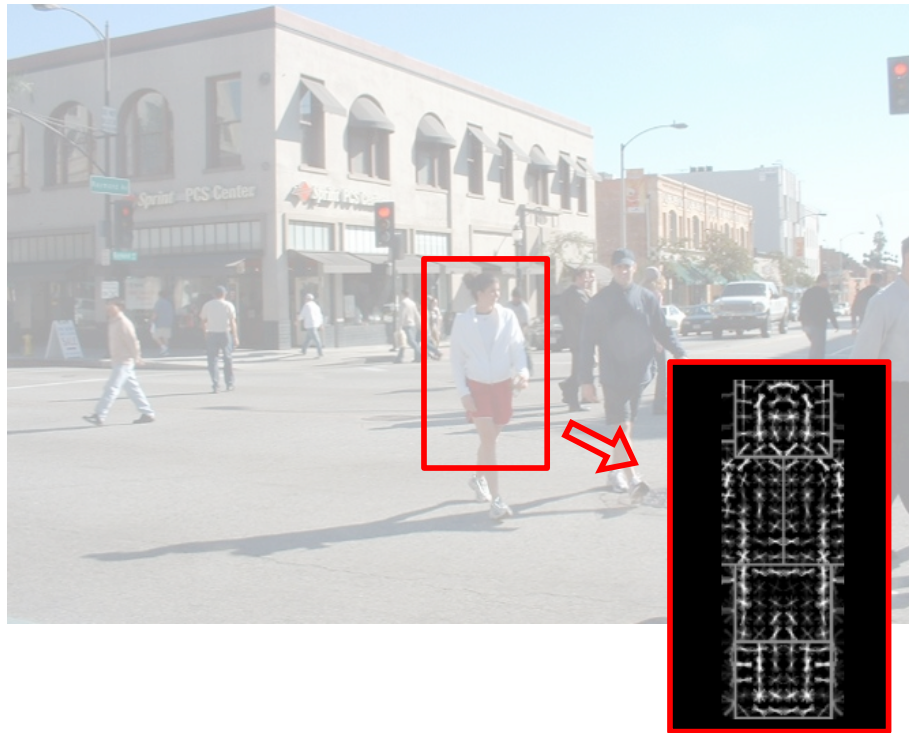


# 画像認識—人物検出を例に





# 画像認識のプロセス



大量の学習サンプルを用いて分類器を学習

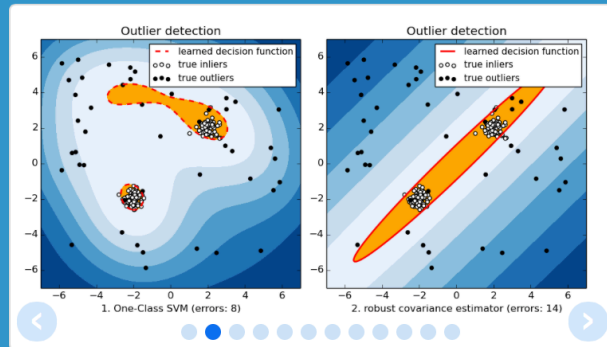
# scikit-learn



[Home](#) [Installation](#) [Documentation](#) ▾ [Examples](#)

Google™ Custom Search

Search ✕



## scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

### Classification

Identifying to which set of categories a new observation belong to.

**Applications:** Spam detection, Image recognition.

**Algorithms:** *SVM*, *nearest neighbors*, *random forest*, ... — Examples

### Regression

Predicting a continuous value for a new example.

**Applications:** Drug response, Stock prices.

**Algorithms:** *SVR*, *ridge regression*, *Lasso*, ... — Examples

### Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** *k-Means*, *spectral clustering*, *mean-shift*, ... — Examples

### Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** *PCA*, *Isomap*, *non-negative matrix factorization*. — Examples

### Model selection

Comparing, validating and choosing parameters and models.

**Goal:** Improved accuracy via parameter tuning

**Modules:** *grid search*, *cross validation*, *metrics*. — Examples

### Preprocessing

Feature extraction and normalization.

**Application:** Transforming input data such as text for use with machine learning algorithms.

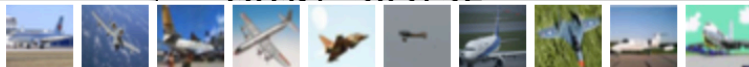
**Modules:** *preprocessing*, *feature extraction*. — Examples

# cifar10：一般物体認識のデータセット

- 物体カテゴリ10，各カテゴリ1000枚，画像サイズ32x32
  - <http://www.cs.toronto.edu/~kriz/cifar.html>
  - 同ページからcifar10 python version (163MB) をダウンロード

```
mbp13:8th okatani$ ls -l cifar-10-batches-py/  
total 363752
```

airplane



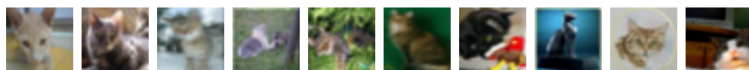
automobile



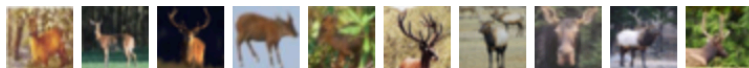
bird



cat



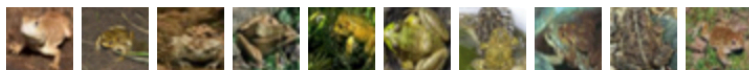
deer



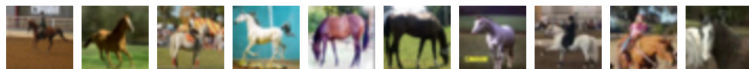
dog



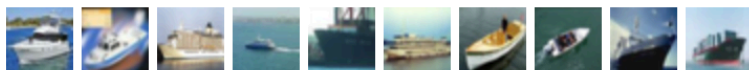
frog



horse



ship



truck



```
aff      158   3  31   2009  batches.meta  
aff 31035704   3  31   2009  data_batch_1  
aff 31035320   3  31   2009  data_batch_2  
aff 31035999   3  31   2009  data_batch_3  
aff 31035696   3  31   2009  data_batch_4  
aff 31035623   3  31   2009  data_batch_5  
aff           88   6   5   2009  readme.html  
aff 31035526   3  31   2009  test_batch
```

# cifar10：一般物体認識のデータセット

- 10000画像ごとに'data\_batch\_x'というファイルに格納
  - pythonのdictionary形式：dataとlabelsに画像とカテゴリーラベルが入っている
  - 以下、読み込み例。画像1枚を取り出して'img.png'にセーブするまで

```
>>> import numpy
>>> import cPickle
>>> fo = open('cifar-10-batches-py/data_batch_1')
>>> dict = cPickle.load(fo)
>>> images = dict['data']
>>> labels = dict['labels']
>>> images.shape
(10000, 3072)
>>> len(labels)
10000
>>> labels[0:10]
[6, 9, 9, 4, 1, 1, 2, 7, 8, 3]
>>> img = numpy.zeros((1024,3), numpy.uint8)
>>> img[:,2] = images[0,0:1024]
>>> img[:,1] = images[0,1024:2048]
>>> img[:,0] = images[0,2048:3072]
>>> import cv2
>>> cv2.imwrite('img.png', img.reshape(32,32,3))
```



# Support vector machines

複数の学習サンプルが所与：  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_l)$

2クラス分類を考える：  $y_i = 1$  or  $-1$

特徴量      ラベル

分類の仕方：  $w^T x_i > 0 \rightarrow \hat{y}_i = 1$

$w^T x_i < 0 \rightarrow \hat{y}_i = -1$

誤分類による損失の尺度：  $-y_i w^T x_i$

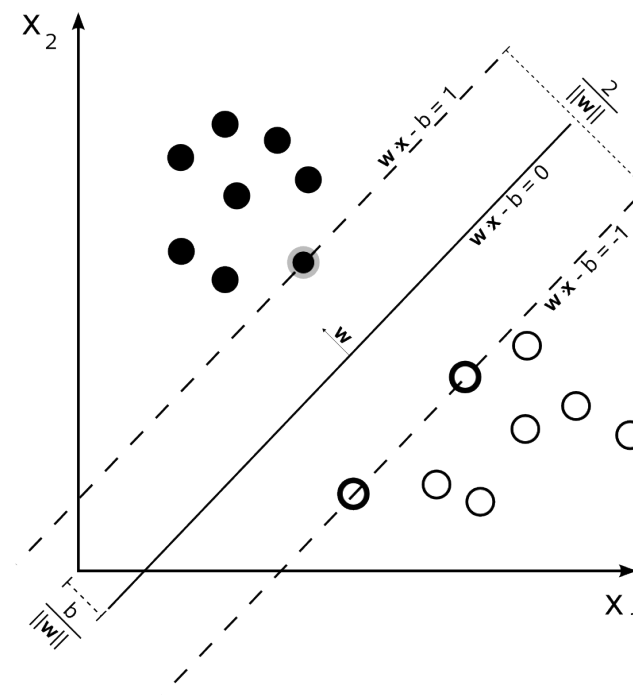
L1-loss

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^l (\max(0, 1 - y_i w^T x_i))$$

L2-loss

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^l (\max(0, 1 - y_i w^T x_i))^2$$

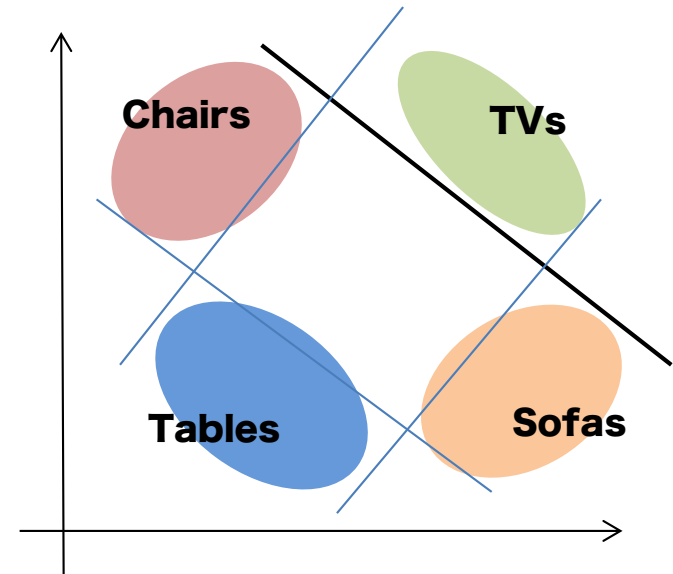
正解を過剰に評価しない



## 2クラス分類器を用いた多クラス分類

- 2クラス分類器をクラス数分用意する 1 対他分類器 (one-versus-the-rest classifier) が一般的
  - 1対1分類器(one-versus-one classifier) もある
- 1. クラス  $k$  の線形分類モデル=クラス  $k$  とそれ以外の全クラスを2分する2クラス分類器  $y(x)$  を学習
- 2. モデルの出力を「スコア」とみなし最大スコアを与えるクラスに分類

$$\operatorname{argmax}_k y_k(\mathbf{x})$$



# 特徴量＝画素値

- data\_batch\_1の10000枚だけで分類をテストしてみる

```
>>> fo = open('cifar-10-batches-py/data_batch_1')
>>> dict = cPickle.load(fo)
>>> images = dict['data']
>>> labels = dict['labels']

>>> from sklearn import svm
>>> index = range(3072)
>>> import random
>>> random.shuffle(index)
>>> selected_pixels = index[0:100]
>>> features = images[:,selected_pixels]
>>> features.shape
(10000, 100)

>>> clf = svm.LinearSVC()
>>> clf.fit(features[:5000,:], labels[:5000])

>>> preds = clf.predict(features[5000:,:])
```

3072個のrgb値のうちランダムに選んだ100個を「特徴量」に

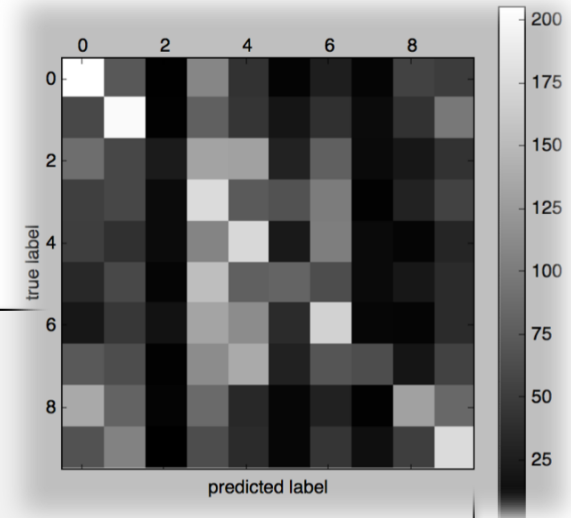
Linear SVMで0-4999番目のサンプルを学習

学習結果を使って5000-9999のサンプルのカテゴリラベルを予測

# 結果の検証

- confusion matrixを使った評価

```
>>> import confmat
>>> confmat.show(labels[5000:], preds)
mean accuracy = x.xx
```



```
import numpy, pylab
from sklearn import metrics

def show(true_labels, pred_labels):
    cm = metrics.confusion_matrix(true_labels, pred_labels)
    print 'mean accuracy = ',
    float(numpy.diag(cm).sum())/len(true_labels)
    pylab.matshow(cm, cmap=pylab.cm.gray)
    pylab.colorbar()
    pylab.xlabel('predicted label')
    pylab.ylabel('true label')
    pylab.show()
```

confmat.py

- 3072個の画素値から選ぶ個数を100から300, 500と変えるとどうなるか？



# 特徴量の正規化

- 特徴空間でのサンプルの分布が平均0, 分散1 になるように正規化

```
>>> from sklearn import preprocessing
>>>
>>> scaler=preprocessing.StandardScaler().fit(features[:5000,:].astype(float))
>>> feat_scaled=scaler.transform(features.astype(float))
>>> feat_scaled[:5000,:].mean(axis=0)[0:6]
array([ 9.14657239e-17,  5.57776048e-17, -8.21342994e-17,
        -1.29452005e-17,  1.38733469e-16, -6.19948537e-17])
```

学習サンプルのみに  
ついて正規化を実行

同じ正規化をテスト  
サンプルにも適用

```
>>> clf.fit(features[:5000,:], labels[:5000])
...
>>> preds = clf.predict(feat_scaled[5000:,:])
>>> confmat.show(labels[5000:],preds)
mean accuracy = 0.2158
```

特徴空間の各軸につ  
いての平均が0に

```
>>> clf.fit(feat_scaled[:5000,:], labels[:5000])
...
>>> preds = clf.predict(feat_scaled[5000:,:])
>>> confmat.show(labels[5000:],preds)
mean accuracy = 0.3306
```

正規化によって正答  
率が上昇

# 特徴量の検討：格子の色統計量

- 画像を格子に分割し，格子内のRGB値の平均と分散を特徴にする
  - 特徴数 = (分割数) x 2 (平均と分散) x 3 (RGB)

```
def grid_mean_std(images, num_divs=8):cifar_features.py  
    feat = numpy.zeros((images.shape[0], num_divs*num_divs*2*3), float)  
    for i in range(images.shape[0]):  
        f = []  
        r = images[i, 0:1024].reshape(32, 32)  
        stride = 32/num_divs  
        for k in range(num_divs):  
            for l in range(num_divs):  
                f.append(r[k*stride:(k+1)*stride, l*stride:(l+1)*stride].mean())  
                f.append(r[k*stride:(k+1)*stride, l*stride:(l+1)*stride].std())  
        g = images[i, 1024:2048].reshape(32, 32)  
        for k in range(num_divs):  
            for l in range(num_divs):  
                f.append(g[k*stride:(k+1)*stride, l*stride:(l+1)*stride].mean())  
                f.append(g[k*stride:(k+1)*stride, l*stride:(l+1)*stride].std())  
        b = images[i, 2048:3072].reshape(32, 32)  
        for k in range(num_divs):  
            for l in range(num_divs):  
                f.append(b[k*stride:(k+1)*stride, l*stride:(l+1)*stride].mean())  
                f.append(b[k*stride:(k+1)*stride, l*stride:(l+1)*stride].std())  
        feat[i, :] = numpy.array(f)  
  
    return feat
```

# 特徴量の検討：格子の色統計量

- 画像を格子に分割し，格子内のRGB値の平均と分散を特徴にする
  - 特徴数 = (分割数) x 2 (平均と分散) x 3 (RGB)

```
>>> feats=cifar_features.grid_mean_std(images, 4)
```

色統計量の特徴量  
を求める

```
>>>
```

```
scaler=preprocessing.StandardScaler().fit(feats[:5000,:].a
```

正規化

```
>>> feats=scaler.transform(feats)
```

学習

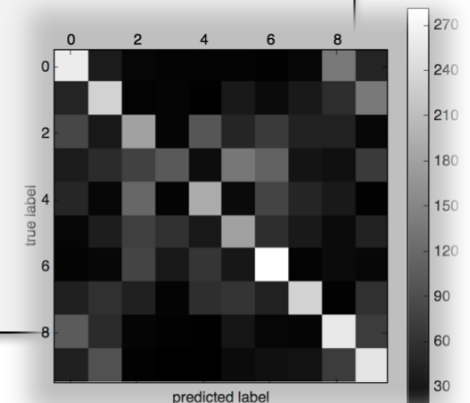
```
>>> clf.fit(feats[:5000],labels[:5000])
```

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,  
          intercept_scaling=1, loss='l2', multi_class='ovr', penalty='l2',  
          random_state=None, tol=0.0001, verbose=0)
```

```
>>> preds=clf.predict(feats[5000:])
```

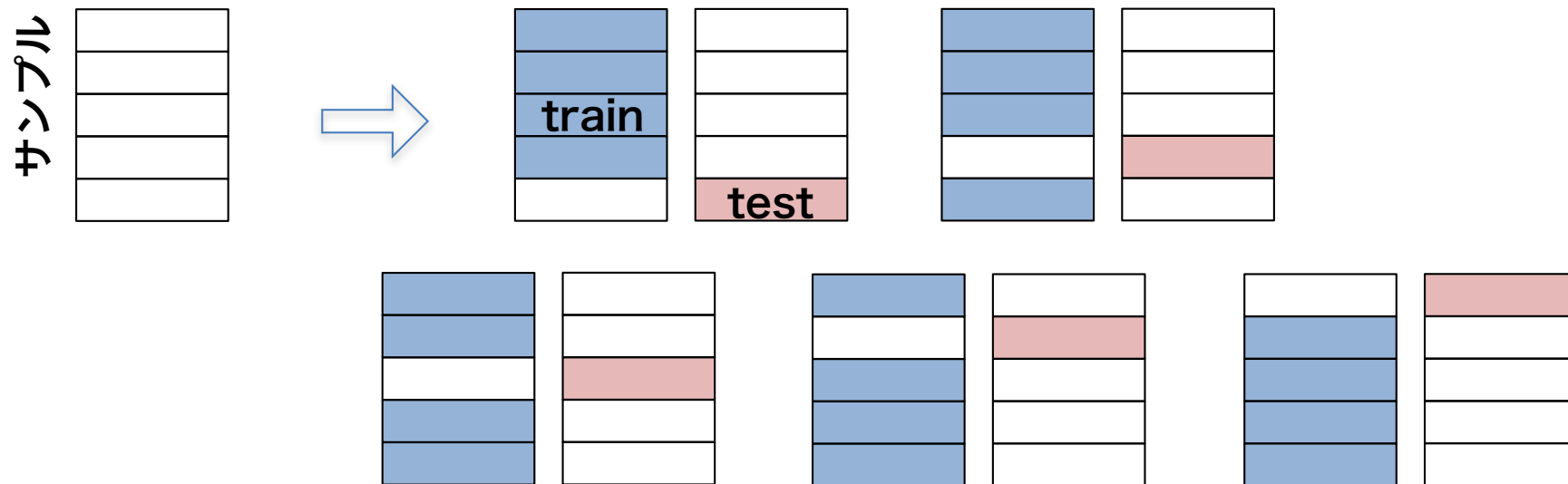
テスト

```
>>> confmat.show(labels[5000:], preds)  
mean accuracy = 0.4254
```



# 交差検定 : cross validation

- サンプルは学習用とテスト用に分けるが、分け方で性能が変化
- 分け方を系統的に変えて性能を調べ、平均的な性能を見積もる
  - 例 : 5-fold cross validation



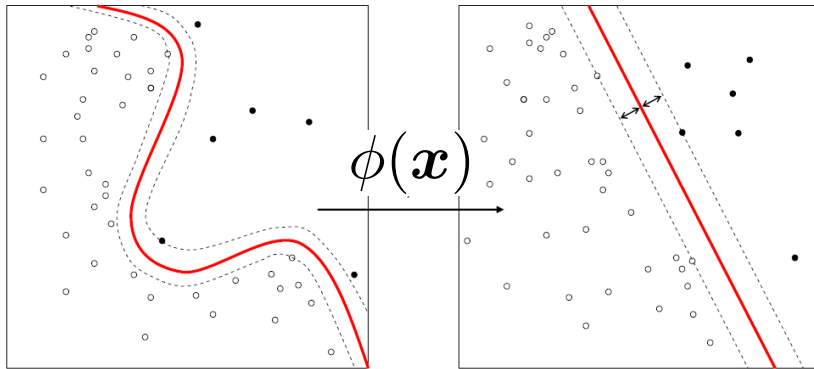
```
>>> score=cross_validation.cross_val_score(clf, feats,
numpy.array(labels,int), cv=5)
>>> score
array([ 0.438 ,  0.412 ,  0.425 ,  0.4215,  0.428 ])
```



# Kernel (Nonlinear) SVM

- 特徴空間を変換する非線形変換 $\phi$ でサンプルを別の空間に投影
- 新しい空間でlinear SVMを学習
- $\phi$ ではなく、その内積であるkernel functionを指定：

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$



- linear:  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$ .
- polynomial:  $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d, \gamma > 0$ .
- radial basis function (RBF):  $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2), \gamma > 0$ .
- sigmoid:  $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r)$ .

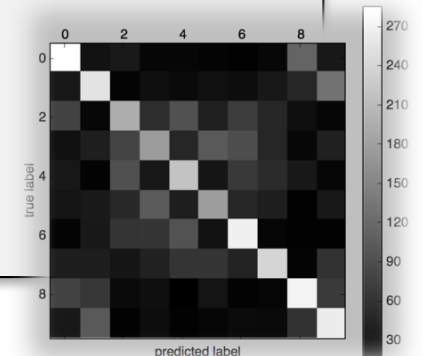
```
>>> clf2=svm.SVC()

>>> clf2.fit(feats[:5000], labels[:5000])
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)

>>> preds=clf2.predict(feats[5000:])

>>> confmat.show(labels[5000:], preds)
mean accuracy = 0.4476
```

1/n\_features



# SVMのパラメータチューニング

- パラメータCは最も重要なもので、多くの場合データごとに最適値は異なる
- カーネルSVMではさらに追加のパラメータがある
- パラメータを少しずつ変えてexhaustive searchを行うのが一般的
  - その際、cross validationを行って各パラメータの性能を正しく見積もる

```
>>> from sklearn import grid_search
>>> params=[{'C':[1e-5,1e-3,1,100,10000]}]
>>> clf=grid_search.GridSearchCV(svm.LinearSVC(), params, cv=2)
>>> clf.fit(feats,labels)
...
>>> clf.grid_scores_
[mean: 0.34770, std: 0.00090, params: {'C': 1e-05}, mean: 0.42450, std:
0.00590, params: {'C': 0.001}, mean: 0.42490, std: 0.00310, params: {'C':
1}, mean: 0.25120, std: 0.01460, params: {'C': 100}, mean: 0.25180, std:
0.03540, params: {'C': 10000}]
```

## レポート2

- 講義と同条件でのcifar10の物体認識で、スライド配布した方法の最良の精度（～45%）を上回る方法を実現し、結果を報告せよ
  - 講義と同条件=data\_batch\_1の10,000サンプルのうち、後半5,000サンプルをテスト（精度検証）に使うということ
  - 方法の例1：特徴量を改善する
  - 方法の例2：分類器（classifier）を変更する
    - sklearn.ensembleのRandomForestClassifier/ExtraTreesClassifierなど
  - 方法の例3：学習サンプルを増やす
    - data\_batch\_2やその他のサンプルを使い（＝学習サンプルを増やして）分類器を学習する・ただしテストサンプルはdata\_batch\_1の後半5,000サンプルとすること
  - 参考：これまでのレコード＝96.53% (2015)
    - ただしdata\_batch\_1,...,6で学習し、test\_batchでテスト