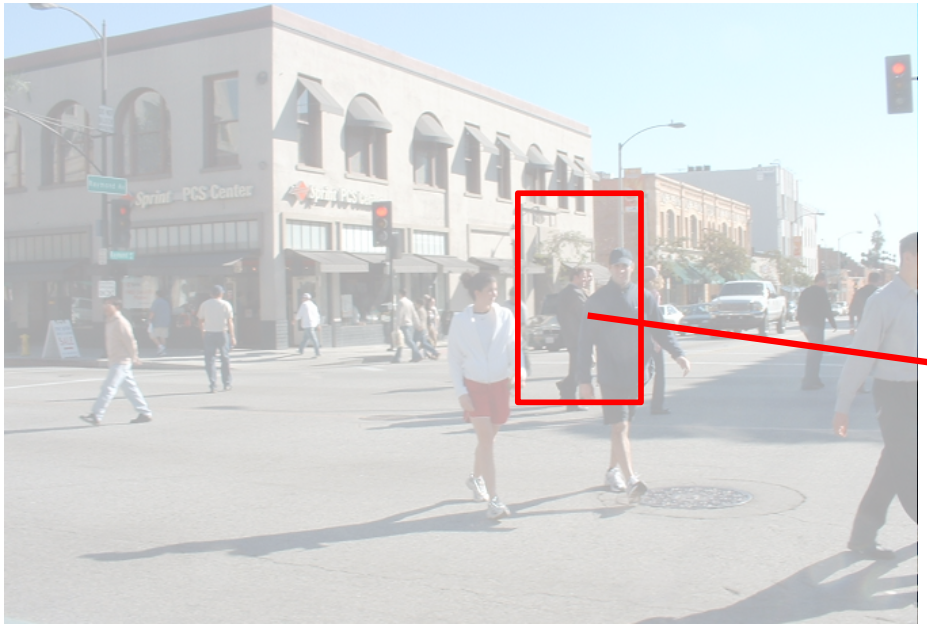
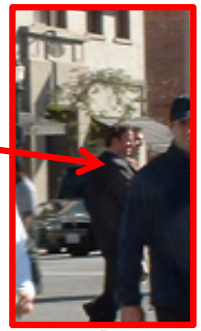
コンピュータビジョン
画像認識

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

認識器

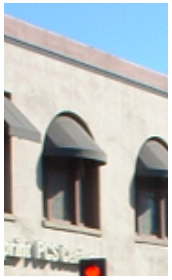
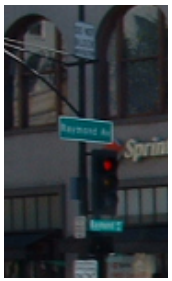
**人か
人でないか**

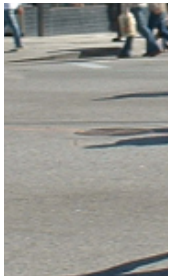
人の画像




…

**人ではない
画像**

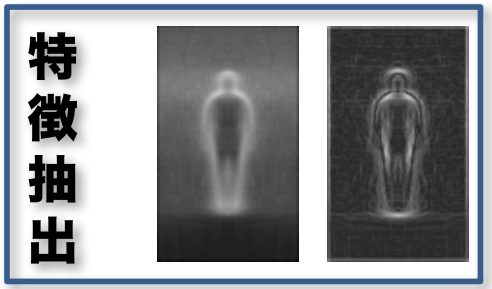



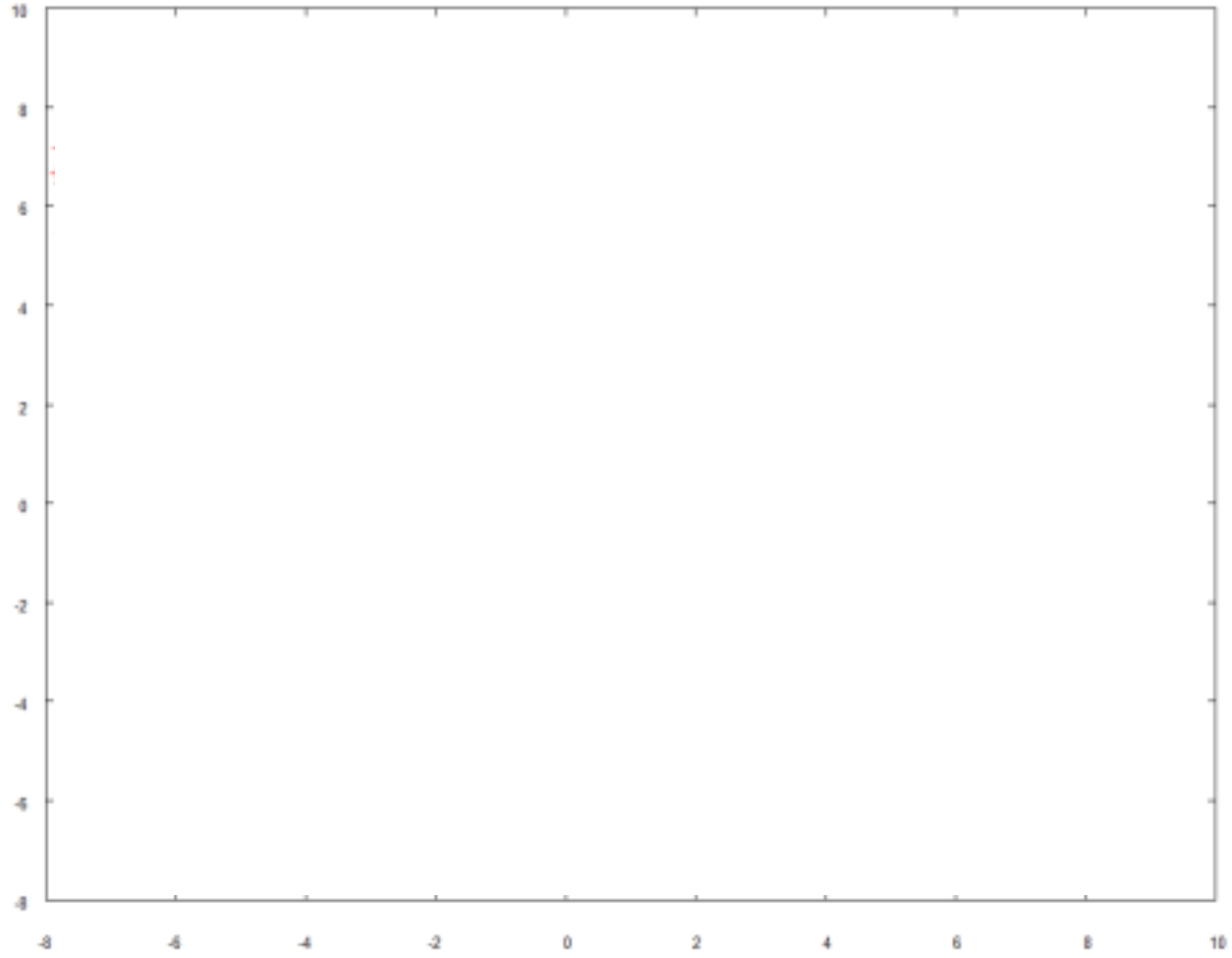


…

画像認識―人物検出を例に

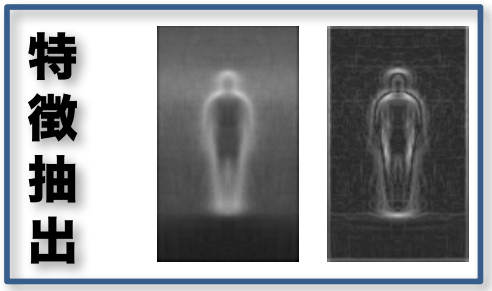
特徴抽出：「人らしさ」を取り出す写像

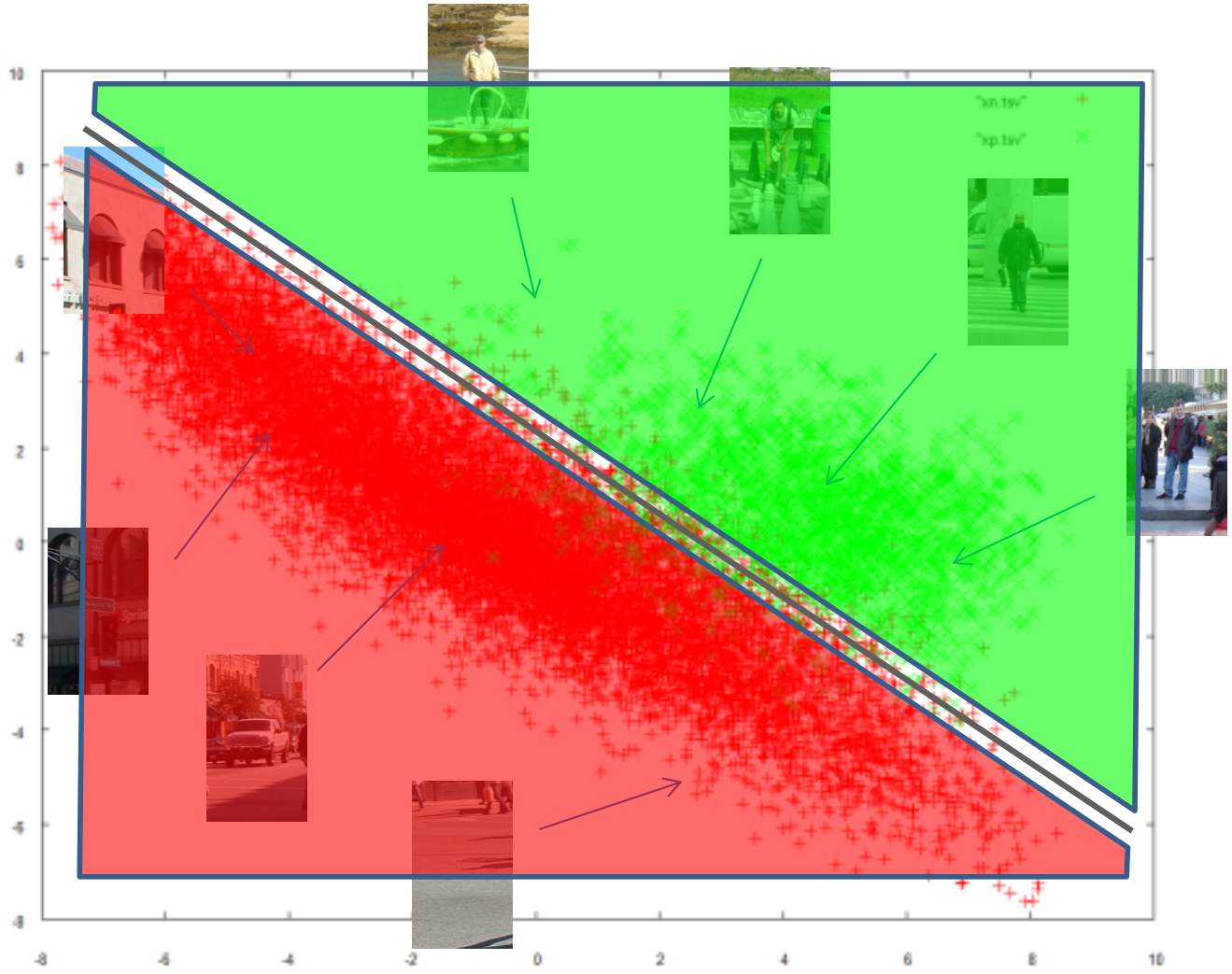




画像認識―人物検出を例に

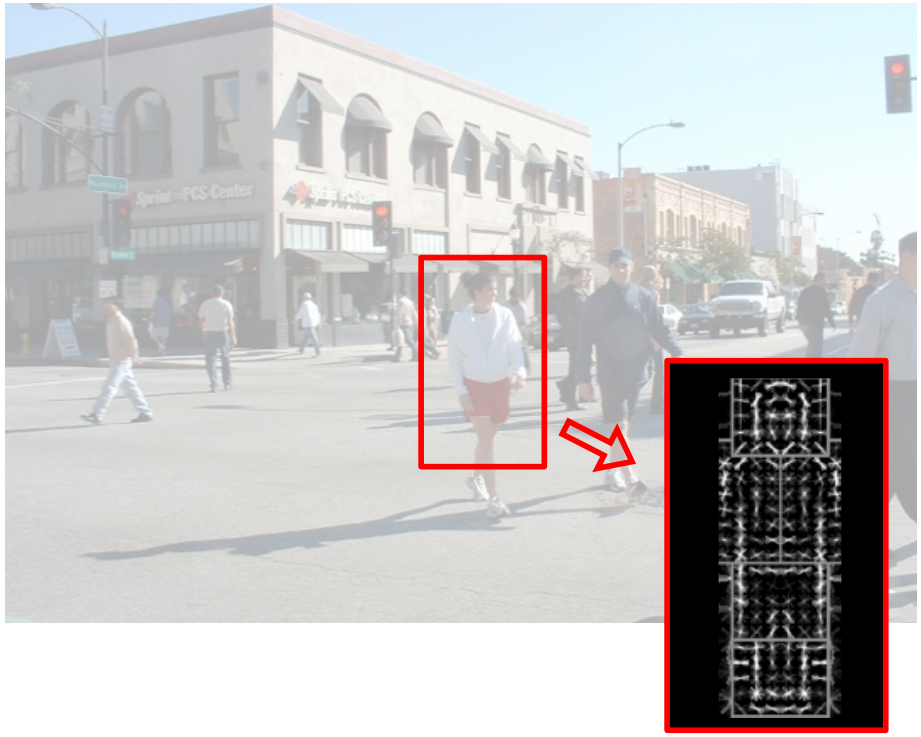
特徴抽出：「人らしさ」を取り出す写像

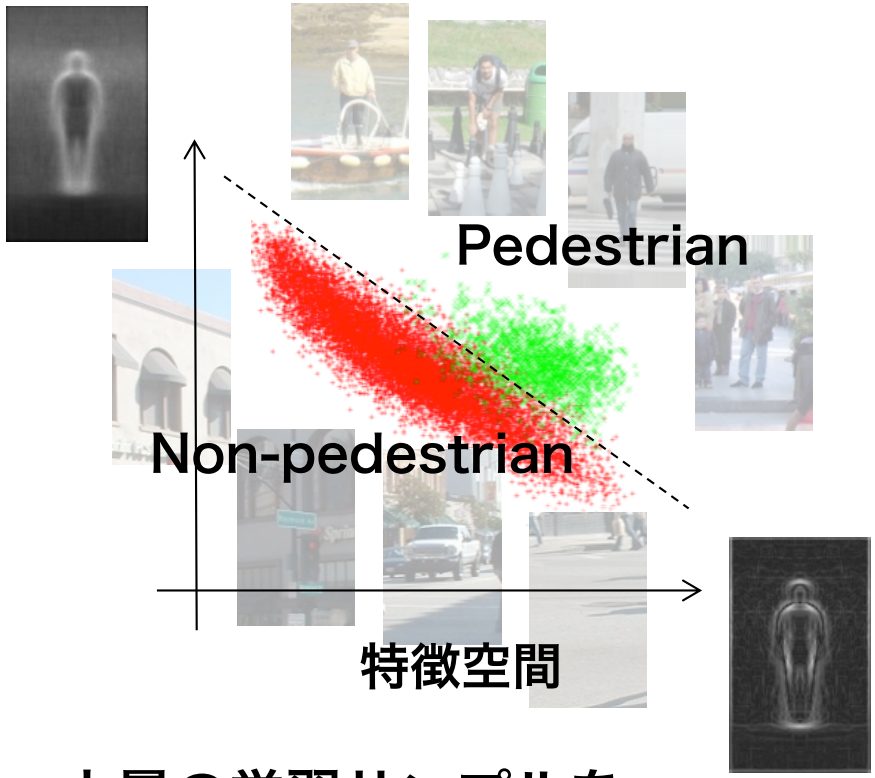




画像認識のプロセス

画像 → 特徴抽出 → 特徴量 → 分類 → 結果





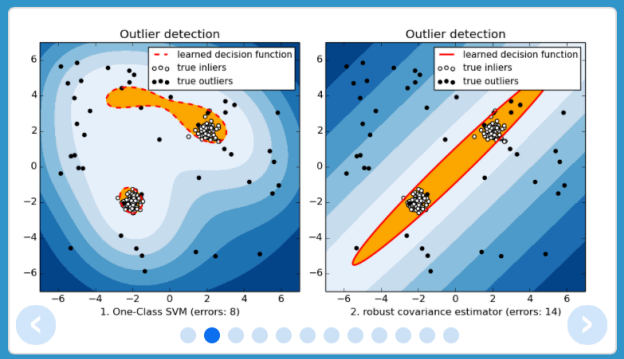
大量の学習サンプルを
用いて分類器を学習

scikit-learn

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*

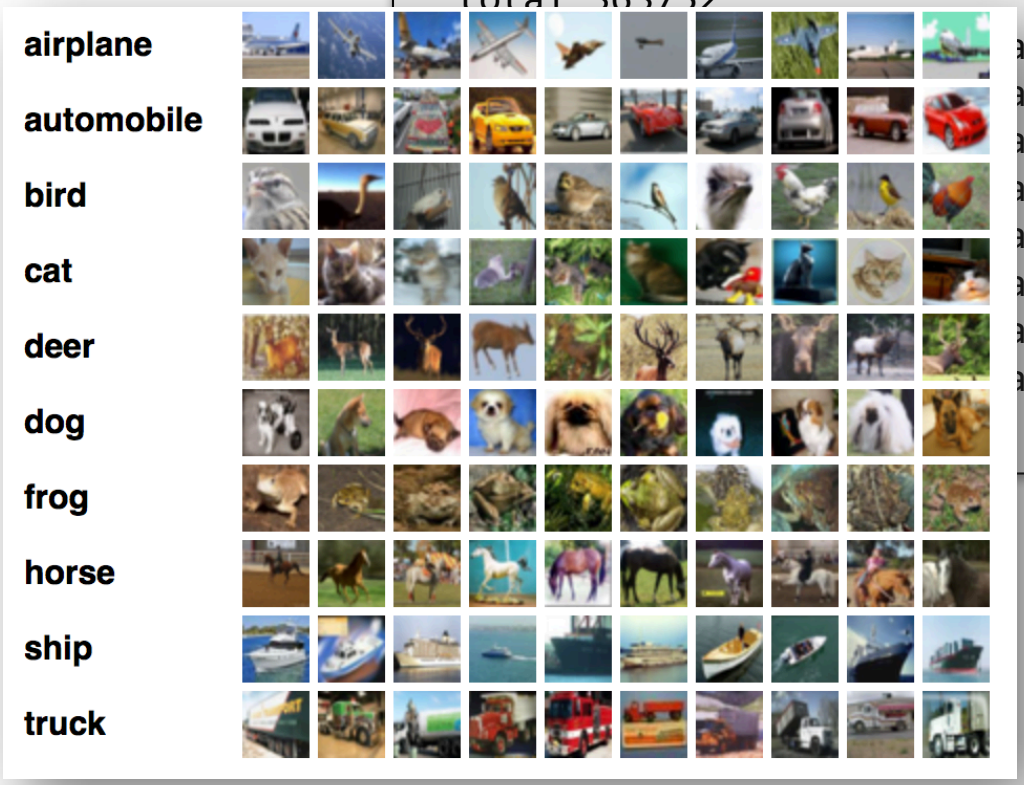
Preprocessing

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
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 _pickle as cPickle
>>> fo = open('cifar-10-batches-py/data_batch_1', 'rb')
>>> dict = cPickle.load(fo, encoding='latin1')
>>> 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))
```

サポートベクタマシン
(support vector machine; SVM)

- ２クラス分類を考える： $d_n = 1 \ \text{or} \ -1$

- 学習サンプル： $(\boldsymbol{x}_1, d_1), (\boldsymbol{x}_2, d_2), \cdots, (\boldsymbol{x}_N, d_N)$

- 分類方法：

$$y(\mathbf{x}) = \begin{cases} 1 & \text{if } u(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

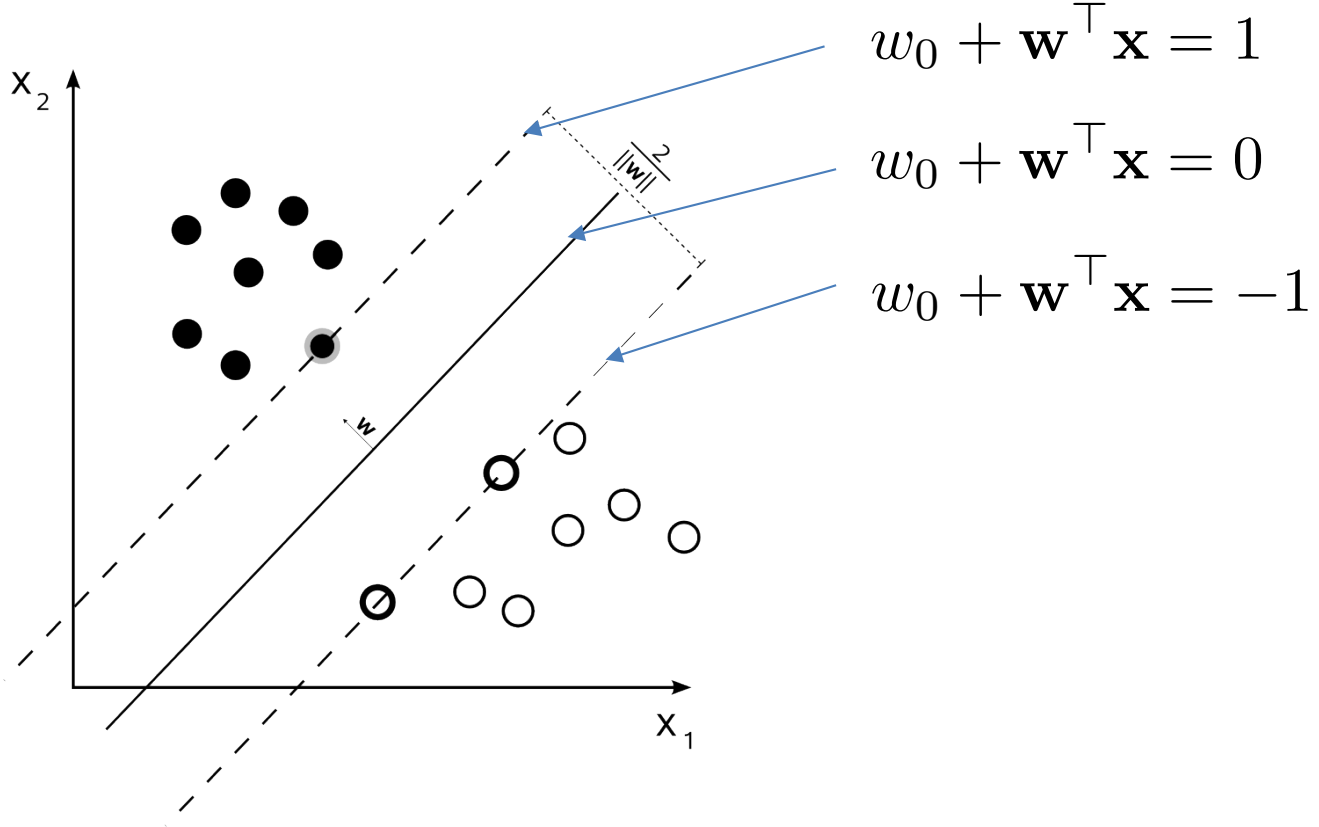
$$u(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_I x_I \ = w_0 + \mathbf{w}^\top \mathbf{x}$$

- 次を最小化する（ハードマージンの場合）：

$$\|\mathbf{w}\| \quad \text{subject to} \quad d_n(w_0 + \mathbf{w}^\top \mathbf{x}) \geq 1$$

サポートベクタマシン
(support vector machine; SVM)

- 線形分離可能であると仮定
- 2クラスを分離する並行な超平面で，それらの間隔が最大となるようなものを見つけたい
- そんな２超平面から等距離にある並行超平面を２クラス境界に選択



サポートベクタマシン
(support vector machine; SVM)

- ソフトマージン（線形分離不可能な場合）

$$E(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\sum_{n}^{N}\left(\max\left(0, 1 - \mathbf{w}^\top \begin{bmatrix}1\\ \mathbf{x}_n\end{bmatrix} d_n\right)\right)^2$$

正解を過剰に評価しない働き

- ハードマージン，ソフトマージンを問わず，常に大域的な最適解を得ることができる
  - 凸関数

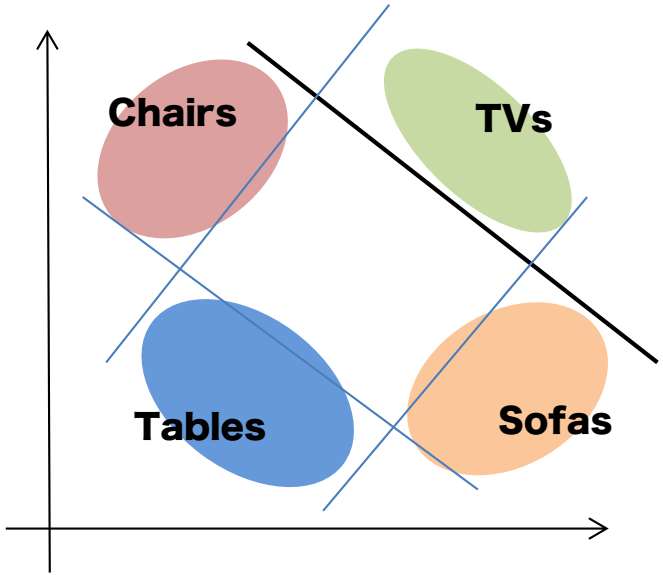
２クラス分類器を用いた多クラス分類

- 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

$$\underset{k}{\operatorname{argmax}}\, y_k(\mathbf{x})$$



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

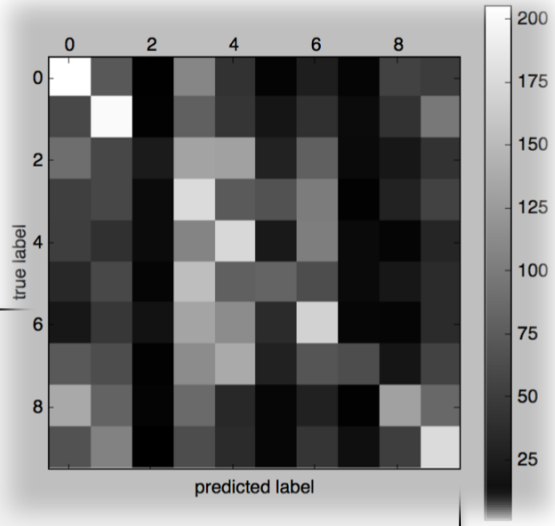
```
>>> fo = open('cifar-10-batches-py/data_batch_1', 'rb')
>>> dict = cPickle.load(fo, encoding='latin1')
>>> images = dict['data']
>>> labels = dict['labels']

>>> from sklearn import svm
>>> index = list(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ーラベルを予測

結果の検証



- confusion matrixを使った評価

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

confmat.py

```
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()
```

  - 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])

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

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

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

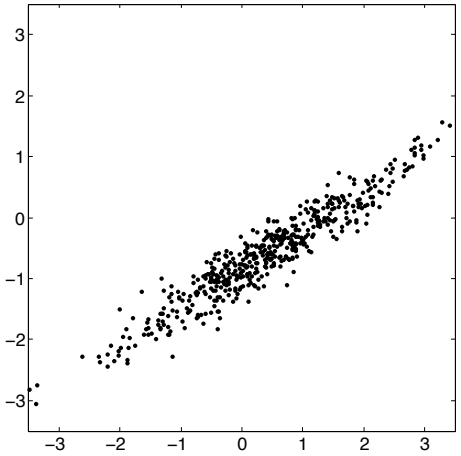
特徴空間の各軸についての平均が０に

正規化によって正答率が上昇

特徴量の正規化（標準化）

$$\mathbf{x}_n = [x_{n1}, x_{n2}, \ldots, x_{nI}]^\top$$

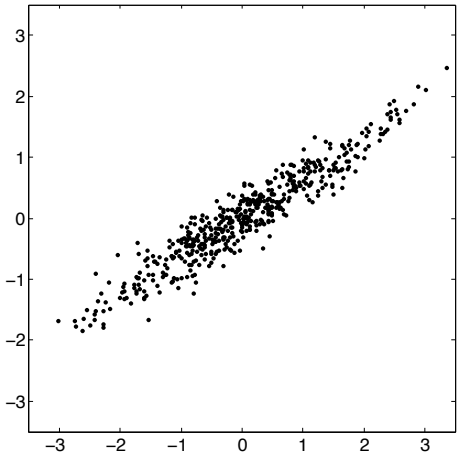
0）入力サンプルの分布



1）平均を差し引く

$$x_{ni} \leftarrow x_{ni} - \bar{x}_i$$

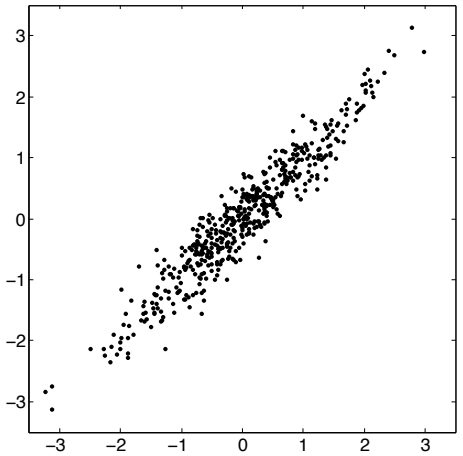
$$\bar{x}_i \equiv \sum_{n=1}^{N} x_{ni}/N$$



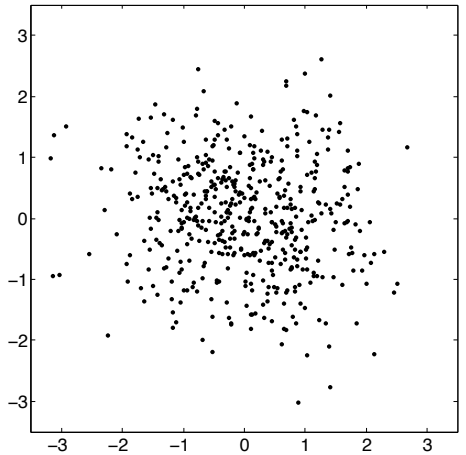
2）標準偏差で割る

$$x_{ni} \leftarrow \frac{x_{ni} - \bar{x}_i}{\sigma_i}$$

$$\sigma_i = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_{ni} - \bar{x}_i)^2}$$



3）白色化成分間の相関をなくす



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）

cifar_features.py

```
def grid_mean_std(images, num_divs=8):
    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）

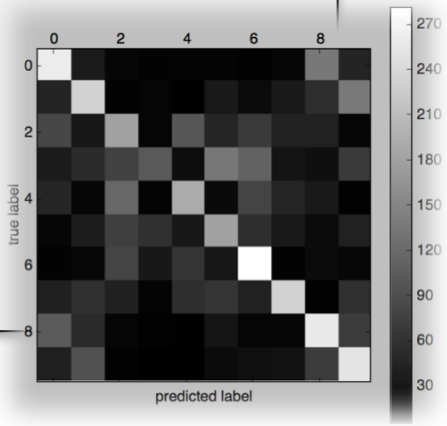
```
>>> import cifar_features
>>> feats=cifar_features.grid_mean_std(images, 4)

>>>
scaler=preprocessing.StandardScaler().fit(feats[:5000,:].astype(float))
>>> feats=scaler.transform(feats)

>>> 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:])

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

色統計量の特徴量を求める

正規化

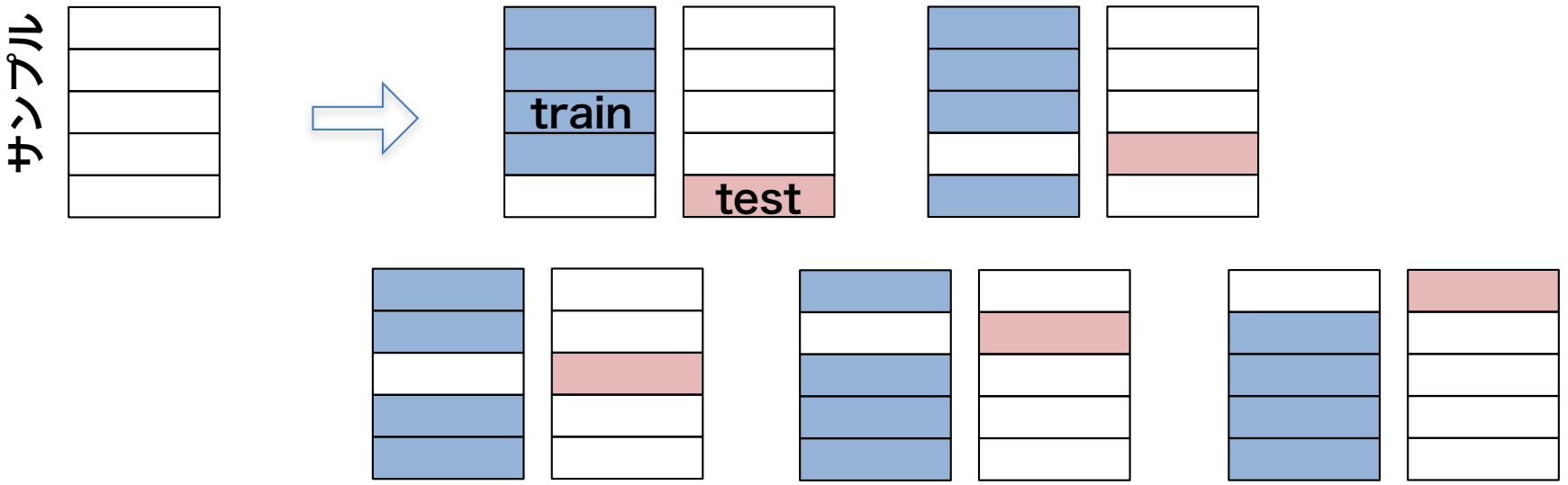
学習

テスト



交差検定：cross validation

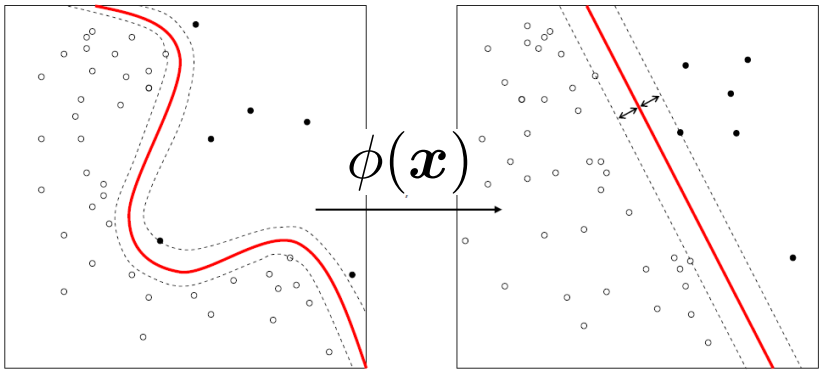
- サンプルは学習用とテスト用に分けるが，分け方で性能が変化
- 分け方を系統的に変えて性能を調べ，平均的な性能を見積もる
  - 例：5-fold cross validation




```
>>> 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Φでサンプルを別の空間に投影
- 新しい空間でlinear SVMを学習
- Φではなく，その内積であるkernel functionを指定：

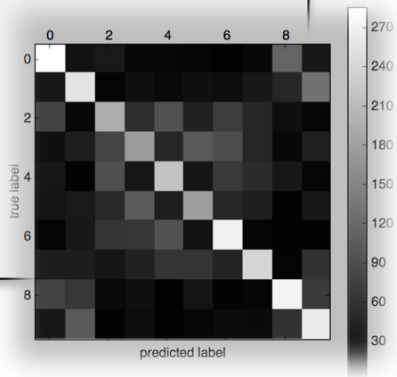


$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$

- 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（8/3まで）

- 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
    - sklean.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