

データサイエンス

第10回

～クラス分類(1)～

情報理工学系研究科
創造情報学専攻
中山 英樹

本日の内容

- クラス分類
 - パターン認識入門
- 各手法の外観・位置づけ
 - 識別的アプローチ
 - 生成的アプローチ

クラス分類（クラス識別）

- クラスタリングとは全く違うので注意
- データから何かを発見したい → 教師なし学習

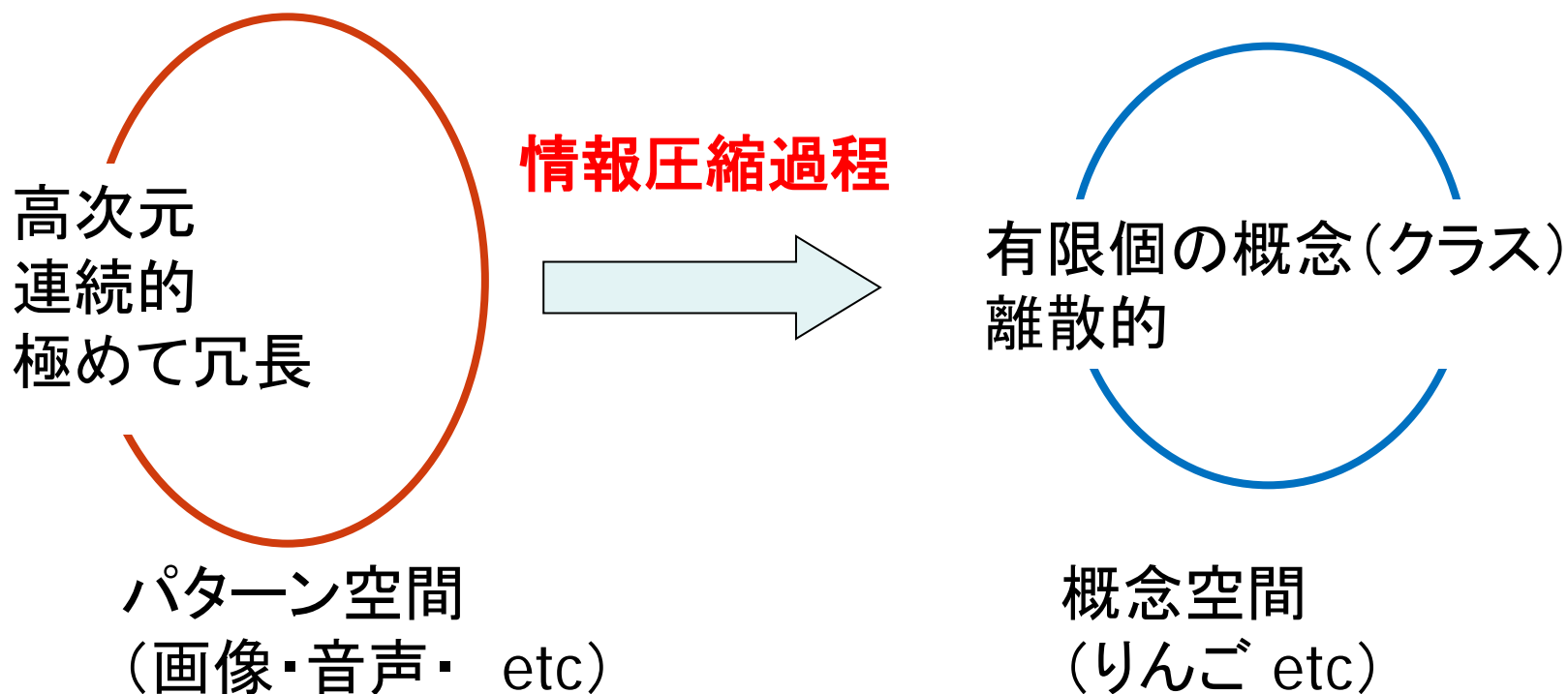
説明変数	手法
量的データ(比尺度)	主成分分析、因子分析、LPP
量的データ(間隔尺度)	クラスター分析、多次元尺度構成法、数量化Ⅳ類
質的データ	数量化Ⅲ類、対応分析

- データを使って何かを予測したい → 教師あり学習

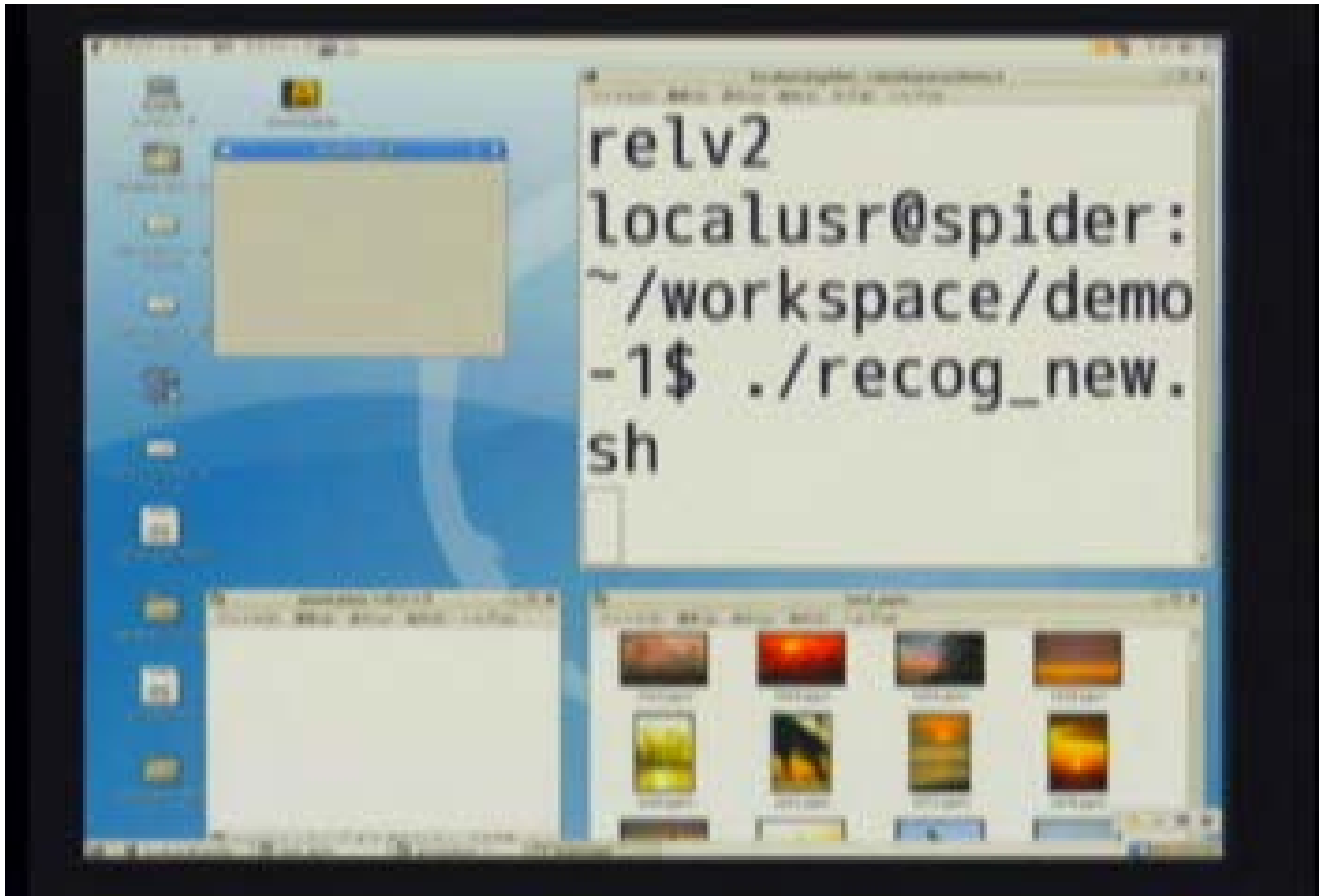
目的変数	説明変数	手法
量的データ	量的データ	回帰分析
	質的データ	数量化Ⅰ類
質的データ	量的データ	判別分析、SVM、kNN...
	質的データ	数量化Ⅱ類

応用：パターン認識

- 実世界の情報（＝パターン情報）を分類する（≡認識する）問題
- 最近は“機械学習”とほぼ同義になっている感があるが、もともとはその限りではない（ルールベースなど）

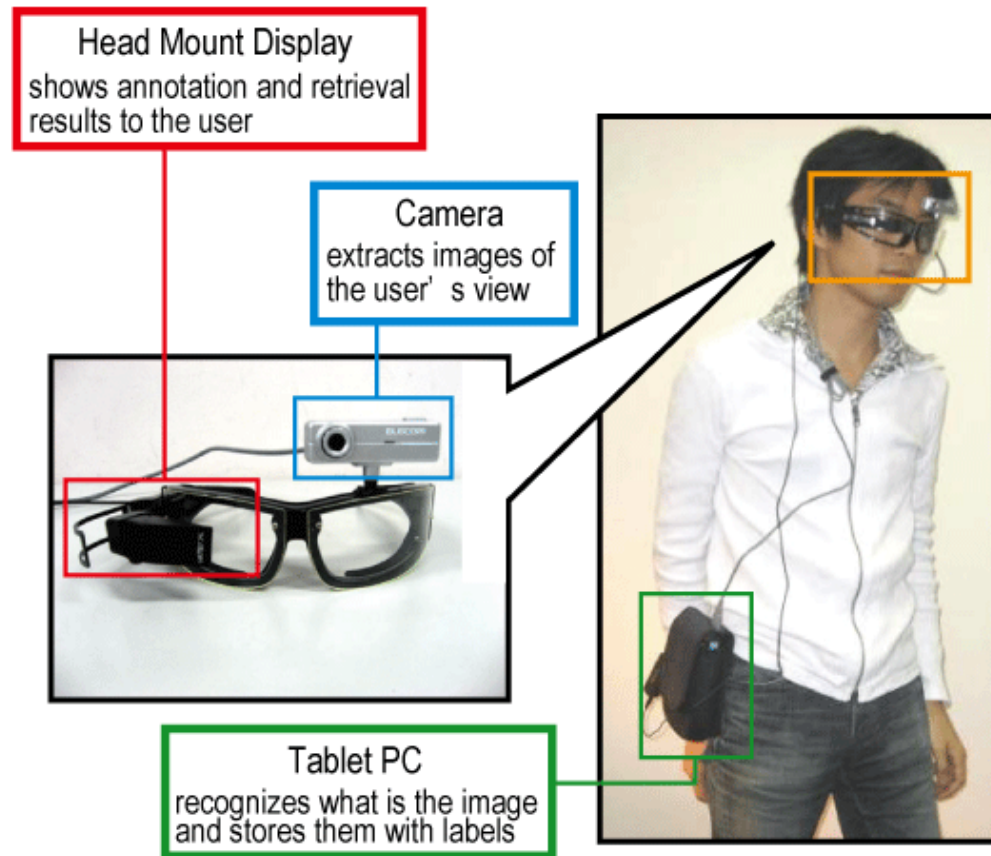


画像認識の例



画像認識の例

- 人工知能ゴーグル
 - 装着者の見たものを認識し、記録し続けるシステム



画像認識の例

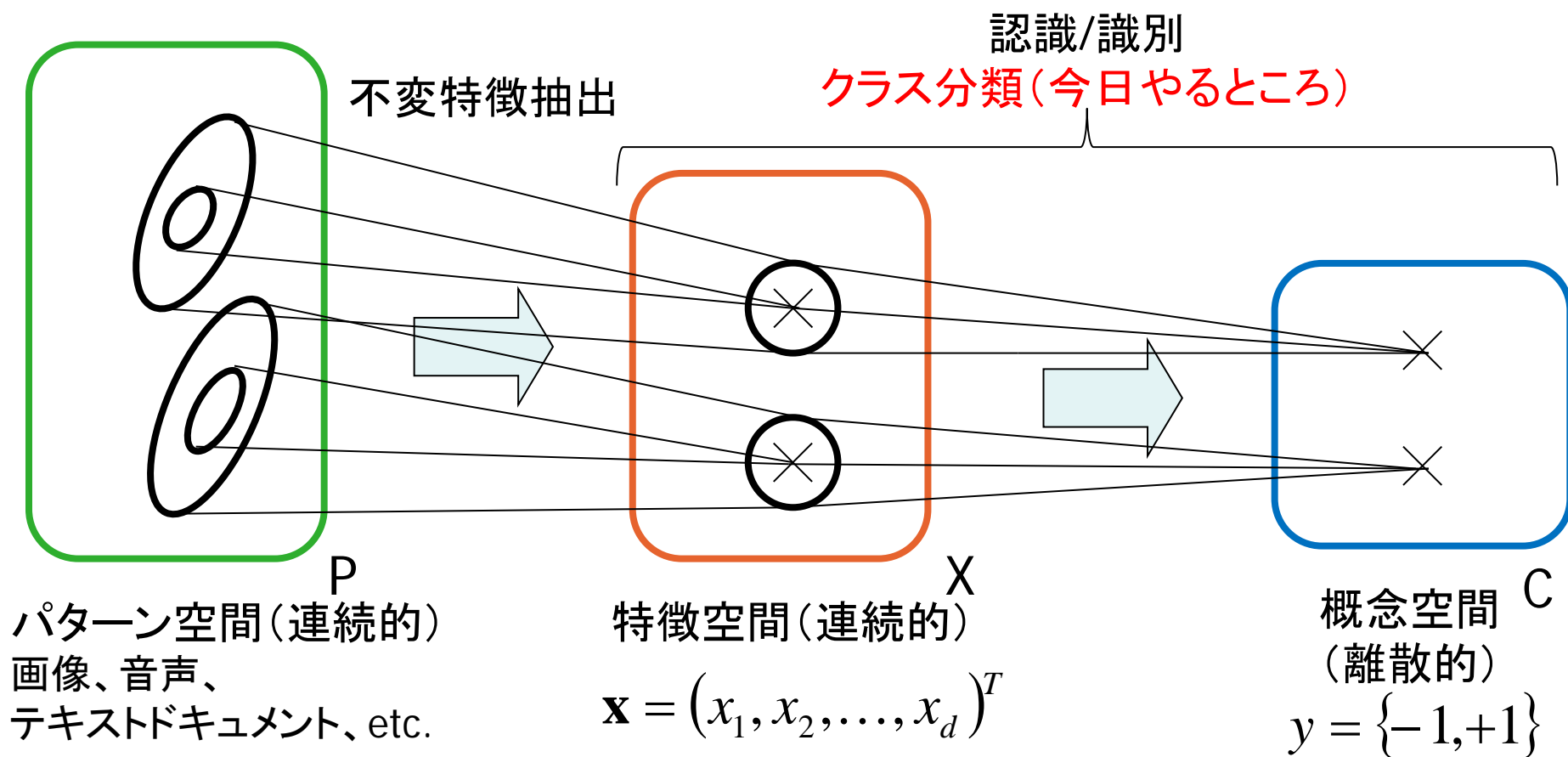




[Fei-Fei et al. CVPR2007 Tutorial]

パターン認識のパイプライン

- よい特徴量（説明変数）を抽出・選択することは極めて重要
 - Garbage in garbage out
 - （ドメイン依存なので今日は扱いませんが…）



準備：ベイズの定理



Thomas Bayes (1702-1761)

- 条件付き確率（事後確率）：
事象Aが起こったもとでBも起こる確率

$$P(B | A) = \frac{P(A, B)}{P(A)} = \frac{P(A | B)P(B)}{P(A)}$$

← Bの事前確率

Bの(Aに対する)事後確率

事後確率・事前確率を相互に書き換えることができる

- 覚え方

$$P(A, B) = \underline{P(B | A)P(A) = P(A | B)P(B)}$$

パターン認識的には…

- パターンの特徴ベクトル \mathbf{x} が与えられた時のクラス C の事後確率が重要

$$P(C | \mathbf{x}) = \frac{P(\mathbf{x} | C)P(C)}{P(\mathbf{x})}$$

- 事後確率を最大とするクラスへ識別
 - 誤識別率を最小にする
 - 誤識別のリスク（ペナルティ）がクラスによらず一定の時、最適な識別境界を与える（ベイズ識別と一致）

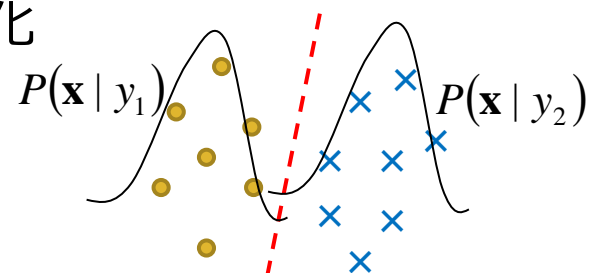
$$\hat{C} = \arg \max_c P(C | \mathbf{x})$$

分類のアプローチ（事後確率の推定方法）

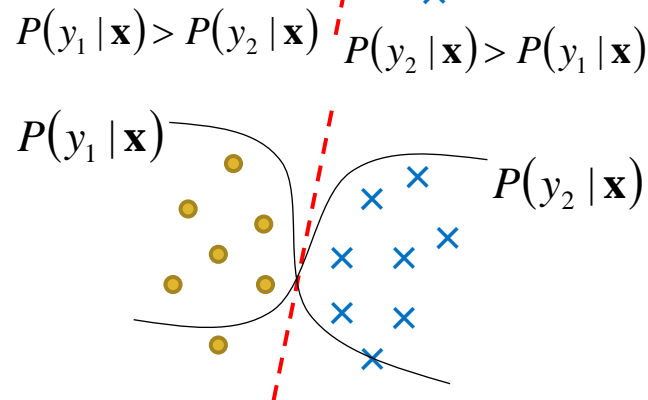
より一般的

- 1. 生成モデル
 - クラスごと条件付き確率と事前確率をモデル化
 - ナイーブベイズ
 - k-最近傍法

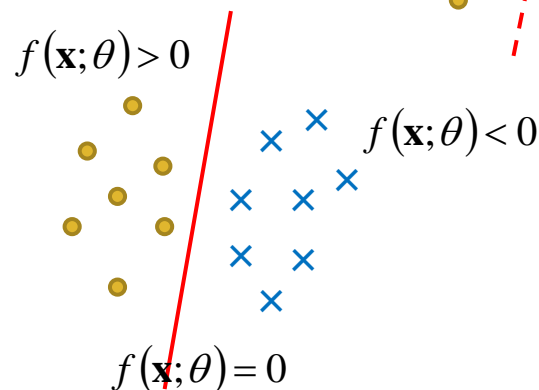
$$P(y | \mathbf{x}) = \frac{P(\mathbf{x} | y)P(y)}{P(\mathbf{x})}$$



- 2. 識別モデル
 - 事後確率を $P(y | \mathbf{x})$ 直接的にモデル化（元の分布はどうでもよい）
 - ロジスティック回帰



- 3. 識別関数
 - 識別の境界面だけモデル化
 - SVM



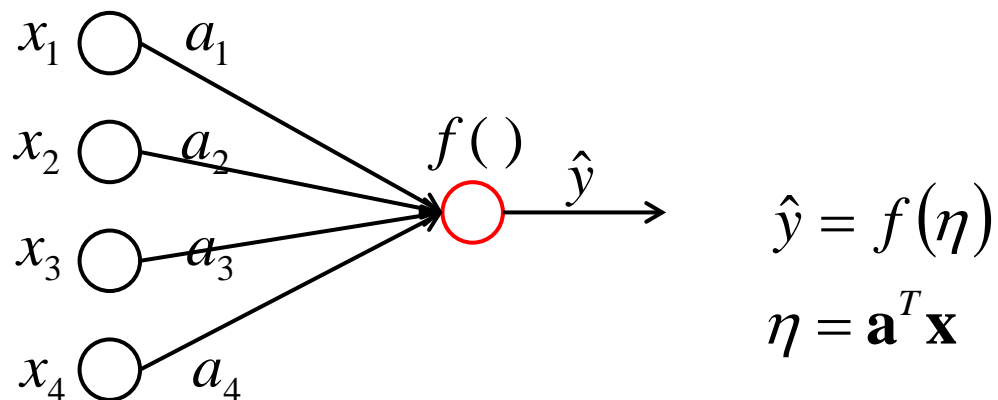
より識別に特化

どちらのアプローチがよい？

- 一概には言えない
 - 識別的アプローチのメリット
 - 閉じた世界（例えばベンチマークデータセット）における識別タスクでは性能が良いことが多い
 - 識別精度、計算コスト、メモリコスト、etc.
 - パラメータ数が比較的少なく済む（→ 省サンプル、低計算コスト）
 - 生成的アプローチのメリット
 - タスクに関して何らかの事前知識を有している場合に利用できる
 - 未知のクラスに対処できる
- 「ある問題を解くとき、その途中段階で難しい問題を解かず、できるだけその問題を直接解くべきである」 by V. Vapnik

識別的アプローチ：線型識別関数&識別モデル

- 要は単純パーセプトロン
 - 活性化関数 f と、誤差をどう考えるかで異なってくる



$$f(\eta) = \eta \quad \rightarrow \text{線形回帰と等価}$$

$$f(\eta) = \frac{1}{1 + \exp(-\eta)} \quad \rightarrow \text{ロジスティック回帰と等価}$$

単純パーセプトロンの学習

- 二乗誤差最小化の場合

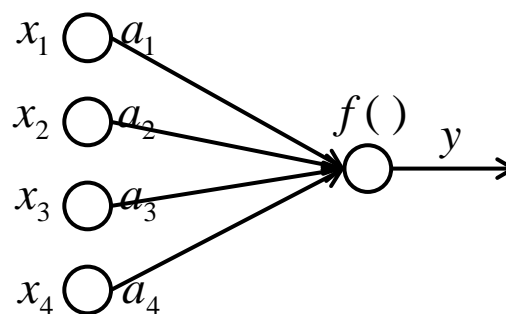
$$\varepsilon^2 = \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \sum_{i=1}^N (y_i - f(\mathbf{a}^T \mathbf{x}_i))^2$$

$$\frac{\partial \varepsilon^2}{\partial a_k} = -2 \sum_{i=1}^N (y_i - \hat{y}_i) x_{ik}$$

なので、最急降下法による更新式は、

$$a_k \leftarrow a_k + \alpha \left(\sum_{i=1}^N (y_i - \hat{y}_k) x_{ik} \right)$$

Widrow-Hoffの学習規則



※ $f(\eta) = \eta$ の時は、線形回帰と同じ解析解が求まる

線型識別関数

- 以下、2カテゴリの分類問題を扱う
 - 出力 (目的変数)は $y = \pm 1$ の二値とする

$$\hat{y} = \begin{cases} 1 & \mathbf{a}^T \mathbf{x} > 0 \\ -1 & \mathbf{a}^T \mathbf{x} < 0 \end{cases}$$

符号しか使わない

- 多クラス(N クラス)の分類は、2クラス分類の線形識別関数の組み合わせで実現可能 (詳しくは次回)
 - One-versus-all : あるクラスと、それ以外のクラス全てを識別する関数 N 個のうち、最も高い出力を出した識別器の結果をとる
 - One-versus-one : 全ての2クラス識別器 $_N C_2$ 個の多数決結果

最小二乗識別

- 最も簡単な方法（解析解）
- 線形重回帰分析において、目的変数にダミー変数を導入しただけ
- 最小二乗線形判別ともよばれる $\hat{\mathbf{a}}_{LS} = (XX^T)^{-1} X \mathbf{y}$
- 意外と馬鹿にならない
 - 2クラス識別の場合は、フィッシャー判別分析による識別と等価
 - 他クラス識別（各クラスのダミー変数を0/1で与える）の場合、クラス外分散を固定し、クラス内分散を最小化する線形射影となる

損失関数の議論

- 準備：マージン $m_i = (\mathbf{a}^T \mathbf{x}_i) y_i$
 - 値が大きいほど、正しい側へ余裕をもって識別されていることを示す
- 二乗誤差 (L2損失) は、識別問題における精度の指標として適切か？

$$J_{LS}(\mathbf{a}) = \sum_{i=1}^N (y_i - \mathbf{a}^T \mathbf{x}_i)^2 = \sum_{i=1}^N y_i^2 \left(1 - \frac{\mathbf{a}^T \mathbf{x}_i}{y_i} \right)^2 = \sum_{i=1}^N (1 - (\mathbf{a}^T \mathbf{x}_i) y_i)^2 = \sum_{i=1}^N (1 - m)^2$$

$y_i = \pm 1$

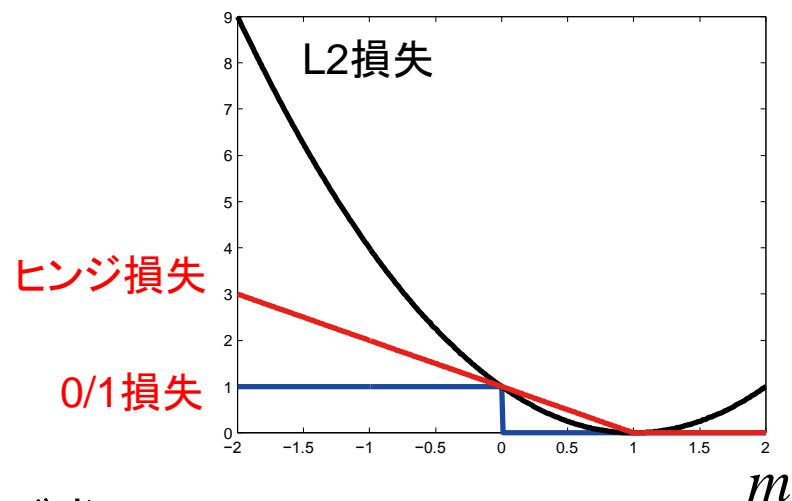
損失関数の議論

- 理想的には**0/1損失**が望ましい
 - 重要なのは出力の符号だけ

$$J_{0/1}(\mathbf{a}) = \frac{1}{2} \sum_{i=1}^N (1 - \text{sign}(m_i))$$

- しかし、凸な関数でないので最適解が求まらない！
- 仕方ないので、解ける範囲で近似
⇒ **ヒンジ損失**

$$J_{\text{Hinge}}(\mathbf{a}) = \sum_{i=1}^N \max(0, 1 - m_i)$$



Support Vector Machine (SVM)

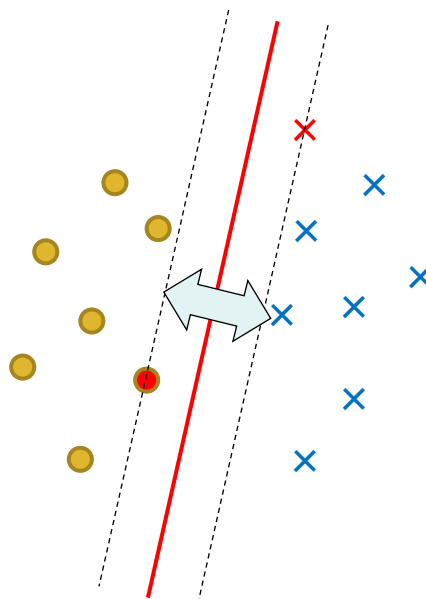
- 現在、最も標準的に用いられる識別器の一つ
- ヒンジ損失 + L2正則化

$$\hat{\mathbf{a}} = \arg \min_{\mathbf{a}} \left[\sum_{i=1}^n \max(0, 1 - (\mathbf{a}^T \mathbf{x}_i) y_i) + \lambda \|\mathbf{a}\|^2 \right]$$

- 経験的に、さまざまなタスクで優れた汎化性能を有する
(とされる)
- ライブラリ多数 (libsvm等)

SVM: マージン最大化からの導出

- こちらの方が一般的な導出
- 最も識別境界に近いサンプル（=サポートベクター）の
マージンが最大となるように線を引く
- 次回もうちょっと詳しくやります



ロジスティック回帰

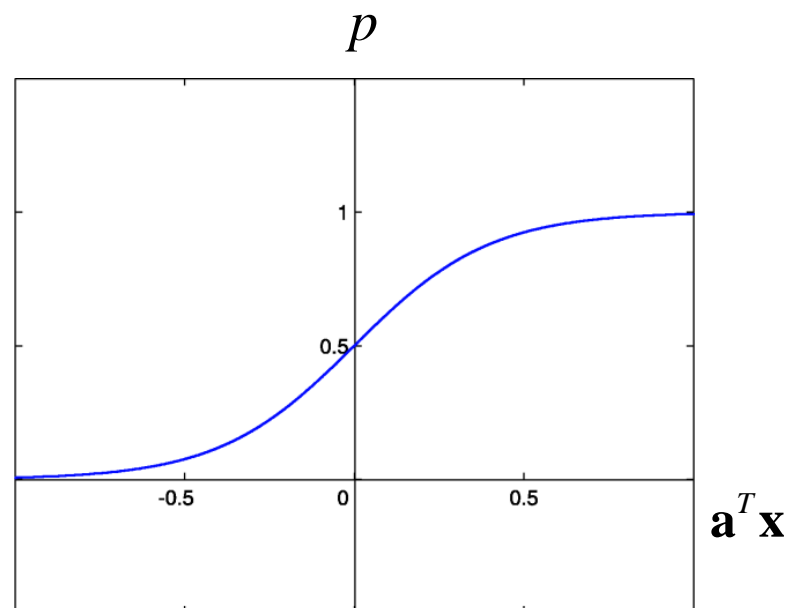
- 事後確率を直接推定（= 識別モデル）
- 二値データ（質的データ）の回帰

$$P(y = 1 | \mathbf{x}) = p$$

$$P(y = -1 | \mathbf{x}_i) = 1 - p$$

$$p = \frac{\exp(\mathbf{a}^T \mathbf{x})}{1 + \exp(\mathbf{a}^T \mathbf{x})}$$

(ロジスティック関数)



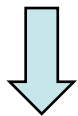
最尤推定による解き方

- 最尤推定

訓練データ集合 $\{\mathbf{x}_i, t_i\}, t_i \in \{0, 1\}$

尤度関数は $L = \prod_{i=1}^N p_i^{y_i} \{1 - p_i\}^{1-y_i} \quad p_i = \frac{\exp(\mathbf{a}^T \mathbf{x}_i)}{1 + \exp(\mathbf{a}^T \mathbf{x}_i)}$

負の対数尤度は $E(\mathbf{a}) = -\ln L = \sum_{i=1}^N \{y_i \ln p_i + (1 - y_i) \ln(1 - p_i)\}$



$$\frac{E(\mathbf{a})}{\partial \mathbf{a}} = \sum_{i=1}^N \underline{(p_i - y_i) \mathbf{x}_i}$$

エラーに説明変数をかけたもの

損失関数の観点からの解釈

- 負の対数尤度を整理すると

$$\begin{aligned} E(\mathbf{a}) &= \sum_{i=1}^N \{ y_i \ln p_i + (1 - y_i) \ln(1 - p_i) \} \\ &= \sum_{i=1}^N \ln \{ 1 + \exp(-y_i \mathbf{a}^T \mathbf{x}_i) \} = \sum_{i=1}^N \ln \{ 1 + \exp(-m_i) \} \end{aligned}$$

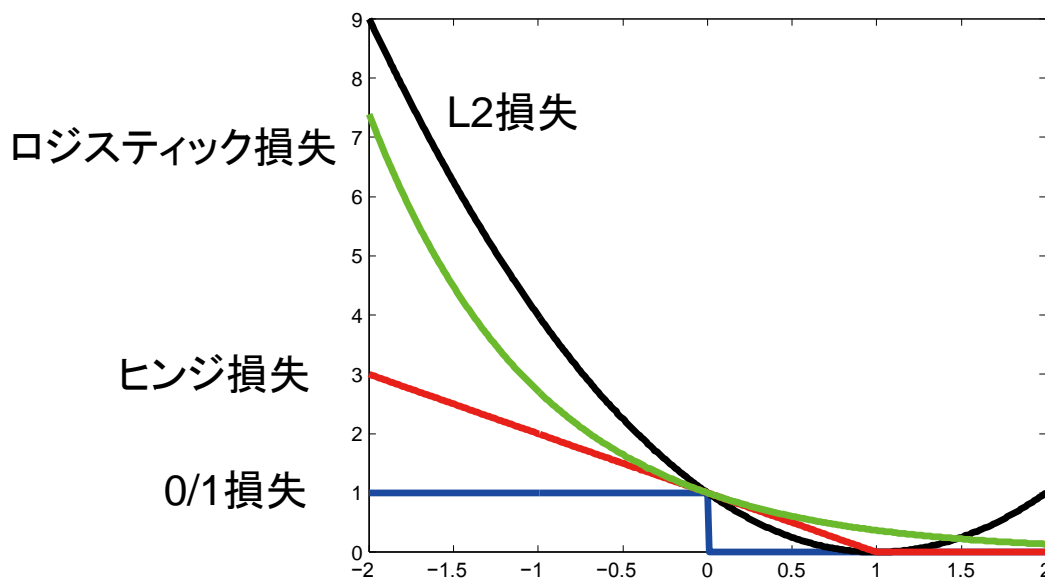
- つまり、対数尤度最大化規準は

$$\arg \min_{\mathbf{a}} \sum_{i=1}^N \ln(1 + \exp(-m_i))$$

と書けることから、**ロジスティック損失** $\ln(1 + \exp(-m))$ を最小化する識別関数の学習を行っていることが分かる

識別的アプローチ（線形モデル）

- 大枠としては、どれもパーセプトロン
- 損失の測り方が違う



- 実用的には
 - とりあえず、SVM、ロジスティック回帰を試してみることが多い
 - 正則化などは、回帰の章と同様のテクニックが導入可能

生成的アプローチ

- 事後確率分布だけでなく、同時確率分布までモデル化

$$\underline{P(C | \mathbf{x})} \propto \underline{P(\mathbf{x} | C)} \underline{P(C)}$$

事後確率 条件付き確率 事前確率

- クラスの事前確率は、何らかの先見知識がある場合を除き、単純にサンプルの割合で推定することが多い

$$\hat{P}(C) = \frac{N_C}{N}$$

- 条件付き確率（生成モデル）の推定がポイント
例）正規分布を用い、最尤推定する（パラメトリック）

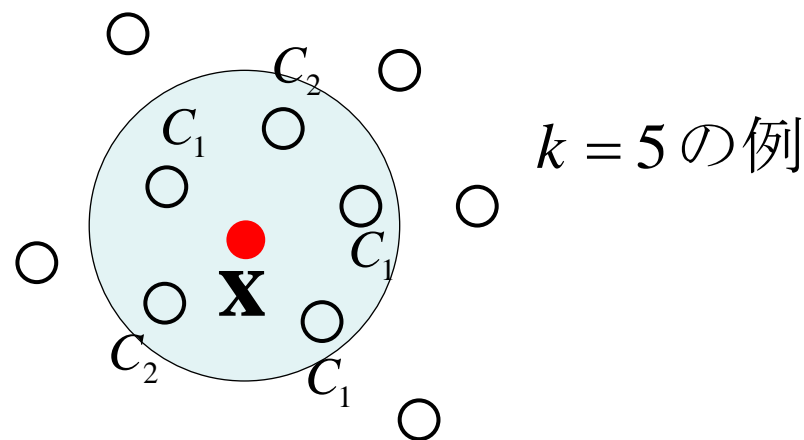
$$\hat{P}(\mathbf{x} | C) = N(\mathbf{x}; \hat{\mu}_C, \hat{\Sigma}_C) \quad \hat{\mu}_C, \hat{\Sigma}_C \text{ は最尤推定量、すなわちクラス } C \text{ のサンプルの平均と分散}$$

パラメトリックなモデルによる生成的分類

- 正規分布によるモデル化の場合
 - 各クラスの共分散パラメータを一定と仮定した場合、線形判別分析と同じ識別境界が得られる
- より複雑なモデルは、実際はあまり用いられない
 - パラメータが多いといろいろ大変
 - 計算コストが膨大
 - 学習サンプルも大量に必要
 - 推定自体困難
 - ...

K-最近傍法 (K-nearest neighbor, K-NN)

- 識別則は非常にシンプル
 - パターン入力 x について、最も近い上位 K 個の学習サンプルの中で、最も多い数のサンプルが所属するクラスへ x を識別



- 直感的には
 - 入力に類似している学習サンプルの多数決
 - サンプルが増えると精度は上がるが、識別のコストは非常に大きくなる

K-NN:確率的な解釈

- クラス C_k に属する学習サンプル数を N_k , 全学習サンプル数を $N = \sum N_k$ とする
- 入力 \mathbf{x} を中心とし、 \mathbf{x} の最近傍 K 点を含む超球の体積を V とする
- 超球に含まれる C_k のサンプル数を K_k とする

超球の内部（局所領域）
については以下のように近似できる

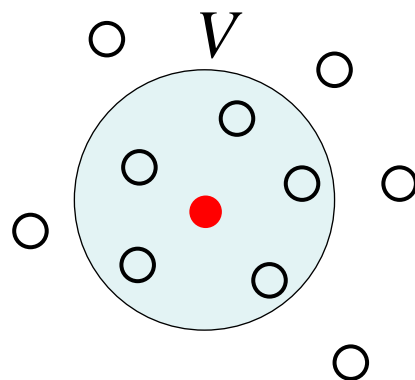
$$P(\mathbf{x} | C_k) = \frac{K_k}{N_k V}$$

$$P(\mathbf{x}) = \frac{K}{NV}$$

$$P(C_k) = \frac{N_k}{N}$$

$$\therefore P(C_k | \mathbf{x}) = \frac{P(\mathbf{x} | C_k) P(C_k)}{P(\mathbf{x})} \cong \frac{K_k}{K}$$

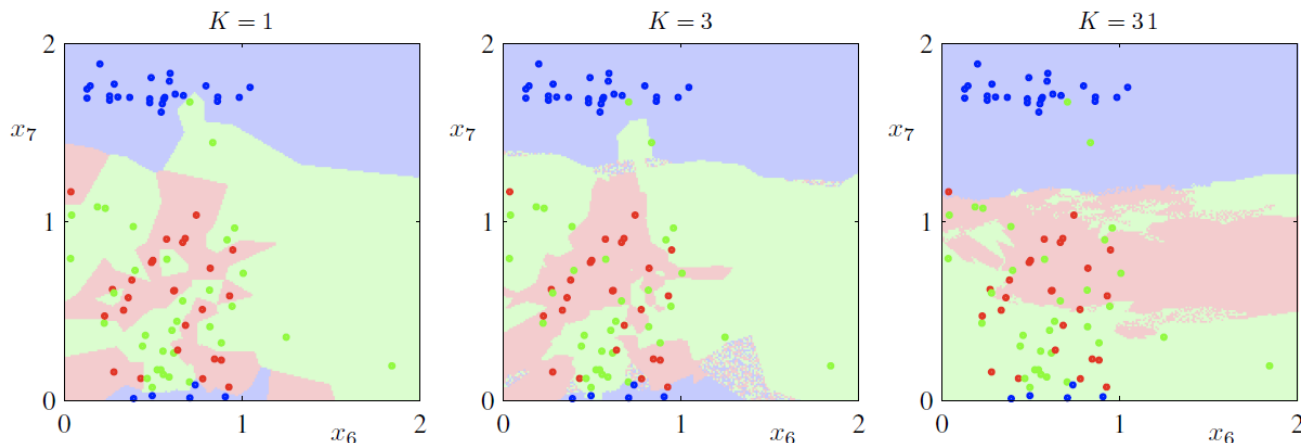
確率密度が一定と近似



多クラス識別も自然に実現できる

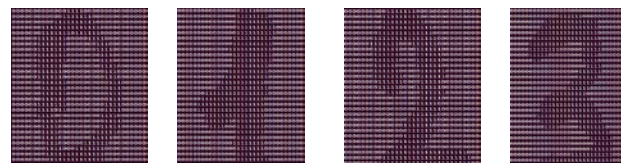
K-NN: 確率的な解釈

- 背後では、生成モデルの推定を行っている
 - 生成モデルのパラメータは置かないので、**ノンパラメトリック**な手法と呼ばれる
- Kは生成モデルの滑らかさをコントロールするパラメータ
 - モデルそのもののパラメータではないことがポイント
 - $K \rightarrow \text{大}$: より大域的・単純な分布
 - $K \rightarrow \text{小}$: より局所的・複雑な分布



手書き文字認識

- 0~9の手書きの数字をKNNで認識してみる
- digits.zipを解凍
 - 32x32サイズのバイナリ画像



```
>>> import kNN  
>>> kNN.handwritingClassTest()  
(テスト画像の識別がスタート)
```

KNN : コード

```
def handwritingClassTest():
    hwLabels = []
    trainingFileList = listdir('trainingDigits')
    m = len(trainingFileList)
    trainingMat = zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]      #take off .txt
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i,:] = img2vector('trainingDigits/%s' % fileNameStr)
    testFileList = listdir('testDigits')          #iterate through the test set
    errorCount = 0.0
    mTest = len(testFileList)    #テストサンプル数。多すぎて時間がかかる場合、適当に減らしてみるとよい
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]      #take off .txt
        classNumStr = int(fileStr.split('_')[0])
        vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
        classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
        print "the classifier came back with: %d, the real answer is: %d" % (classifierResult, classNumStr)
        if (classifierResult != classNumStr): errorCount += 1.0
    print "¥nthe total number of errors is: %d" % errorCount
    print "¥nthe total error rate is: %f" % (errorCount/float(mTest))
```

#load the training set
(全部メモリに持っておく必要がある)

k=3 (これを変えてみる)

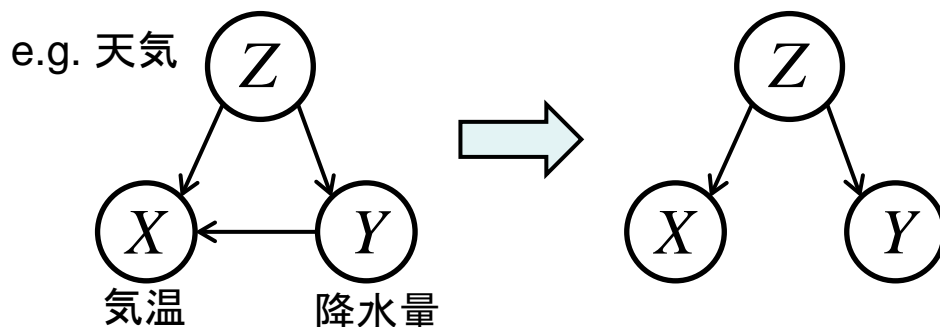
KNN : コード

```
def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX, (dataSetSize,1)) - dataSet    #まず引き算
    sqDiffMat = diffMat**2    #要素ごとに二乗
    sqDistances = sqDiffMat.sum(axis=1)    #和をとる (二乗和)
    distances = sqDistances**0.5    #平方根 (ユークリッド距離)
    sortedDistIndicies = distances.argsort()
    classCount={}
    for i in range(k):    #最近傍k個までラベルを見る
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    sortedClassCount = sorted(classCount.iteritems(),
key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]
```

ナイーブベイズ (単純ベイズ)

- もともと識別手法の名前ではない（非常に一般的かつ重要な概念）
 - 条件付きの同時確率をこの条件付き確率の積へばらす近似方法

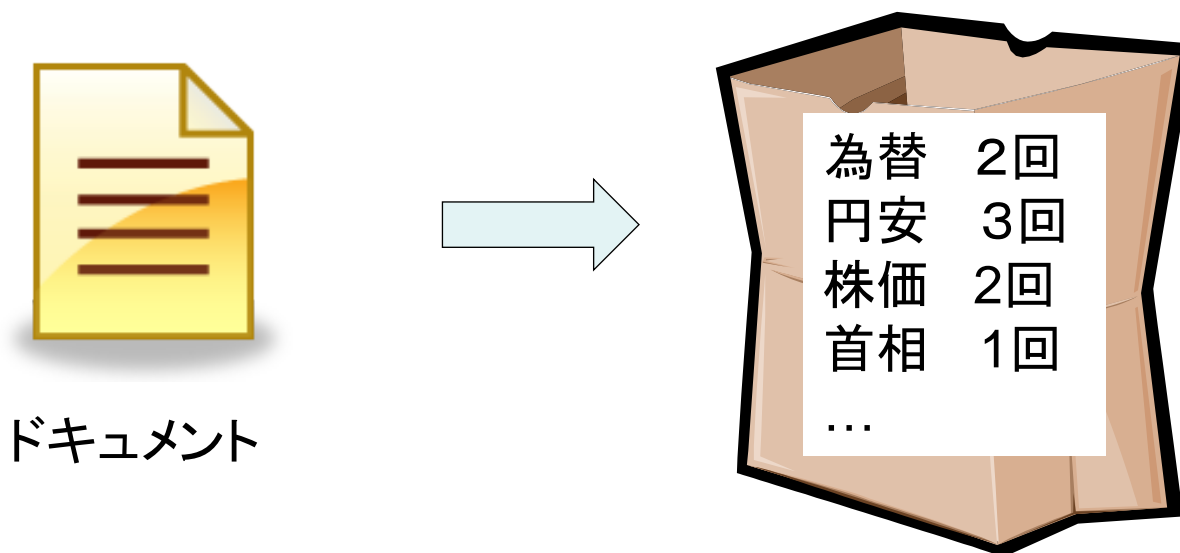
$$P(X, Y | Z) \cong P(X | Z)P(Y | Z)$$



- Z が潜在的な構造をよく捉えていれば、比較的妥当な近似になると期待できる

ナイーブベイズ識別

- テキスト分類の基本的な手法
- ドキュメントを、出現する単語の集合として表現
=bag-of-words
 - 出現回数だけ利用
 - 各単語の位置や出現順などのコンテキストは考慮しない



ナイーブベイズ識別

ドキュメント D の事後確率

$$P(C | D) \propto \underbrace{P(D | C)}_{\text{訓練サンプル中の比率で近似(あるいは単に一定)}} P(C)$$

訓練サンプル中の比率で近似(あるいは単に一定)

ナイーブベイズ

$$\hat{P}(D | C) = P(W_1, W_2, \dots, W_n | C) \propto P(W_1 | C) P(W_2 | C) \cdots P(W_n | C)$$

$$P(W_i | C) = \frac{\text{カテゴリ } C \text{ に属する訓練データ中の単語 } W_i \text{ の数}}{\text{カテゴリ } C \text{ に属する訓練データの全単語数}}$$

- あらかじめ、訓練データ中の単語を数え上げておくだけで識別ができる！

例) スパムメール識別

非スパム

Hi Peter,

With Jose out of town, do you want to meet once in a while to keep things going and do some interesting stuff?

Let me know
Eugene

スパム

--- Codeine 15mg -- 30 for \$203.70 -
- VISA Only!!! --

-- Codeine (Methylmorphine) is a
narcotic (opioid) pain reliever
-- We have 15mg & 30mg pills --
30/15mg for \$203.70 - 60/15mg for
\$385.80 - 90/15mg for \$562.50 --
VISA Only!!! ---

```
>>> import bayes
```

```
>>> bayes.spamTest()
```

```
classification error ['home', 'based', 'business', 'opportunity', 'knocking', 'your', 'door', 'don',  
'rude', 'and', 'let', 'this', 'chance', 'you', 'can', 'earn', 'great', 'income', 'and', 'find', 'your',  
'financial', 'life', 'transformed', 'learn', 'more', 'here', 'your', 'success', 'work', 'from', 'home',  
'finder', 'experts']
```

```
the error rate is: 0.1
```

(ランダムにサンプルを選ぶので毎回違った結果になる)

スパム識別：コード

```
def spamTest():
    (略：データの読み込みなど)
    vocabList = createVocabList(docList)    #create vocabulary
    trainingSet = range(50); testSet=[]    #create test set
    for i in range(10):
        randIndex = int(random.uniform(0,len(trainingSet))) #ランダムに10サンプルをテスト用を選ぶ
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainMat=[]; trainClasses = []
    for docIndex in trainingSet: #train the classifier (get probs) trainNB0
        trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V,p1V,pSpam = trainNB0(array(trainMat),array(trainClasses))
    errorCount = 0
    for docIndex in testSet:    #classify the remaining items
        wordVector = bagOfWords2VecMN(vocabList, docList[docIndex])
        if classifyNB(array(wordVector),p0V,p1V,pSpam) != classList[docIndex]:
            errorCount += 1
        print "classification error",docList[docIndex]
    print 'the error rate is: ',float(errorCount)/len(testSet)
    #return vocabList,fullText
```

スパム識別：コード

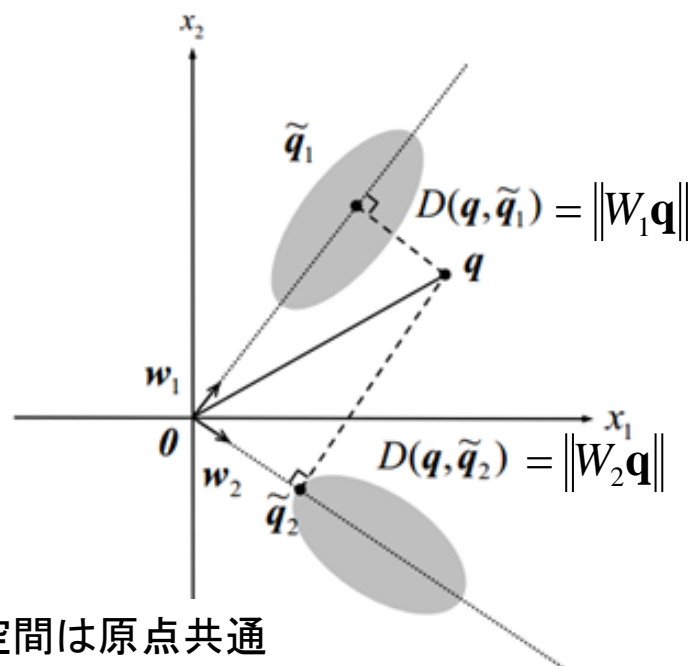
```
def trainNB0(trainMatrix,trainCategory):    # 2 クラス識別が前提
    numTrainDocs = len(trainMatrix)
    numWords = len(trainMatrix[0])
    pAbusive = sum(trainCategory)/float(numTrainDocs)    # スпамメールの事前確率
    p0Num = ones(numWords); p1Num = ones(numWords)    #change to ones()
    p0Denom = 2.0; p1Denom = 2.0    #change to 2.0
    for i in range(numTrainDocs):    #クラスごとに単語の数え上げ&条件付き確率計算
        if trainCategory[i] == 1:
            p1Num += trainMatrix[i]
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    p1Vect = log(p1Num/p1Denom)    #change to log()
    p0Vect = log(p0Num/p0Denom)    #change to log()
    return p0Vect,p1Vect,pAbusive
```

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):    # 2 クラス識別が前提

    p1 = sum(vec2Classify * p1Vec) + log(pClass1)    #element-wise mult
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
```

おまけ：部分空間法 (CLAFIC)

- 各クラスの成す部分空間 (PCAで学習) への近さを基準に識別
- 特徴が線形な構造を有している場合に有効
- 日本発の技術



距離最小基準: $\hat{C} = \arg \min_{C_i} \|W_i \mathbf{q}\|$

角度最小基準: $\hat{C} = \arg \min_{C_i} \frac{\|W_i \mathbf{q}\|}{\|\mathbf{q}\|}$

※部分空間は原点共通
(自己相関行列によるPCA)

まとめ

- クラス分類（クラス識別）
 - 前提として、特徴抽出は重要
 - ベイズの定理、事後確率、ベイズ識別則
 - 識別的アプローチ、生成的アプローチの違い
- 識別的アプローチ：クラス間の“違い”だけ分かればよい
 - 線形識別関数：SVM、最小二乗識別
 - 線形識別モデル：ロジスティック回帰
- 生成的アプローチ：クラスの分布も知りたい
 - k-NN、ナイーブベイズ

次回は22日です

- 15日は月曜日の講義なので注意
- 二回目のレポート課題は15日までにホームページに掲載します。
- 次回（予定）
 - SVM、アンサンブル学習、random forest