線形判別分析

- 線形判別分析（linear discriminant analysis, LDA）
  - ロジスティック回帰のように、直接 $\Pr(Y = k \mid X = x)$ をモデル化するのではなく、 $\Pr(X = x \mid Y = k)$ を正規分布でモデル化し、ベイズの定理によって $\Pr(Y = k \mid X = x)$ を計算

- ロジスティック回帰との比較
  - ロジスティック回帰はクラス間のデータ分布が完全に分かれている場合、パラメータ推定が不安定になる
  - データ分布がクラスごとに正規分布に従っている場合、線形判別分析のほうがパラメータ推定が安定する

線形判別分析

- ベイズの定理による事後確率の計算

$$\Pr\left(Y=k \middle| X=x\right)=\frac{\Pr\left(Y=k\right)\Pr\left(X=x \middle| Y=k\right)}{\Pr\left(X=x\right)}$$

$$=\frac{\Pr\left(Y=k\right)\Pr\left(X=x \middle| Y=k\right)}{\sum_{l=1}^{K}\Pr\left(X=x, Y=l\right)}$$

$$=\frac{\Pr\left(Y=k\right)\Pr\left(X=x \middle| Y=k\right)}{\sum_{l=1}^{K}\Pr\left(Y=l\right)\Pr\left(X=x \mid Y=l\right)}$$

$$p_k\left(x\right)=\frac{\pi_k f_k\left(x\right)}{\sum_{l=1}^{K}\pi_l f_l\left(x\right)}$$

K: クラスの数
π_k: クラス k の事前確率
$f_k(x)$: クラス k のデータの密度関数

線形判別分析

- 予測変数がひとつ（p = 1）の場合
  - $f_k(x)$ が正規分布だと仮定

$$f_k(x)=\frac{1}{\sqrt{2\pi}\sigma_k}\exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right)$$

  - 分散がすべてのクラスで等しいとすると

$$p_k(x)=\frac{\pi_k\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)}{\sum_{l=1}^{K}\pi_l\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(x-\mu_l)^2\right)}$$

線形判別分析

- 分類

$$p_k(x) \propto \pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)$$

が最も大きいクラス k に分類すればよい。右辺の対数をとってクラスに依存しない項を削除した、

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

を考えても良い。

線形判別分析

- 分類

$K = 2, \pi_1 = \pi_2$ の場合、分離境界は

$$x \cdot \frac{\mu_1}{\sigma^2} - \frac{\mu_1^2}{2\sigma^2} + \log(\pi_1) = x \cdot \frac{\mu_2}{\sigma^2} - \frac{\mu_2^2}{2\sigma^2} + \log(\pi_2)$$

より

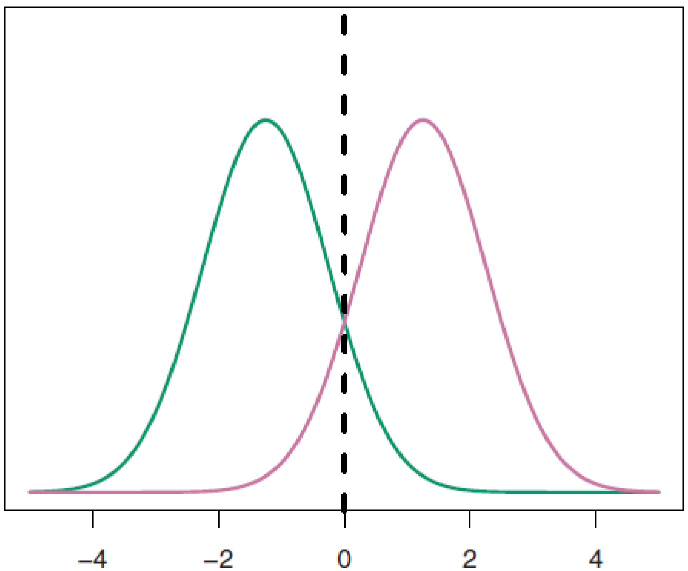
$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2}$$

線形判別分析

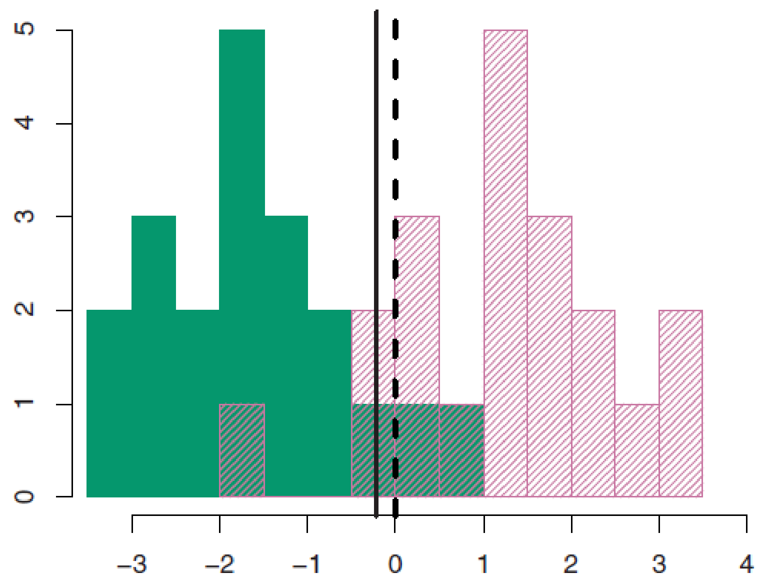
- 例(人工データ)

それぞれの正規分布から20個ずつサンプリング

点線:ベイズ決定境界



実線:LDA決定境界



$\mu_1 = -1.25$ $\mu_2 = 1.25$

$\sigma_1^2 = 1$ $\sigma_2^2 = 1$

ベイズエラー率:10.6%

LDAのテストエラー率:11.1%

線形判別分析

- パラメータ推定
  - 実際には μ_k, σ^2 は未知なのでデータから推定

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i \qquad \hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} \left(x_i - \hat{\mu}_k\right)^2$$

n: 学習データの全事例数
n_k: クラス k に属する事例の数

線形判別分析

- パラメータ推定
  - π_k も未知の場合はデータから推定

$$\hat{\pi}_k = \frac{n_k}{n}$$

  - これらの推定値を用いて

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k)$$

    がもっとも大きくなるクラスに分類

  - 判別関数 $\hat{\delta}_k(x)$ は x の線形関数

線形判別分析

- 予測変数が2個以上（$p > 1$）の場合
  - X（p次元のベクトル）が多変量正規分布（multivariate Gaussian distribution）に従うと仮定

$$X \sim N(\mu, \Sigma)$$

$$E(X) = \mu$$ ・・・Xの平均

$$\mathrm{Cov}(X) = \Sigma$$ ・・・Xの分散共分散行列（$p \times p$）

  - 多変量正規分布の確率密度関数

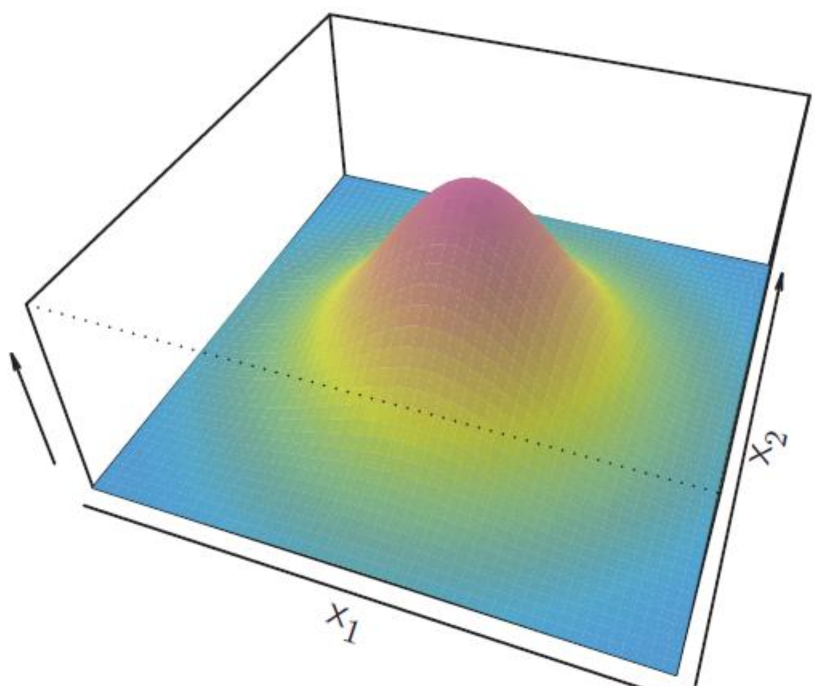
$$f(x) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu) \right)$$

Σの行列式

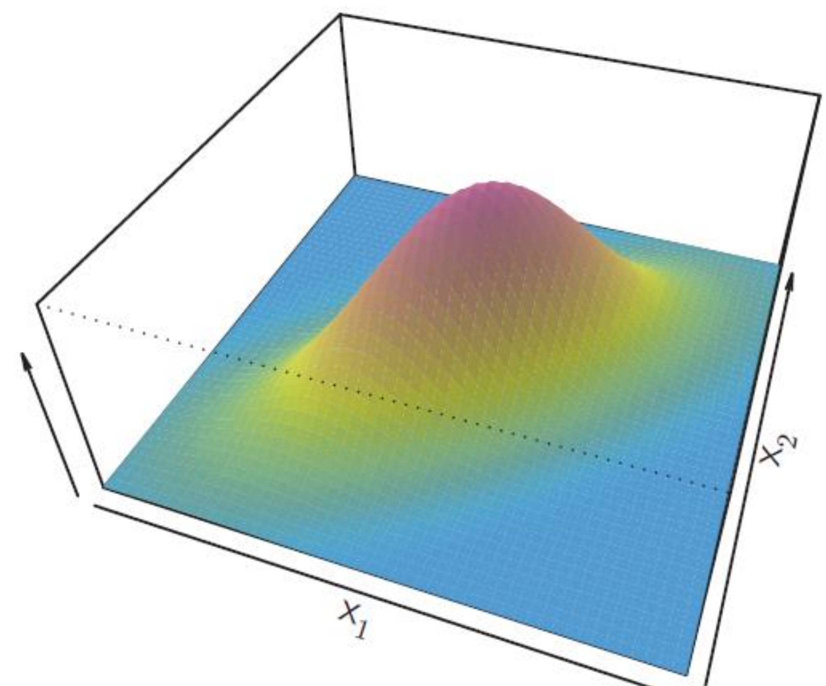
線形判別分析

- 多変量正規分布

２つの変数に相関がない場合



相関係数0.7



線形判別分析

- 分類

$$\pi_k f(x) = \frac{\pi_k}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k)\right)$$

が最大のクラスに分類すればよい

対数をとって k に依存しない項を削除すると

$$\delta_k(x) = x^T \Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T \Sigma^{-1}\mu_k + \log \pi_k$$

$\rightarrow x$ の線形関数

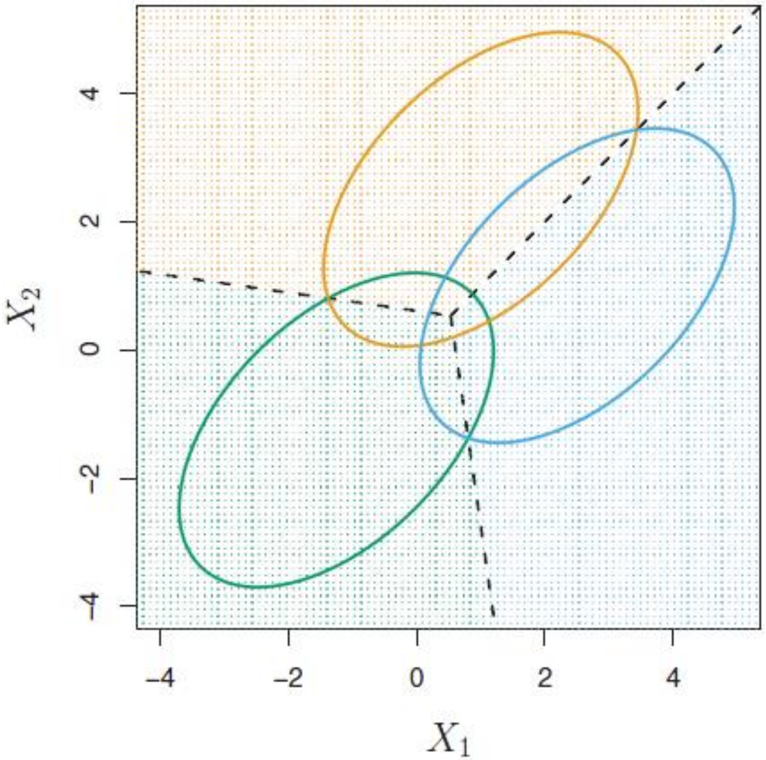
線形判別分析

K = 3, p = 2 の例

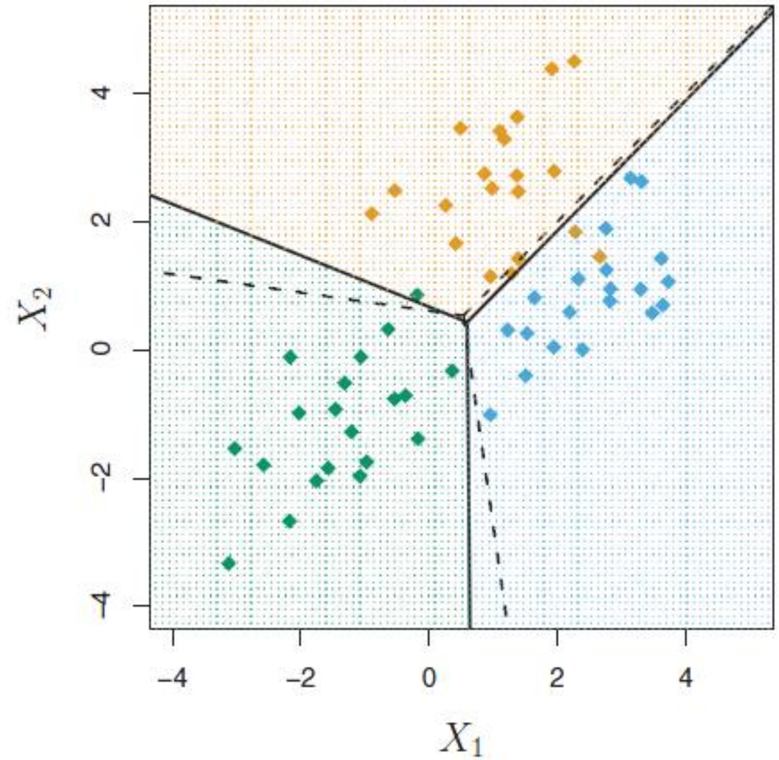
学習データ：各クラス20事例

楕円：各多変量正規分布の95%の範囲

点線：ベイズ決定境界



実線：LDA決定境界



ベイズエラー率：7.46%

LDAのテストエラー率：7.7%

線形判別分析

- LDA決定境界
  - クラス k とクラス l の境界

$$\delta_k(x) = \delta_l(x)$$

$$x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k = x^T \Sigma^{-1} \mu_l - \frac{1}{2} \mu_l^T \Sigma^{-1} \mu_l + \log \pi_l$$

  - 前ページの例の場合 $\pi_k = \pi_l$ なので

$$x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k = x^T \Sigma^{-1} \mu_l - \frac{1}{2} \mu_l^T \Sigma^{-1} \mu_l$$

$\rightarrow$ 直線（$p > 2$ の場合は超平面）

線形判別分析

- パラメータ推定

$$\hat{\pi}_k = \frac{n_k}{n}$$

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

$$\hat{\Sigma} = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$$

線形判別分析

- Default データセットでの結果
  - $p = 2$: balance と student
  - 混同行列（confusion matrix）

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

学習エラー率　$1 - (9644+81)/10000 = 0.0275$

小さく見えるが、単純にすべて No と判定した場合でもエラー率は $333/10000 = 0.0333$

線形判別分析

- 正解率以外の評価指標

		True default status		
		No	Yes	Total
Predicted default status	No	True Neg. (TN)	False Neg. (FN)	TN+FN
	Yes	False Pos. (FP)	True Pos. (TP)	TP+FP
	Total	TN+FP	TP+FN	TP+TN+FP+FN

- 感度(sensitivity): Yes であるものを正しく検出できた割合

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \qquad \frac{81}{81+252}=0.243$$

- 特異度(specificity): No であるものを正しく検出できた割合

$$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} \qquad \frac{9644}{9644+23}=0.998$$

線形判別分析

- 感度が低い理由
  - LDA はエラー率が小さくなるように分類、つまり

$\Pr(\text{default} = \text{Yes} \mid X = x) > 0.5$ であれば default に分類

- 感度を改善するには判別の閾値を変えて、例えば、

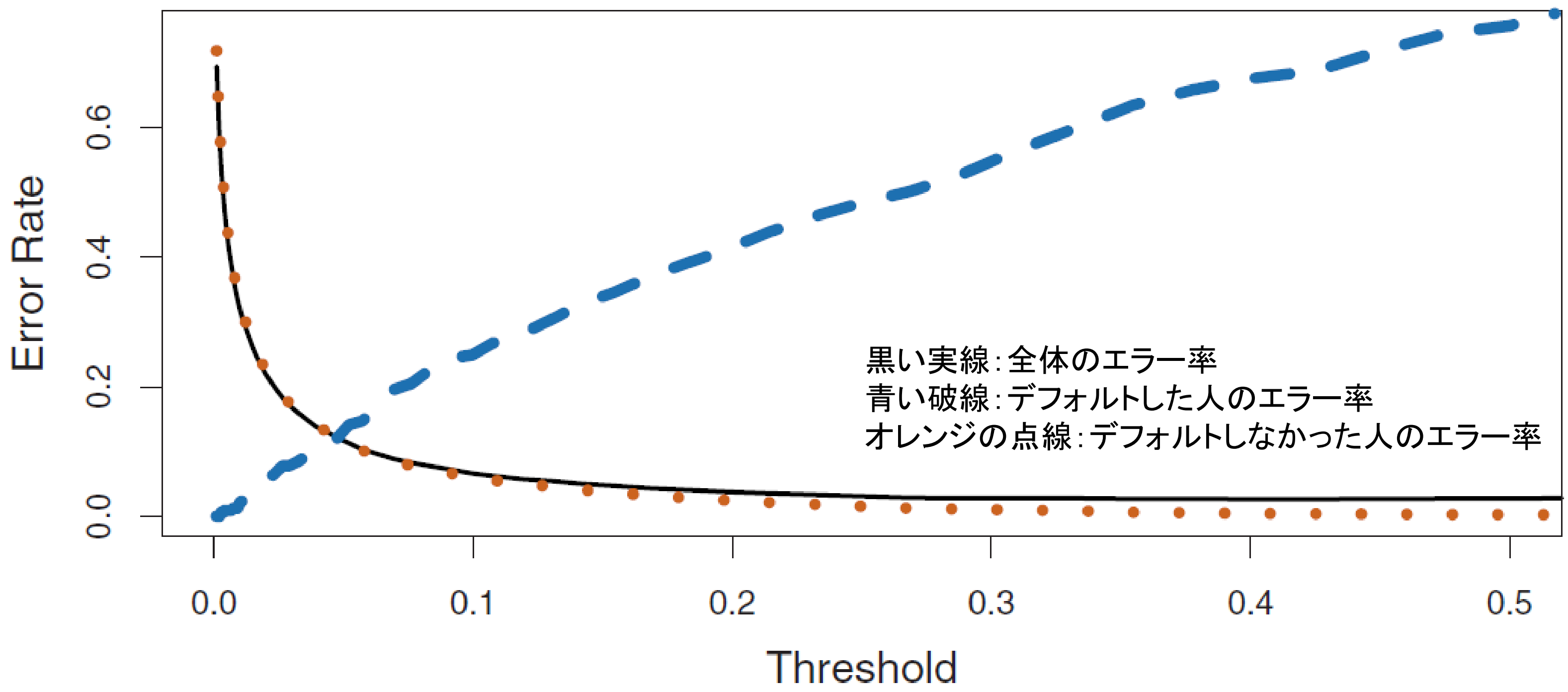
$\Pr(\text{default} = \text{Yes} \mid X = x) > 0.2$ であれば default に分類

		True default status		
		No	Yes	Total
Predicted default status	No	9,432	138	9,570
	Yes	235	195	430
	Total	9,667	333	10,000

感度 = 195 / 333 = 0.586　　　エラー率 = 1 – (9432+195) = 0.0373

線形判別分析

- 判別の閾値とエラー率



閾値を下げると感度は上がるが全体のエラー率は上がる

線形判別分析

- ROC曲線（receiver operating characteristics curve）
  - 判別の閾値を変えて TP 率と FP 率をプロット

True positive rate = Sensitivity

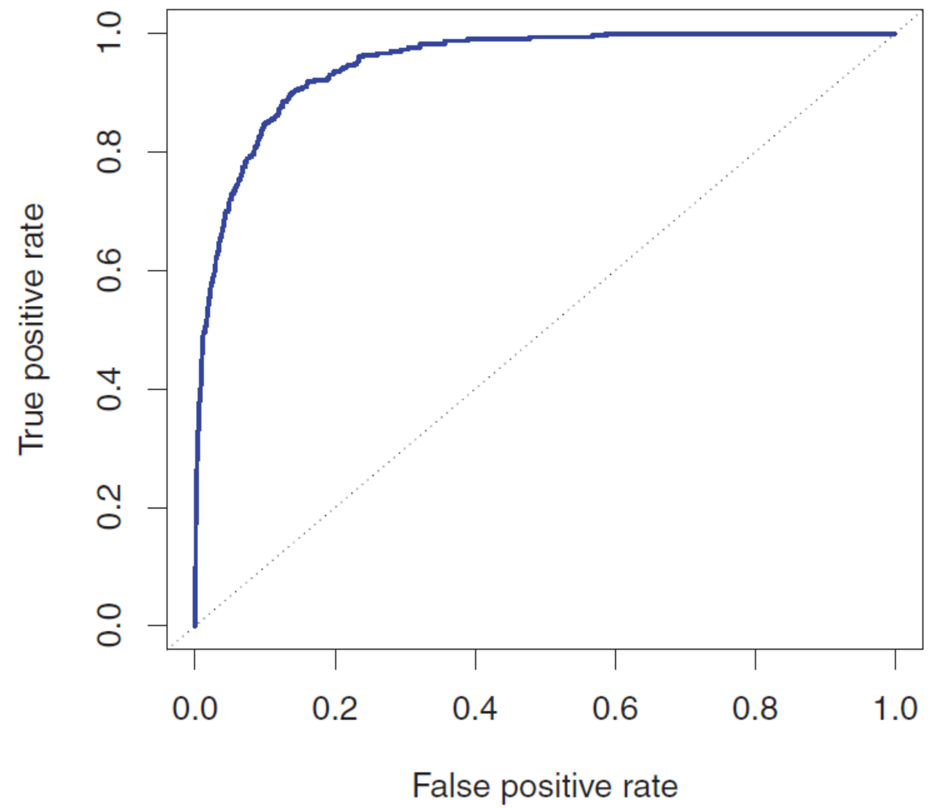
$$= \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

False positive rate = 1 - Specificity

$$= \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}}$$

AUC (Area Under the Curve)

= 曲線の下の部分の面積



線形判別分析

- 評価指標のまとめ

		True class		
		− or Null	+ or Non-Null	Total
Predicted class	− or Null	TN	FN	N*
	+ or Non-Null	FP	TP	P*
	Total	N	P	

名称	定義	別名
False Positive rate	FP/N	Type I error, 1 – Specificity
True Positive rate	TP/P	1 – Type II error, power, sensitivity, recall
Positive Predictive value	TP/P*	Precision, 1 – false discovery proportion
Negative Predictive value	TN/N*	

Python実習

- Stock Market データセット
  - Smarket
    - S&P 500 stock index の値動き（%）データ
    - 2001年～2005年（1250日分）

Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
2001	0.381	-0.192	-2.624	-1.055	5.01	1.1913	0.959	Up
2001	0.959	0.381	-0.192	-2.624	-1.055	1.2965	1.032	Up
2001	1.032	0.959	0.381	-0.192	-2.624	1.4112	-0.623	Down
2001	-0.623	1.032	0.959	0.381	-0.192	1.276	0.614	Up
2001	0.614	-0.623	1.032	0.959	0.381	1.2057	0.213	Up
2001	0.213	0.614	-0.623	1.032	0.959	1.3491	1.392	Up
2001	1.392	0.213	0.614	-0.623	1.032	1.445	-0.403	Down
:	:	:	:	:	:	:	:	:

Python実習

- Smarket データセットの読み込み
  - ダウンロード
    - http://www.logos.t.u-tokyo.ac.jp/~tsuruoka/lecture/sml/Smarket.csv
    - 上記ファイルをホームディレクトリに保存（Windowsの場合）

```
>>> import os
>>> os.chdir(os.path.expanduser("~"))
```

  - 学習データ・テストデータの準備

```
>>> import pandas as pd
>>> from patsy import dmatrices
>>> Smarket = pd.read_csv('Smarket.csv')
>>> Smarket_train = Smarket.query('Year < 2005')
>>> y_train, X_train = dmatrices('Direction ~ Lag1 + 
Lag2', Smarket_train, return_type = 'dataframe')
>>> Smarket_2005 = Smarket.query('Year >= 2005')
>>> y_test, X_test = dmatrices('Direction ~ Lag1 + Lag2', 
Smarket_2005, return_type = 'dataframe')
```

Python実習

- 線形判別分析
  - LinearDiscriminantAnalysis()

```
>>> from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis as LDA
>>> trained = LDA().fit(X_train.iloc[:,1:3], y_train.iloc[:,1])
>>> trained.priors_
array([0.49198397, 0.50801603])
>>> trained.means_
array([[ 0.04279022,  0.03389409],
       [-0.03954635, -0.03132544]])

>>> preds = trained.predict(X_test.iloc[:,1:3])
>>> import numpy as np
>>> np.mean(preds == y_test.iloc[:,1])
0.5595238095238095
```

$\hat{\pi}_1 = 0.492,\ \hat{\pi}_2 = 0.508$

$\hat{\mu}_1 = \begin{pmatrix} 0.0428 \\ 0.0339 \end{pmatrix},\ \hat{\mu}_2 = \begin{pmatrix} -0.0395 \\ -0.0313 \end{pmatrix}$

Python実習

- 事後確率
  - Pr(Y | X)

```
>>> probs = trained.predict_proba(X_test.iloc[:,1:3])
>>> print(probs)
array([[0.49017925, 0.50982075],
       [0.4792185 , 0.5207815 ],
       [0.46681848, 0.53318152],
...
```

- 異なる閾値での判定

```
>>> np.mean((probs[:,1] >= 0.51) == y_test.iloc[:,1])
0.5317460317460317
```

二次判別分析

- 二次判別分析（quadratic discriminant analysis, QDA）
  - LDA 同様、各クラスのデータは多変量正規分布と仮定
  - ただし、クラスごとに異なる分散共分散行列を想定

$$X \sim N\left(\mu_k, \Sigma_k\right)$$

  - 判別

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}\left(x-\mu_k\right)^T \Sigma_k^{-1}\left(x-\mu_k\right) - \frac{1}{2}\log\left|\Sigma_k\right| + \log \pi_k \\ &= -\frac{1}{2}x^T\Sigma_k^{-1}x + x^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\log\left|\Sigma_k\right| + \log \pi_k\end{aligned}$$

が一番大きいクラスに分類

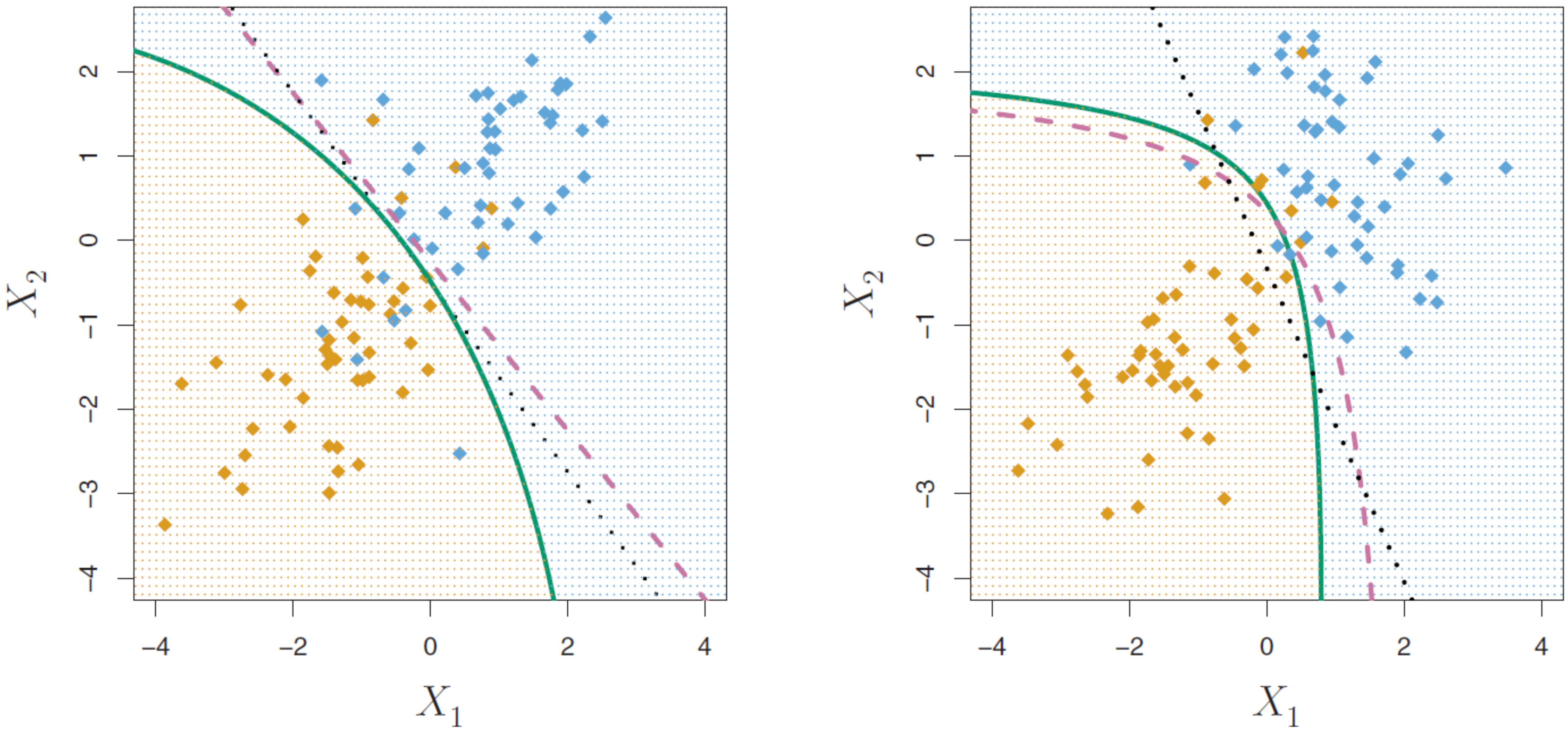
二次判別分析

- LDA vs QDA
  - QDA はクラスごとに分散共分散行列のパラメータを求める必要がある
    - パラメータ数: $Kp(p+1)/2$
    - 学習データが小さいときには過学習の可能性
  - LDA はバリアンスは小さいが分散共分散行列が共通という仮定が現実とあっていない場合はバイアスが大きくなる

二次判別分析

紫の破線：ベイズ決定境界
黒い点線：LDA決定境界
緑の曲線：QDA決定境界

$\Sigma_1 = \Sigma_2$ $\qquad$ $\Sigma_1 \neq \Sigma_2$



分類手法の比較

- 分類手法
  - ロジスティック回帰
  - 線形判別分析（LDA）
  - 二次判別分析（QDA）
  - K最近傍法（KNN）

- どの手法をどのような状況で使うべきか

分類手法の比較

- ロジスティック回帰とLDA
  - LDA

p=1の場合

$$\log\left(\frac{p_1(x)}{1-p_1(x)}\right)=\log\left(\frac{p_1(x)}{p_2(x)}\right)=c_0+c_1x$$

ただし、c_0, c_1 は $\mu_1, \mu_2, \sigma, \pi_1, \pi_2$ の関数

  - ロジスティック回帰

$$\log\left(\frac{p_1(x)}{1-p_1(x)}\right)=\beta_0+\beta_1x$$

  - 決定境界はどちらも直線（または超平面）
  - LDA はデータの分布が各クラスそれぞれ正規分布であると仮定（共分散行列は共通）
  - 似たような分類精度になることが多いが、LDA の仮定がほぼ満たされている場合は LDA が有利

分類手法の比較

- KNN
  - ノンパラメトリック
    - 分離境界に対する仮定なし
    - 非線形で複雑な分離境界を持つ問題にも対応できる
  - どの予測変数が重要かわからない

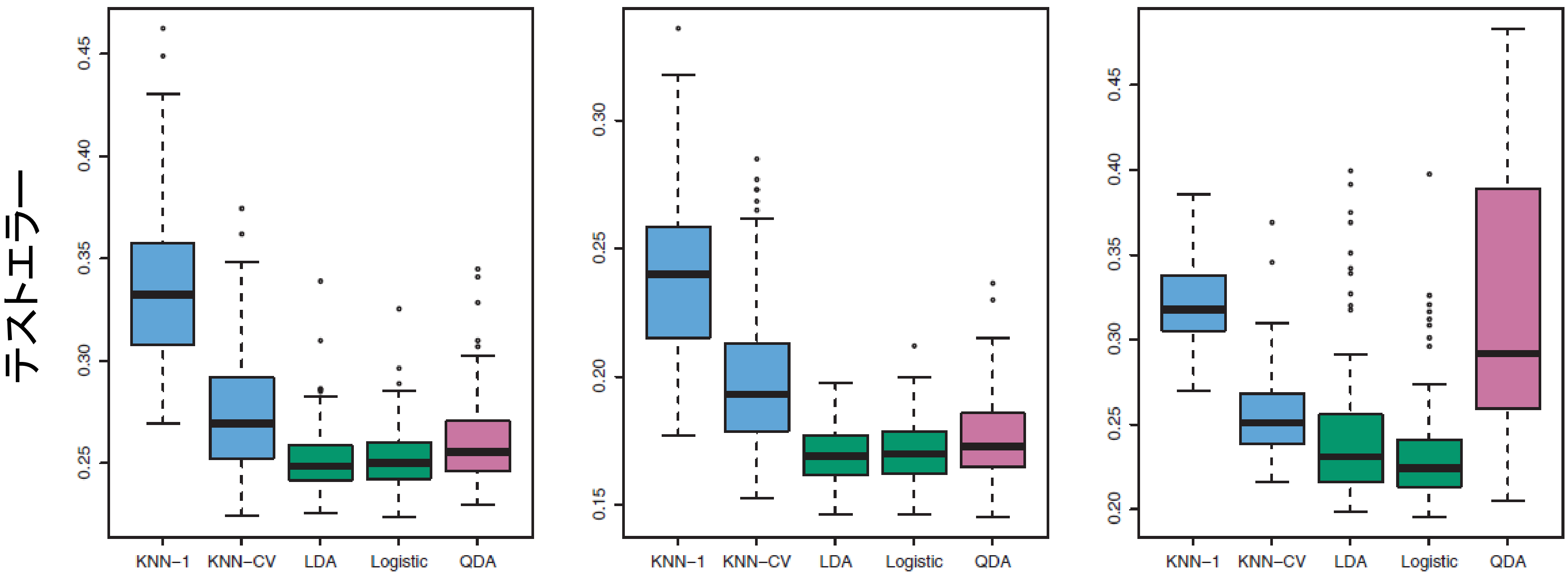
- QDA
  - KNN ほど柔軟ではないがLDAやロジスティック回帰よりは幅広い問題をモデル化できる
  - 学習データが少ないときは KNN より有利

分類手法の比較

- 学習データ：各クラス20個
- 各クラスのデータ分布
  - 正規分布（変数間に相関なし）
  - 分散共分散行列は共通

- 学習データ：各クラス20個
- 各クラスのデータ分布
  - 正規分布（変数間に相関あり）
  - 分散共分散行列は共通

- 学習データ：各クラス50個
- 各クラスのデータ分布
  - *t*分布



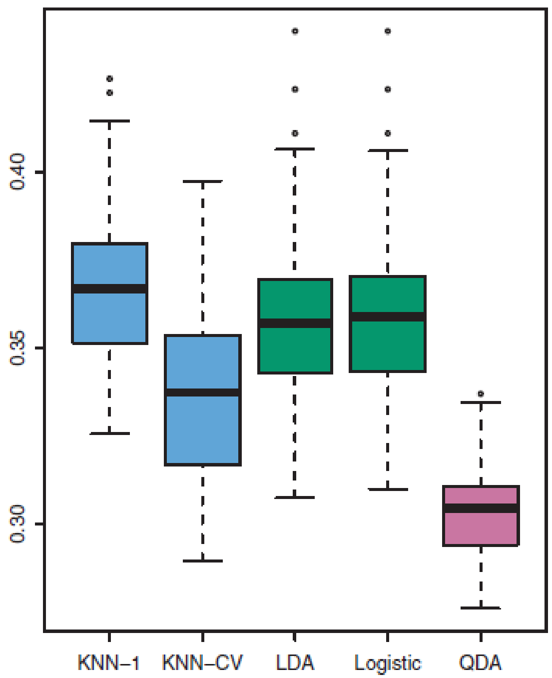
分類手法の比較

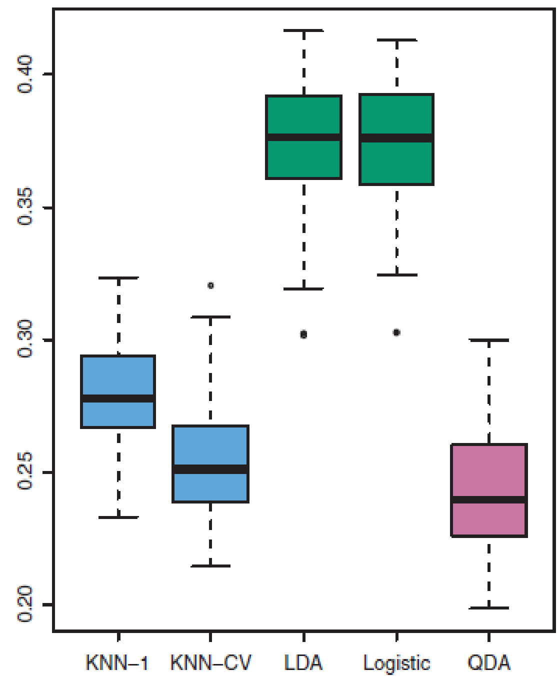
- 各クラスのデータ分布
  - 正規分布（変数間に相関あり）
  - 分散共分散行列は異なる

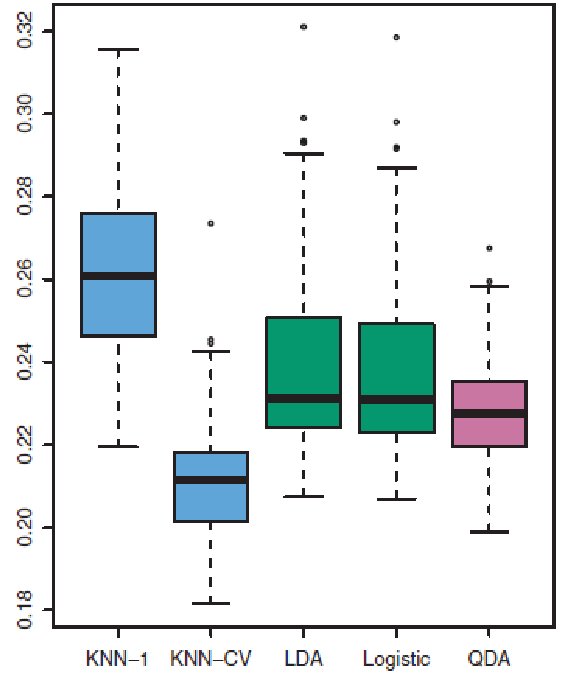
- 各クラスのデータ分布
  - 正規分布（変数間に相関なし）
- 応答変数の値
  - X_1^2, X_2^2, $X_1 \times X_2$ を入力変数としたロジスティック関数からサンプリング
- 決定境界：二次関数

- 各クラスのデータ分布
  - 正規分布（変数間に相関なし）
- 応答変数の値
  - 左図の場合よりもさらに複雑な非線形関数からサンプリング

テストエラー







分類手法の比較

- まとめ
  - どのような状況でも常にベストというような手法はない
  - 真の決定境界が線形であれば LDA かロジスティック回帰
  - 決定境界に多少の非線形性がある場合は QDA
    - ただし LDA やロジスティック回帰でも、予測変数に二次の変数や積の変数などを追加すれば非線形性にある程度対処することは可能
  - 決定境界の非線形性が強い場合は KNN

Python実習

- 二次判別分析
  - QuadraticDiscriminantAnalysis()

```
> from sklearn.discriminant_analysis import
QuadraticDiscriminantAnalysis as QDA
>>> trained = QDA().fit(X_train.iloc[:,1:3], y_train.iloc[:,1])
>>> preds = trained.predict(X_test.iloc[:,1:3])
>>> np.mean(preds == y_test.iloc[:,1])
0.5992063492063492
```

Python実習

- K最近傍法
  - KNeighborsClassifier

```
>>> from sklearn.neighbors import KNeighborsClassifier as KNN
>>> trained = KNN(n_neighbors=1).fit(X_train.iloc[:,1:3],
y_train.iloc[:,1])
>>> preds = trained.predict(X_test.iloc[:,1:3])
>>> np.mean(preds == y_test.iloc[:,1])
0.5
```

→ ｋの値を変えるとどうなるか？

Python実習

- Caravan Insurance データセット
  - 事例数：5822人
  - 予測変数：人種、年齢、収入、etc
  - 応答変数：Caravan Insurance を購入したかどうか
  - ダウンロード
    - https://www.logos.t.u-tokyo.ac.jp/~tsuruoka/lecture/sml/Caravan.csv

```
> Caravan = pd.read_csv('Caravan.csv')
> Caravan.shape
> Caravan.head()
> Caravan.describe()
```

Python実習

- 標準化
  - 入力変数のスケールをそろえる
    - 平均をゼロ、標準偏差を1にする

```
>>> labels = pd.DataFrame(np.zeros(len(Caravan)))
>>> labels.loc[Caravan['Purchase'] == 'Yes'] = 1
>>> CaravanX = Caravan.drop(labels='Purchase', axis=1)
>>> train_size = 1000
>>> X_test  = CaravanX.iloc[0:train_size, ]
>>> X_train = CaravanX.iloc[train_size:, ]
>>> Y_test  = labels.iloc[0:train_size, ]
>>> Y_train = labels.iloc[train_size:, ]
>>>
>>> from sklearn import preprocessing
>>> scaler = preprocessing.StandardScaler()
>>> scaler.fit(X_train)
>>> X_train_scaled = scaler.transform(X_train)
>>> print(X_train_scaled.mean(axis=0))
>>> print(X_train_scaled.std(axis=0))
>>> X_test_scaled = scaler.transform(X_test)
```

Python実習

- 精度評価

```
>>> trained = KNN(n_neighbors = 1).fit(X_train_scaled,
Y_train.iloc[:,0])
>>> X_test_labels = trained.predict(X_test_scaled)
>>> X_test_prob = trained.predict_proba(X_test_scaled)
>>> np.mean(Y_test.iloc[:,0] == X_test_labels)
...
```

Python実習

- 精度評価
  - 保険会社としては、Yes と判定された人のうち、どれだけの人が実際に保険を購入してくれるのかを知りたい

```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(Y_test.iloc[:,0], X_test_labels)
array([[874,  67],
       [ 50,   9]], dtype=int64)
>>> 9 / (67 + 9)
0.11842105263157894
```

→ 意外と悪くない（ランダムに選択した場合は約6%）

→ k の値を変えるとどうなるか？