機械学習：

初めに

機械学習モデリングプロセス

1.問題定義 :何を解決したいか

2.データ選定:( どのようなデータを使用するか。収集方法も重要)

3.データ前処理:(正規化、欠損値補完など)

4.モデルの選定

線形回帰モデル

線形回帰モデル

ロジスティック回帰モデル

主成分分析

サポートベクターマシーン

...

多数存在するモデルから選択

複数のモデルを試行し、比較検討する場合もある。

5.モデルの学習

パラメータの調整

6.モデルの評価

モデルの精度を評価

このレポートでは4.モデルの選定 を中心に記載する。

モデルの分類

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学習方法 | タスク | モデル名称 | パラメータの推定(アルゴリズム) | モデル選択・評価 |
| 教師あり学習 | 予測 | 線形回帰  非線形回帰 | 最小2乗法  最尤法 | ホールドアウト法  交差検証法 |
| 分類 | サポートベクターマシン | マージン最大化 |
| ロジスティック回帰 | 最尤法+ 確率的勾配降下法 |
| k-近傍法 | |
| 教師なし学習 | クラスタリング | k平均法 | | 無し |
| 主成分分析 | 固有値分解 |

線形回帰モデル

・回帰問題

ある入力(数値)から出力(連続値)を予測する問題

入力はm次元のベクトル、出力はスカラー値

入力ベクトルの各要素を説明変数または特徴量と呼ぶ

出力のスカラー値を目的変数と呼ぶ

m=1の場合、単回帰 m>1の場合、重回帰と呼ぶ

・線形回帰モデル

回帰問題を解くための機械学習モデル

教師あり学習に分類される

入力(説明変数)とm次元パラメータの線形結合に及び切片を加算したものを出力する

入力(説明変数)を(,,…………)

パラメータを(,,…………)とすると

= ) + ,行列表現では = **x** +

( = 1として、 = ) , = **x**と

表現することもある)

・モデルの評価

データを学習用と検証用データに分割

学習用データでモデルを学習し、検証用データでモデルを検証

(決定係数),RMSE(二乗平均平方根誤差), MAE(平均絶対誤差)などが

精度指標として用いられる。

・パラメータの推定

最小二乗法により推定

学習データのMSE(平均二乗誤差) を最小とするパラメータを探索

⇒　勾配が0になる点を求める。

パラメータの推定値:訓練データ*X,* 訓練データの出力値yとすると

　　以下で求められる。

= y

ハンズオン  
ボストンの住宅価格予測

1. 線形単回帰分析 RMから価格を予測

# 線形単回帰分析  
# 説明変数  
data = df.loc [:, ['RM']].values  
# 目的変数  
target = df.loc [:, 'PRICE'].values  
  
## sklearnモジュールからLinearRegressionをインポート  
from sklearn.linear\_model import LinearRegression  
  
# オブジェクト生成  
model = LinearRegression ()  
# model.get\_params()  
model = LinearRegression (fit\_intercept=True, normalize=False, copy\_X=True, n\_jobs=1)  
  
# fit関数でパラメータ推定  
model.fit (data, target)  
# 単回帰の回帰係数と切片を出力  
print ('単回帰の回帰係数と切片 推定された回帰係数: %.3f, 推定された切片 : %.3f' % (model.coef\_, model.intercept\_))

単

単回帰の回帰係数と切片 推定された回帰係数: 9.102, 推定された切片 : -34.671

1. 線形重回帰分析 CRIM,RMから価格を予測

# 重回帰分析(2変数)  
# カラムを指定してデータを表示  
# print (df [['CRIM', 'RM']].head ())  
  
# 説明変数  
data2 = df.loc [:, ['CRIM', 'RM']].values  
# 目的変数  
target2 = df.loc [:, 'PRICE'].values  
# オブジェクト生成  
model2 = LinearRegression (fit\_intercept=True, normalize=False, copy\_X=True, n\_jobs=1)  
  
# fit関数でパラメータ推定  
model2.fit (data2, target2)  
# 予測  
# print (model2.predict ([[0.2, 6]]))  
  
# 回帰係数と切片の値を確認  
# 単回帰の回帰係数と切片を出力  
# print ('単回帰の回帰係数と切片 推定された回帰係数: %.3f, 推定された切片 : %.3f' % (model.coef\_, model.intercept\_))  
# 重回帰の回帰係数と切片を出力  
print ('重回帰の回帰係数と切片 推定された回帰係数:',model2.coef\_,' 推定された切片 : ',model2.intercept\_)  
  
# モデルの検証  
# 1. 決定係数  
print ('単回帰決定係数: %.3f, 重回帰決定係数 : %.3f' % (model.score (data, target), model2.score (data2, target2)))

重回帰の回帰係数と切片 推定された回帰係数: [-0.26491325 8.39106825] 推定された切片 : -29.24471945192995

単回帰決定係数: 0.484, 重回帰決定係数 : 0.542

⇒ 重回帰の方が決定係数が高く、より良いモデルと言える。

単回帰モデル（学習と検証に分割して、実施）

# train\_test\_splitをインポート  
from sklearn.model\_selection import train\_test\_split  
  
# 70%を学習用、30%を検証用データにするよう分割  
X\_train, X\_test, y\_train, y\_test = train\_test\_split (data, target,  
 test\_size=0.3, random\_state=666)  
# 学習用データでパラメータ推定  
model.fit (X\_train, y\_train)  
# 作成したモデルから予測（学習用、検証用モデル使用）  
y\_train\_pred = model.predict (X\_train)  
y\_test\_pred = model.predict (X\_test)  
# matplotlibをインポート  
import matplotlib.pyplot as plt  
  
# 学習用、検証用それぞれで残差をプロット  
plt.scatter (y\_train\_pred, y\_train\_pred - y\_train, c='blue', marker='o', label='Train Data')  
plt.scatter (y\_test\_pred, y\_test\_pred - y\_test, c='lightgreen', marker='s', label='Test Data')  
plt.xlabel ('Predicted Values')  
plt.ylabel ('Residuals')  
# 凡例を左上に表示  
plt.legend (loc='upper left')  
# y = 0に直線を引く  
plt.hlines (y=0, xmin=-10, xmax=50, lw=2, color='red')  
plt.xlim ([10, 50])  
plt.show ()  
  
# 平均二乗誤差を評価するためのメソッドを呼び出し  
from sklearn.metrics import mean\_squared\_error  
  
# 学習用、検証用データに関して平均二乗誤差を出力  
print ("学習用、検証用データに関して平均二乗誤差を出力")  
print ('MSE Train : %.3f, Test : %.3f' % (  
mean\_squared\_error (y\_train, y\_train\_pred), mean\_squared\_error (y\_test, y\_test\_pred)))  
# 学習用、検証用データに関してR^2を出力  
print ('R^2 Train : %.3f, Test : %.3f' % (model.score (X\_train, y\_train), model.score (X\_test, y\_test)))

学習用、検証用データに関して平均二乗誤差を出力

MSE Train : 44.983, Test : 40.412

R^2 Train : 0.500, Test : 0.434

非線形回帰モデル

・非線形回帰モデル

線形回帰モデル ⇒　線形構造が内在すると思われる現象に適用

非線形回帰モデル ⇒　非線形構造が内在すると思われる現象に適用

線形構造は単純なので、複雑な現象には適してないので

複雑な構造を表現できる非線形回帰モデルが必要である。

目的変数を基底関数と呼ばれる既知の非線形関数で変換したものとパラメータを

線型結合したものを出力

= )) +

パラメータは最小二乗法や最尤法で推定（線形回帰モデルと同様）

基底関数として、多項式関数やガウス型基底関数を用いる

・正則化法

過学習は 正則化法 で回避 正則化しすぎると未学習

未学習 : 学習データに対して、十分小さな誤差が得られない

過学習 : 小さな誤差は得られたけど、テスト集合誤差との差が大きい

正則化法はモデルの複雑さに伴って、その値が大きくなるペナルティ項を課した

関数の最小化を考える手法

・モデルの評価と選択

ホールドアウト法: データを学習とテスト用の2グループに分割して使用

データが大量にない場合は、精度が良くない

交差検証法：データをm個のグループに分割し、

m-1個のグループのデータ -> 学習用

残りのグループのデータ -> テスト用

これを m 回繰り返す。

全てのデータを学習用、テストに利用するので、精度が良い。

ロジスティック回帰モデル

・分類問題(クラス分類)

ある入力からクラスに分類する問題

入力は m 次元のベクトル (m=1)の時はスカラ

出力は 0 or 1 の値

・ロジスティック回帰モデル

分類問題を解くための機械学習モデル

入力からそのラベルを予測するシステムを構築すること

入力とパラメータの線形結合をシグモイド関数に入力

出力は y=1 になる確率の値になる

P(Y=1｜x) ＝ σ( ))

・パラメータの推定

　 最尤法で推定する

　 対数尤度関数をパラメータで微分して0になる値を求める必要があるが、

　 解析的にこの値を求めることは困難 ⇒ 勾配降下法により求める

・勾配降下法

　 反復学習によりパラメータを逐次的に更新するアプローチの一つ

　 = – η∂E/∂w

　 ηは学習率と呼ばれるハイパーパラメータでモデルのパラメータの

収束しやすさを調整

　※ 線形回帰モデルではMSEのパラメータに関する微分が0になる値を解析に求めることが可能だが、実処理では勾配降下法によりパラメータを求める。

・分類の評価方法

混同行列を用いる。

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 検証用データの結果 | |
| Positive | Negative |
| モデルの予測結果 | Positive | True Positive(TP) | False Positive(FP) |
| Negative | False Negative(FN) | True Negative(TN) |

True Positive ：PositiveをPositiveと判別

False Positive ：PositiveをNegativeと判別

False Negative ：NegativeをPositiveと判別

True Negative ：NegativeをNegativeと判別

例：Positive ⇒伝染病に陽性(感染), Negative ⇒感染してない

Positive ⇒生存, Negative ⇒ 死亡

・混同行列を用いた指標の算出

正解率(Accuracy)

(TP +TN)/(TP+FP+FN+TN)

適合率 (Precision)

(TP)/(TP+FP)

見逃しが多くてもより正確な予測をしたい場合に利用

再現率 (Recall)

(TP)/(TP+FN)

誤りが多少多くても抜け漏れを少なくしたい場合

F値 : 適合率 と再現率の調和平均

適合率 と再現率はトレードオフの関係にあるのでバランスを見るのに利用

ハンズオン  
タイタニックデータを利用

乗客の生存・死亡を判定

1. 運賃から予測

# タイタニック

from pandas import DataFrame  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
  
# titanic data csvファイルの読み込み  
titanic\_df = pd.read\_csv('data/titanic\_train.csv')  
# 1. ロジスティック回帰  
# チケット価格から生死を判別  
#運賃だけのリストを作成  
data1 = titanic\_df.loc[:, ["Fare"]].values  
# 生死フラグのみのリストを作成  
label1 = titanic\_df.loc[:,["Survived"]].values  
# 学習データと検証データを分割  
traindata1, testdata1, trainlabel1, testlabel1 = train\_test\_split(data1, label1, test\_size=0.2)  
  
eval\_model1=LogisticRegression()  
predictor\_eval1=eval\_model1.fit(traindata1, trainlabel1).predict(testdata1)  
print("traindata scor:",eval\_model1.score(traindata1, trainlabel1))  
print("testdata score:",eval\_model1.score(testdata1,testlabel1))  
# モデル評価  
from sklearn import metrics  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
confusion\_matrix1=confusion\_matrix(testlabel1, predictor\_eval1)  
print("混同行列:",confusion\_matrix1)  
accuracy\_score1 = accuracy\_score(testlabel1, predictor\_eval1)  
print("正解率 (Accuracy):",accuracy\_score1)

traindata scor: 0.6713483146067416

testdata score: 0.6703910614525139

混同行列: [[104 11]

[ 48 16]]

正解率 (Accuracy): 0.6703910614525139

1. 2変数(年齢,Pclass\_Gender)から生死を判別

# タイタニック  
import pandas as pd  
from pandas import DataFrame  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.linear\_model import LogisticRegression  
  
# titanic data csvファイルの読み込み  
titanic\_df = pd.read\_csv('data/titanic\_train.csv')  
  
# 1. ロジスティック回帰  
# 2変数(年齢,Pclass\_Gender)から生死を判別  
  
# Pclass\_Genderというカラムを作成  
# Pclass 社会階級が高いと値が低い  
# Gender 女性が低い  
# -> 社会階級の高い女性が生存しやすかった。という仮説  
# 年齢はnullを中央値で補完  
# Ageカラムのnullを中央値で補完し、AgeFillカラムを作成  
titanic\_df['AgeFill'] = titanic\_df['Age'].fillna(titanic\_df['Age'].mean())  
titanic\_df['Gender'] = titanic\_df['Sex'].map({'female': 0, 'male': 1}).astype(int)  
titanic\_df['Pclass\_Gender'] = titanic\_df['Pclass'] + titanic\_df['Gender']  
# AgeFill,Pclass\_Genderのリストを作成  
data2 = titanic\_df.loc[:, ["AgeFill", "Pclass\_Gender"]].values  
  
#生死フラグのみのリストを作成  
label2 = titanic\_df.loc[:,["Survived"]].values  
  
# 学習データと検証データを分割  
from sklearn.model\_selection import train\_test\_split  
traindata2, testdata2, trainlabel2, testlabel2 = train\_test\_split(data2, label2, test\_size=0.2)  
eval\_model2=LogisticRegression()  
predictor\_eval2=eval\_model2.fit(traindata2, trainlabel2).predict(testdata2)  
  
print("traindata scor:",eval\_model2.score(traindata2, trainlabel2))  
print("testdata score:",eval\_model2.score(testdata2,testlabel2))  
# モデル評価  
from sklearn import metrics  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
confusion\_matrix2=confusion\_matrix(testlabel2, predictor\_eval2)  
print(confusion\_matrix2)  
print("混同行列:",confusion\_matrix2)  
accuracy\_score2 = accuracy\_score(testlabel2, predictor\_eval2)  
print("正解率 (Accuracy):",accuracy\_score2)

traindata scor: 0.7682584269662921

testdata score: 0.770949720670391

混同行列: [[99 19]

[22 39]]

正解率 (Accuracy): 0.770949720670391

考察

2変数モデルの方が正解率が高かった。

欠損値があるカラムも適切な前処理により活用できることが分かった。

sklearnには正解率のほかに適合率、再現率,F値も

sklearn.metrics.precision\_score

sklearn.metrics.recall\_score

sklearn.metrics.f1\_score

として、計算メソッドが実装されているので必要に応じ容易に算出できる。

主成分分析

相関のある多数の変数から相関のない少数で全体のばらつきを最もよく表す

主成分と呼ばれる変数を合成する多変量解析の一手法。

データの次元を削減するために用いられる。

次元を削減に伴う情報の損失はなるべく小さくする必要がある。

学習データの分散が最大になる方向への線形変換 を求める手法である

少数変数を利用した分析や可視化 (2 ・ 3 次元の場合 が実現可能

⇒ 教師なし学習の際の次元削減の手法とした用いられる。

データを固有値展開し、固有値が大きいものから固有ベクトルを基底として、

データを基底に投影する

⇒ 最大固有値に対応する固有ベクトルで線形変換された特徴量を第一主成分と呼ぶ

⇒ k 番目の固有値に対応する固有ベクトルで変換された特徴量を第 k 主成分と呼ぶ

実装例：

乳がん検査データを用いたロジスティック回帰において主成分分析の効果を試行する。

1. ロジスティック回帰、目的変数は検査データの全カラム

# 乳がん検査データを用いたロジスティック回帰  
import pandas as pd  
from sklearn import datasets  
from sklearn.linear\_model import LogisticRegression  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix  
  
breast\_cancer\_data = datasets.load\_breast\_cancer()  
breast\_cancer\_df = pd.DataFrame(breast\_cancer\_data.data, columns=breast\_cancer\_data.feature\_names)  
# 目的変数の抽出  
breast\_cancer\_df\_tgt = pd.DataFrame(breast\_cancer\_data.target, columns=['target'])  
# 説明変数の抽出  
X = breast\_cancer\_df.loc[:, 'mean radius':]  
# 目的変数  
y = breast\_cancer\_df\_tgt  
# 学習用とテスト用でデータを分離  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)  
  
# 標準化  
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
  
# ロジスティック回帰で学習  
logistic = LogisticRegression()  
logistic.fit(X\_train\_scaled, y\_train)  
# 検証  
print('Train score: {:.3f}'.format(logistic.score(X\_train\_scaled, y\_train)))  
print('Test score: {:.3f}'.format(logistic.score(X\_test\_scaled, y\_test)))

print('Confustion matrix:\n{}'.format(confusion\_matrix(y\_true=y\_test, y\_pred=logistic.predict(X\_test\_scaled))))

Train score: 0.991

Test score: 0.958

Confustion matrix:

[[50 3]

[ 3 87]]

1. ロジスティック回帰、主成分分析で次元数を2にした場合

# 乳がん検査データを用いたロジスティック回帰  
# 主成分分析で次元数を２にした場合  
# import pandas as pd  
from sklearn import datasets  
from sklearn.linear\_model import LogisticRegression  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix  
from sklearn .decomposition import PCA  
  
breast\_cancer\_data = datasets.load\_breast\_cancer()  
breast\_cancer\_df = pd.DataFrame(breast\_cancer\_data.data, columns=breast\_cancer\_data.feature\_names)  
# 目的変数の抽出  
breast\_cancer\_df\_tgt = pd.DataFrame(breast\_cancer\_data.target, columns=['target'])  
# 説明変数の抽出  
X = breast\_cancer\_df.loc[:, 'mean radius':]  
# 目的変数  
y = breast\_cancer\_df\_tgt  
# PCA  
# 次元数2まで圧縮  
pca = PCA(n\_components=2)  
X\_pca = pca.fit\_transform(X)  
  
# 寄与率  
print('explained variance ratio: {}'.format(pca.explained\_variance\_ratio\_))  
  
  
# 学習用とテスト用でデータを分離  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_pca, y, random\_state=0)  
  
# 標準化  
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
  
# ロジスティック回帰で学習  
logistic = LogisticRegression()  
logistic.fit(X\_train\_scaled, y\_train)  
  
# 検証  
print('Train score: {:.3f}'.format(logistic.score(X\_train\_scaled, y\_train)))  
print('Test score: {:.3f}'.format(logistic.score(X\_test\_scaled, y\_test)))  
print('Confustion matrix:\n{}'.format(confusion\_matrix(y\_true=y\_test, y\_pred=logistic.predict(X\_test\_scaled))))

寄与率 explained variance ratio: [0.98204467 0.01617649]

Train score: 0.915

Test score: 0.944

Confustion matrix:

[[47 6]

[ 2 88]]

考察

次元数を2に減らしたみたが、大きく精度を落とすことなく( 0.958に対し0.944)

予測することが出来た。

今回のケースでは次元数が約30個だったため計算時間の削減は無かったが

高次元だと効果が出るだろう。

アルゴリズム

k近傍法

教師あり学習 - 分類問題に用いる手法

・アルゴリズム

予測するデータ点と訓練データの距離を計測し、近いほうからk個の

訓練データを選び、そのラベルの最頻値を割り当てる。

・k=1の場合、最近傍法と呼ぶ。

kの値により結果が変わる。

k を大きくすると決定境界は滑らかになる。

k平均法

教師なし学習のクラスタリングに用いる手法

与えられたデータを k 個のクラスタに分類する

・アルゴリズム

1) 各クラスタ中心の初期値を設定する

2) 各データ点に対して、各クラスタ中心との距離を計算し、

最も距離が近いクラスタを割り当てる

3) 各クラスタの平均ベクトル（中心）を計算する

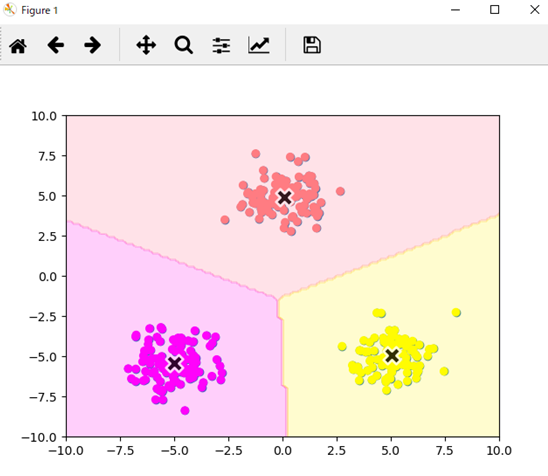
4) 収束するまで 2, 3 の処理を繰り返す

・クラスタ中心中心の初期値を変えるとクラスタリング結果も変わりうる

kの値を変えるとクラスタリング結果も変わる

実装例:ランダムにデータを発生し、クラスタリングする。

# k平均クラスタリング(k-means)  
import numpy as np  
import matplotlib.pyplot as plt  
# データ生成  
x1 = np.random.normal(size=(100, 2)) + np.array([-5, -5])  
x2 = np.random.normal(size=(100, 2)) + np.array([5, -5])  
x3 = np.random.normal(size=(100, 2)) + np.array([0, 5])  
x\_train = np.vstack((x1, x2, x3))  
  
plt.scatter(x\_train[:, 0], x\_train[:, 1])  
  
# 学習  
# k-meansアルゴリズムは以下のとおりである  
# 1) 各クラスタ中心の初期値を設定する  
# 2) 各データ点に対して、各クラスタ中心との距離を計算し、最も距離が近いクラスタを割り当てる  
# 3) 各クラスタの平均ベクトル（中心）を計算する  
# 4) 収束するまで2, 3の処理を繰り返す  
def distance(x1, x2):  
 return np.sum ((x1 - x2) \*\* 2, axis=1)  
  
  
X\_train = x\_train  
  
n\_clusters = 3  
iter\_max = 100  
  
# 各クラスタ中心をランダムに初期化  
centers = X\_train [np.random.choice (len (X\_train), n\_clusters, replace=False)]  
  
for \_ in range (iter\_max):  
 prev\_centers = np.copy (centers)  
 D = np.zeros ((len (X\_train), n\_clusters))  
 # 各データ点に対して、各クラスタ中心との距離を計算  
 for i, x in enumerate (X\_train):  
 D [i] = distance (x, centers)  
 # 各データ点に、最も距離が近いクラスタを割り当  
 cluster\_index = np.argmin (D, axis=1)  
 # 各クラスタの中心を計算  
 for k in range (n\_clusters):  
 index\_k = cluster\_index == k  
 centers [k] = np.mean (X\_train [index\_k], axis=0)  
 # 収束判定  
 if np.allclose (prev\_centers, centers):  
 break  
  
# クラスタリング結果  
y\_pred = np.empty(len(X\_train), dtype=int)  
for i, x in enumerate(X\_train):  
 d = distance(x, centers)  
 y\_pred[i] = np.argmin(d)  
  
xx0, xx1 = np.meshgrid(np.linspace(-10, 10, 100), np.linspace(-10, 10, 100))  
xx = np.array([xx0, xx1]).reshape(2, -1).T  
#　データを可視化  
plt.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_pred, cmap='spring')  
# 中心を可視化  
plt.scatter(centers[:, 0], centers[:, 1], s=200, marker='X', lw=2, c='black', edgecolor="white")  
# 領域の可視化  
pred = np.empty(len(xx), dtype=int)  
for i, x in enumerate(xx):  
 d = distance(x, centers)  
 pred[i] = np.argmin(d)  
plt.contourf(xx0, xx1, pred.reshape(100, 100), alpha=0.2, cmap='spring')  
  
plt.show()



サポートベクターマシーン

教師あり学習の回帰や分類に用いられる手法。2クラス分類に使われることが多い.

2値分類においては、データ点𝑥を𝑦(𝑥)=𝜙(𝑥)+𝑏の正負によって分類する

（ 𝜙(𝑥)は特徴ベクトル）。

マージンを最大化する決定境界（識別面）を求める。

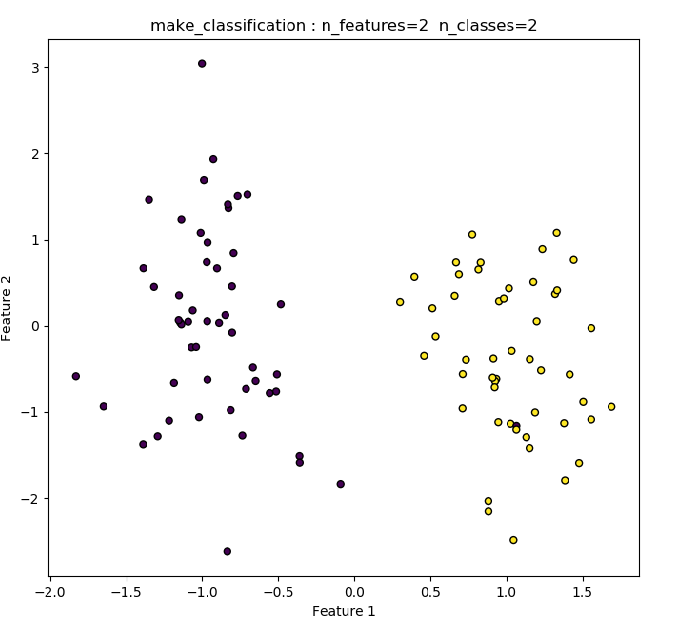
マージン : 決定境界と最も近いデータ点との距離

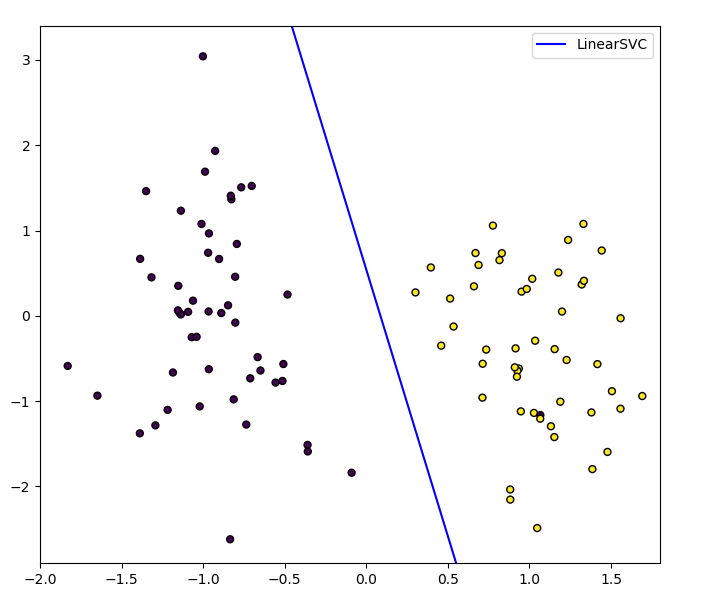
マージン上にあるデータ点をサポートベクターと呼ぶ。

演習1: ランダムなデータを生成し、決定境界を求める。

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error  
from sklearn.metrics import mean\_absolute\_error  
  
# ２クラス用データ生成  
# データ生成  
from sklearn.datasets import make\_classification  
  
X, Y = make\_classification(random\_state=12,  
 n\_features=2,  
 n\_redundant=0,  
 n\_informative=1,  
 n\_clusters\_per\_class=1,  
 n\_classes=2)  
  
fig = plt.figure()  
plt.figure(figsize=(8, 7))  
plt.title("make\_classification : n\_features=2 n\_classes=2")  
plt.scatter(X[:, 0], X[:, 1], marker='o', c=Y, s=25, edgecolor='k')  
plt.xlabel("Feature 1")  
plt.ylabel("Feature 2")  
plt.show()  
  
########################  
# LinearSVC  
########################  
from sklearn.svm import LinearSVC  
  
# トレーニング・テストデータ分割  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, random\_state=0)  
  
# LinearSVC  
linear\_svc = LinearSVC()  
linear\_svc.fit(X\_train, Y\_train)  
  
# 予測　  
Y\_pred = linear\_svc.predict(X\_test)  
  
# 評価  
score = linear\_svc.score(X\_test, Y\_test)  
  
coef = linear\_svc.coef\_[0]  
intercept = linear\_svc.intercept\_  
  
print("score = %.3f" % (score))  
print("Coef =", coef)  
print("Intercept =", intercept)  
  
# プロット  
plt.figure(figsize=(8, 7))  
plt.scatter(X[:, 0], X[:, 1], marker='o', c=Y, s=25, edgecolor='k')  
  
line = np.linspace(-15, 15)  
print("line =",line)  
line2 = -(line \* coef[0] + intercept) / coef[1]  
print("line2 =",line2)  
plt.plot(line, -(line \* coef[0] + intercept) / coef[1], c='b', label="LinearSVC")  
plt.ylim(-2.9, 3.4)  
plt.xlim(-2, 1.8)  
plt.legend()  
plt.show()

データの散布図



得られた決定境界

演習2: タイタニックの生存確率

サポートベクターマシンとロジスティック回帰の比較を行ってみる。

・ロジスティック回帰 - チケット価格から生死を判別

# タイタニック : skl\_logistic\_regression  
import pandas as pd  
from pandas import DataFrame  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
  
# titanic data csvファイルの読み込み  
titanic\_df = pd.read\_csv('data/titanic\_train.csv')  
# ロジスティック回帰  
# チケット価格から生死を判別  
# 運賃だけのリストを作成  
data = titanic\_df.loc[:, ["Fare"]].values  
# 生死フラグのみのリストを作成  
label = titanic\_df.loc[:,["Survived"]].values  
# 学習データと検証データを分割  
traindata, testdata, trainlabel, testlabel = train\_test\_split(data, label, test\_size=0.2)  
  
eval\_model = LogisticRegression()  
predictor\_eval = eval\_model1.fit(traindata, trainlabel).predict(testdata)  
print("traindata scor:",eval\_model.score(traindata, trainlabel))  
print("testdata score:",eval\_model.score(testdata,testlabel))  
# モデル評価  
from sklearn import metrics  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
confusion\_matrix = confusion\_matrix(testlabel, predictor\_eval)  
print("混同行列:",confusion\_matrix)  
accuracy\_score = accuracy\_score(testlabel, predictor\_eval)  
print("正解率 (Accuracy):",accuracy\_score)  
print("report:",metrics.classification\_report(testlabel, predictor\_eval))

正解率 (Accuracy): 0.6480446927374302

・サポートベクターマシン - チケット価格から生死を判別

# タイタニック : skl\_logistic\_regression  
import pandas as pd  
from pandas import DataFrame  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.svm import SVC  
from sklearn.model\_selection import train\_test\_split  
  
# titanic data csvファイルの読み込み  
titanic\_df = pd.read\_csv('data/titanic\_train.csv')  
# ロジスティック回帰  
# チケット価格から生死を判別  
# 運賃だけのリストを作成  
data = titanic\_df.loc[:, ["Fare"]].values  
# 生死フラグのみのリストを作成  
label = titanic\_df.loc[:,["Survived"]].values  
# 学習データと検証データを分割  
traindata, testdata, trainlabel, testlabel = train\_test\_split(data, label, test\_size=0.2)  
  
eval\_model = SVC()  
predictor\_eval = eval\_model.fit(traindata, trainlabel).predict(testdata)  
print("traindata scor:",eval\_model.score(traindata, trainlabel))  
print("testdata score:",eval\_model.score(testdata,testlabel))  
# モデル評価  
from sklearn import metrics  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
confusion\_matrix = confusion\_matrix(testlabel, predictor\_eval)  
print("混同行列:",confusion\_matrix)  
accuracy\_score = accuracy\_score(testlabel, predictor\_eval)  
print("正解率 (Accuracy):",accuracy\_score)  
print("report:",metrics.classification\_report(testlabel, predictor\_eval))

正解率 (Accuracy): 0.7206703910614525

⇒ SVMの方が正解率が良い。

・ロジスティック回帰 - 2変数(年齢,Pclass\_Gender)から生死を判別

# タイタニック  
import pandas as pd  
from pandas import DataFrame  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.linear\_model import LogisticRegression  
  
# titanic data csvファイルの読み込み  
titanic\_df = pd.read\_csv('data/titanic\_train.csv')  
  
# ロジスティック回帰  
# 2変数(年齢,Pclass\_Gender)から生死を判別  
  
# Pclass\_Genderというカラムを作成  
# Pclass 社会階級が高いと値が低い  
# Gender 女性が低い  
# -> 社会階級の高い女性が生存しやすかった。という仮説  
# 年齢はnullを中央値で補完  
# Ageカラムのnullを中央値で補完し、AgeFillカラムを作成  
titanic\_df['AgeFill'] = titanic\_df['Age'].fillna(titanic\_df['Age'].mean())  
titanic\_df['Gender'] = titanic\_df['Sex'].map({'female': 0, 'male': 1}).astype(int)  
titanic\_df['Pclass\_Gender'] = titanic\_df['Pclass'] + titanic\_df['Gender']  
# AgeFill,Pclass\_Genderのリストを作成  
data = titanic\_df.loc[:, ["AgeFill", "Pclass\_Gender"]].values  
  
#生死フラグのみのリストを作成  
label = titanic\_df.loc[:,["Survived"]].values  
  
# 学習データと検証データを分割  
from sklearn.model\_selection import train\_test\_split  
traindata, testdata, trainlabel, testlabel = train\_test\_split(data, label, test\_size=0.2)  
eval\_model=LogisticRegression()  
predictor\_eval=eval\_model.fit(traindata, trainlabel).predict(testdata)  
  
print("traindata scor:",eval\_model.score(traindata, trainlabel))  
print("testdata score:",eval\_model.score(testdata,testlabel))  
# モデル評価  
from sklearn import metrics  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
confusion\_matrix=confusion\_matrix(testlabel, predictor\_eval)  
#print(confusion\_matrix2)  
print("混同行列:",confusion\_matrix)  
accuracy\_score = accuracy\_score(testlabel, predictor\_eval)  
print("正解率 (Accuracy):",accuracy\_score)  
print("report:",metrics.classification\_report(testlabel2, predictor\_eval2))

正解率 (Accuracy): 0.7374301675977654

・SVM- 2変数(年齢,Pclass\_Gender)から生死を判別

# タイタニック  
import pandas as pd  
from pandas import DataFrame  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.svm import SVC  
  
# titanic data csvファイルの読み込み  
titanic\_df = pd.read\_csv('data/titanic\_train.csv')  
  
# ロジスティック回帰  
# 2変数(年齢,Pclass\_Gender)から生死を判別  
  
# Pclass\_Genderというカラムを作成  
# Pclass 社会階級が高いと値が低い  
# Gender 女性が低い  
# -> 社会階級の高い女性が生存しやすかった。という仮説  
# 年齢はnullを中央値で補完  
# Ageカラムのnullを中央値で補完し、AgeFillカラムを作成  
titanic\_df['AgeFill'] = titanic\_df['Age'].fillna(titanic\_df['Age'].mean())  
titanic\_df['Gender'] = titanic\_df['Sex'].map({'female': 0, 'male': 1}).astype(int)  
titanic\_df['Pclass\_Gender'] = titanic\_df['Pclass'] + titanic\_df['Gender']  
# AgeFill,Pclass\_Genderのリストを作成  
data = titanic\_df.loc[:, ["AgeFill", "Pclass\_Gender"]].values  
  
#生死フラグのみのリストを作成  
label = titanic\_df.loc[:,["Survived"]].values  
  
# 学習データと検証データを分割  
from sklearn.model\_selection import train\_test\_split  
traindata, testdata, trainlabel, testlabel = train\_test\_split(data, label, test\_size=0.2)  
eval\_model=SVC()  
predictor\_eval=eval\_model.fit(traindata, trainlabel).predict(testdata)  
  
print("traindata scor:",eval\_model.score(traindata, trainlabel))  
print("testdata score:",eval\_model.score(testdata,testlabel))  
# モデル評価  
from sklearn import metrics  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
confusion\_matrix=confusion\_matrix(testlabel, predictor\_eval)  
#print(confusion\_matrix2)  
print("混同行列:",confusion\_matrix)  
accuracy\_score = accuracy\_score(testlabel, predictor\_eval)  
print("正解率 (Accuracy):",accuracy\_score)  
print("report:",metrics.classification\_report(testlabel2, predictor\_eval2))

正解率 (Accuracy): 0.8044692737430168

⇒ SVMの方が正解率が良い。

考察：

今回はロジスティック回帰モデルよりもサポートベクターマシンの方が

精度が高かった。

ロジスティック回帰モデルのソースとサポートベクターマシンのソースの差異は

2か所 ⇒

from sklearn.svm import SVC と

from sklearn.linear\_model import LogisticRegression

eval\_model= SVC() とeval\_model= LogisticRegression()

　⇒　sklearnを用いると様々なモデルの比較が容易に行える。