機械学習

「第1章：線形回帰モデル」

線形回帰

・未知の値を予測する教師ありの回帰手法

・モデルのパラメータは重みが大きければ、その特徴量は予測に大きく影響

・直線（e.g. y = wx +b）でデータを近似したもの

・入力が多次元ベクトルでも出力は１次元になる

・未知パラメータ（切片、回帰係数）の推定は最小二乗法を使用

＊最尤法でも同じ結果が得られる。

・線形単回帰（m=1）では、与えられたx,yのデータの個数分の連立方程式を解くことで未知パラメータの値が求められるが、計算の簡略化のため式を行列で表現する。

・線形重回帰（m=n次元）では回帰係数wがn個になる

データ分割

・データを分割して、例えば８０％を学習に２０％をモデルの検証に使用する

パラメータ推定

・平均二乗誤差(MAE)の値が小さいほど予測が正しい

・平均二乗誤差の勾配＝０を求めるのが最小二乗法

ハンズオン(scikit-learn)

・１次元もn次元も同じロジック

```
import LinearRegression          ライブラリインポート
model = LinearRegression()
model.fit(data, target)          パラメータ推定、学習
model.predict([[1]])               予測
```

```
Phi = polynomial_features(x_train)
pinv = np.dot(np.linalg.inv(np.dot(Phi.T, Phi)), Phi.T)
w = np.dot(pinv, y_train)
```

・ライブラリなし

線形単回帰：

```
cov = np.cov(x_train, y_train, ddof=0)          …xとyの共分散
a = cov[0, 1] / cov[0, 0]                       …回帰係数 ＝ xyの共分散 ÷ xの分散
b = np.mean(y_train) - a * np.mean(x_train)     …切片 = yの平均 - 回帰係数 × xの平均
```

線形重回帰：

```
def polynomial_features(x, degree=3):
        X = np.ones((len(x), degree+1))
        X_t = X.T
        for i in range(1, degree+1):
                X_t[i] = X_t[i-1] * x
        return X_t.T
```

「第2章：非線形回帰モデル」

・線形で捉えられない場合、非線形を考える

・基底関数（多項式、ガウス型基底 etc.）を選定。パラメータの推定は最小二乗法や最尤法を用いる

・未学習、過学習が起こるが、過学習は正則化で回避

正則化

・複雑なモデルの場合、複雑さに伴ってペナルティ項を大きくする

・L１ノルムを利用したLasso推定量とL２ノルムを利用したRidge推定量がある

モデルの評価と選択

・ホールドアウト法：データを学習用とテスト用に分割し、予測精度や誤謬率を推定する
訓練誤差とテスト誤差のMSEを比較して性能評価

・クロスバリデーション：ホールドアウト法に比べ、データが少ない場合に有効
データをm個に分割してm-1個までを学習に使用し、残りの１個をでモデルのテストをする
クロスバリデーションの値が小さいモデルが良いモデルとなる

ハンズオン

クロスバリデーションはscikit-learnの「from sklearn.metrics import confusion_matrix」を使用

「第3章：ロジスティック回帰モデル」

・教師あり学習、分類

・入力はm次元で、パラメータと線形結合し（線形回帰と同じ）、シグモイド関数にかけて四捨五入で0 or 1を出力する

・シグモイド関数：0 〜1の値をを出力、シグモイド関数の微分はシグモイド関数で表現可能

最尤推定

・データを固定してパラメータを変化させパラメータの分布が特定の値であることがどれほどあるか

・尤度関数：xを与えた時Y=1になる確率、尤度関数を最大にするパラメータを推定

・複数のxに対しY=1になる確率を求めて掛け算し同時確率を求める

・尤度関数の微分は小数点以下が膨大になり計算が難しいので、対数尤度で対数同士の足し算、掛け算にする。計算を楽にするため、対数尤度関数にマイナスをかけて最小化を目指す。

勾配下降法

・対数尤度関数をパラメータで微分して０になる値を求めるのは解析学的に困難なので、最急下降法を使用する。学習率を設定する。

・確率的勾配法
最急下降法はすべてのパラメータに対し計算するので、データが多いと時間とメモリをくってしまう
データを一つづつランダムに選んでパラメータを更新していく。ただし勾配＝０になることは稀。

・ミニバッチ確率的勾配法
データを区切ってパラメータの更新に使用する。50から100に区切ることが多い。

分類モデルの評価

Confusion Matrix:

・正解率…分類したいクラスに偏りがあると正解率はあまり意味がない

・適合率…見逃しが多くてもより正確な予測をしたい場合

ハンズオン(scikit-learn)

```
model = LogisticRegression()
model.fit(data, label)                  学習
model.predict(data)                     予測
model.predict_proba(data)               確率表示
```

ライブラリなし：

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))                    …シグモイド関数

X_train = add_one(x_train)                         …学習データセット

max_iter=100                                       …学習回数の設定
eta = 0.01                                         …学習率の設定
w = np.zeros(X_train.shape[1])                     …０で配列初期化、Xと同じ形に
for _ in range(max_iter):
    w_prev = np.copy(w)
    sigma = sigmoid(np.dot(X_train, w))            …確率算出
    grad = np.dot(X_train.T, (sigma - y_train))    …勾配算出
    w -= eta * grad                                …パラメータ更新
    if np.allclose(w, w_prev):
        break
```

「第4章：主成分分析」

・大きい次元数のデータを情報量を保ちながら低次元に変換する、情報の質を落とさないための手法

・分散が最大(=0)になる軸を探す。分散共分散行列が対称行列なので、軸はすべて直行する

・手法：分散共分散行列の最大固有値に対応する固有ベクトルを求める

・n個のデータを係数ベクトルを用いて線形変換する

・寄与率：主成分分析で情報を圧縮したときに、どれくらい情報を保持または損失しているかの指標

・累積寄与率：現場でよく使用。それぞれの寄与率を累積する

ハンズオン

```
n_components=2
mean = X.mean(axis=0)                                  …確率変数の平均
cov = np.dot((X - mean).T, X - mean) / (len(X) - 1)    …共分散
eigenvalues, eigenvectors = np.linalg.eigh(cov)        …共分散の固有値、固有ベクトルを求める
W = eigenvectors[:, -n_components:]
principal_components = W.T[::-1]
```

「第5章：アルゴリズム」

k近傍法(kNN)

・教師あり学習、分類。高次元データには向かない

・k個のクラスラベルの中で、最も近いラベルを割り当てる。ラベルを増やすと決定境界線が滑らかに

・最近傍法はk-nn1のk=0と同じ

・実装：

```
def distance(x1, x2):
        return np.sum((x1 - x2)**2, axis=1)                    …ユークリッド距離を求める

X_train = x_train
X_test = xx
k = 3                                                          …ラベルの個数

y_pred = np.empty(len(X_test), dtype=y_train.dtype)
for i, x in enumerate(X_test):
        distances = distance(x, X_train)
        nearest_index = distances.argsort()[:k]                …ソートした配列のインデックスを返却
        mode, _ = stats.mode(y_train[nearest_index])           …最も多く存在する値
        y_pred[i] = mode
```

k平均法(k-means)

・教師なし学習、クラスタリング

・手法：

（1）クラスタ中心点の初期値をk個割り当て

（2）各データがどの中心点に最も近いか計算し割り当てる

（3）各クラスタの平均の中心点を計算

（4）収束するまで

・実装：

```
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)                           …最小要素のインデックスを取得
# 各クラスタの中心を計算
```

```
        centers[k] = np.mean(X_train[index_k], axis=0)
        # 収束判定
        if np.allclose(prev_centers, centers):                    …２つの配列の全ての要素が完全に一致
            break
```

「第7章：サポートベクターマシーン」

・教師あり学習、分類にも回帰にも使用

・決定境界線で正負の２値分類を行う。マージンが最も遠くなる境界線を決定する

・境界線に近いデータ（サポートベクター）のみが学習に使われる

・ソフトマージンSVM

綺麗にデータを分類できないとき、誤差を許容する。誤差にペナルティを付与

ペナルティの値が小さいときはより誤差を許容、大きい時は誤差を許容しない

・カーネルトリック

複雑な非線形回帰をするときに使用。複雑な空間を特異空間に変換し線形分離できる

jupyter skl_regression Last Checkpoint: 06/13/2018 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Markdown

```
Out[13]: array([[6.575],
                [6.421],
                [7.185],
                [6.998],
                [7.147]])

In [14]: # 目的変数
         target = df.loc[:, 'PRICE'].values

In [15]: target[0:5]

Out[15]: array([24. , 21.6, 34.7, 33.4, 36.2])

In [16]: ## sklearnモジュールからLinearRegressionをインポート
         from sklearn.linear_model import LinearRegression

In [17]: # 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)

In [18]: # fit関数でパラメータ推定
         model.fit(data, target)

Out[18]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                          normalize=False)

In [31]: #予測
         model.predict([[1]])

Out[31]: array([-25.5685118])
```

重回帰分析(2変数)

線形単回帰

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Markdown

```
In [22]: # 説明変数
         data2 = df.loc[:, ['CRIM', 'RM']].values
         # 目的変数
         target2 = df.loc[:, 'PRICE'].values

In [23]: # オブジェクト生成
         model2 = LinearRegression(fit_intercept = True, normalize = False, copy_X = True, n_jobs = 1)

In [24]: # fit関数でパラメータ推定
         model2.fit(data2, target2)

Out[24]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [25]: model2.predict([[2, 3]])

Out[25]: array([-4.60134121])
```

回帰係数と切片の値を確認

```
In [32]: # 単回帰の回帰係数と切片を出力
         print('推定された回帰係数: %.3f, 推定された切片 : %.3f' % (model.coef_, model.intercept_))

         推定された回帰係数: 9.102, 推定された切片 : -34.671

In [33]: # 重回帰の回帰係数と切片を出力
         print(model2.coef_)
         print(model2.intercept_)

         [-0.26491325  8.39106825]
         -29.24471945192995
```

線形重回帰

jupyter skl_regression Last Checkpoint: 06/13/2018 (unsaved changes)

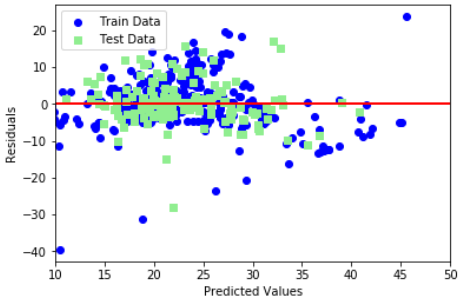
Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

```
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()
```



In [34]:

```
# 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('MSE Train : %.3f, Test : %.3f' % (mean_squared_error(y_train, y_train_pred), mean_squared_error(y_test, y_test_p
# 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)))
```

```
MSE Train : 44.983, Test : 40.412
R^2 Train : 0.500, Test : 0.434
```

In [ ]:

線形回帰　検証

jupyter skl_logistic_regression Last Checkpoint: 03/12/2019 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

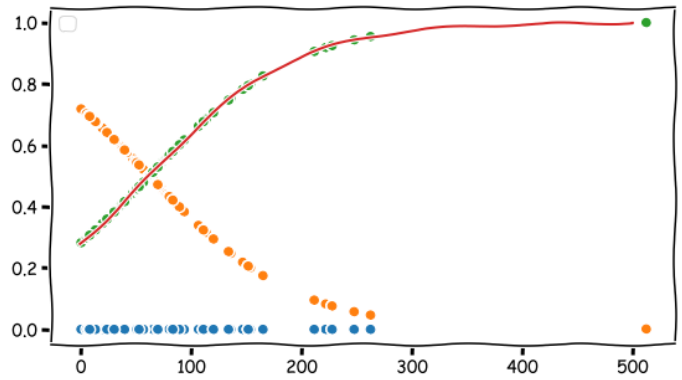
Trusted Python 3

Run Markdown

```
plt.plot(data1, model.predict_proba(data1), 'o')
plt.plot(x_range, sigmoid(x_range), '-')
#plt.plot(x_range, normal_sigmoid(x_range), '-')
#
```

```
No handles with labels found to put in legend.
```

Out[22]: [<matplotlib.lines.Line2D at 0x1a1edd49b0>]



1. ロジスティック回帰

実装(2変数から生死を判別)

ロジスティック回帰

jupyter skl_logistic_regression (unsaved changes)

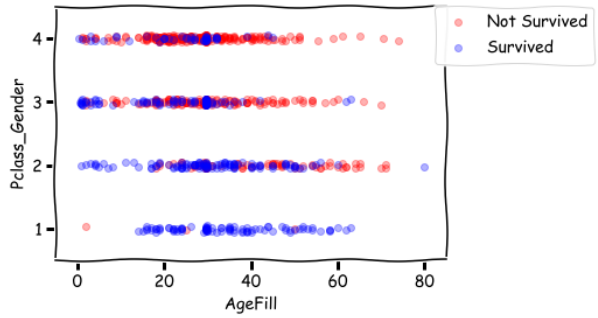
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Markdown

```
sc = ax.scatter(titanic_df.loc[index_notsurvived, 'AgeFill'],
                titanic_df.loc[index_notsurvived, 'Pclass_Gender']+(np.random.rand(len(index_notsurvived))-0.5)*0.1,
                color='b', label='Survived', alpha=0.3)
ax.set_xlabel('AgeFill')
ax.set_ylabel('Pclass_Gender')
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)
ax.legend(bbox_to_anchor=(1.4, 1.03))
```

Out[34]: <matplotlib.legend.Legend at 0x1a171dccf8>



In [35]:
```
#運賃だけのリストを作成
data2 = titanic_df.loc[:, ["AgeFill", "Pclass_Gender"]].values
```

In [36]:
```
#生死フラグのみのリストを作成
label2 =  titanic_df.loc[:,["Survived"]].values
```

In [37]:
```
model2 = LogisticRegression()
```

ロジスティック回帰

jupyter skl_logistic_regression (autosaved)

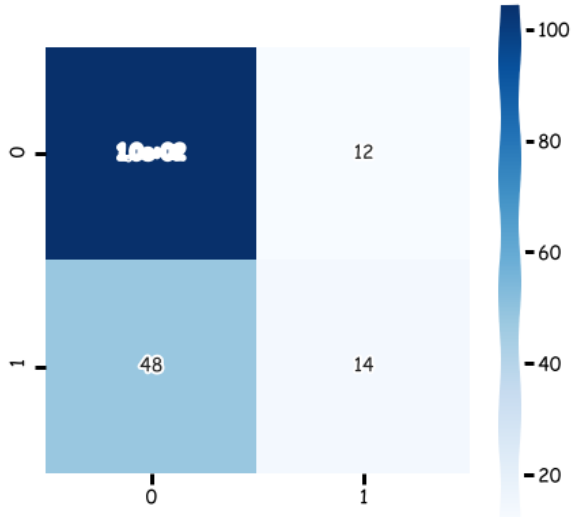
Edit View Insert Cell Kernel Widgets Help

Not Trusted

Run Markdown

```
    cbar_ax=None,
    square=True, ax=None,
    #xticklabels=columns,
    #yticklabels=columns,
    mask=None)
```

Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x10d5e5978>



In [57]:
```
fig = plt.figure(figsize = (7,7))
#plt.title(title)
sns.heatmap(
    confusion_matrix2,
```

クロスバリデーション

jupyter np_pca (unsaved changes)

Logout

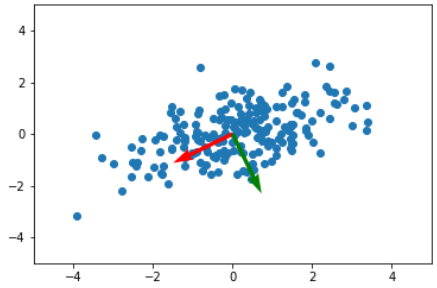
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Markdown

```
plt.xlim(-5, 5)
plt.ylim(-5, 5)
# 第1主成分
plt.quiver(0, 0, first[0], first[1], width=0.01, scale=6, color='red')
# 第2主成分
plt.quiver(0, 0, second[0], second[1], width=0.01, scale=6, color='green')
```

Out[5]: <matplotlib.quiver.Quiver at 0x113e81630>



変換（射影）

元のデータをm次元に変換(射影)するときは行列Wを$W = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_m]$とし、データ点$\boldsymbol{x}$を$\boldsymbol{z} = W^{\mathrm{T}}\boldsymbol{x}$によって変換(射影)する。

よって、データXに対しては$Z = X^{\mathrm{T}}W$によって変換する。

In [6]:
```
Z = np.dot(X - mean, W)
```

In [7]:
```
plt.scatter(Z[:, 0], Z[:, 1])
```

主成分分析

jupyter np_knn (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Markdown

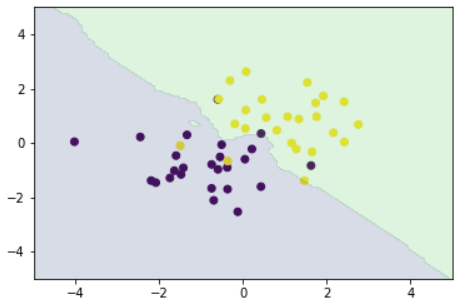
In [5]:
```
def distance(x1, x2):
    return np.sum((x1 - x2)**2, axis=1)

X_train = x_train
X_test = xx
k = 3

y_pred = np.empty(len(X_test), dtype=y_train.dtype)
for i, x in enumerate(X_test):
    distances = distance(x, X_train)
    nearest_index = distances.argsort()[:k]
    mode, _ = stats.mode(y_train[nearest_index])
    y_pred[i] = mode
```

In [6]:
```
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
plt.contourf(xx0, xx1, y_pred.reshape(100, 100).astype(dtype=np.float), alpha=0.2, levels=np.linspace(0, 1, 3))
```

Out[6]: <matplotlib.contour.QuadContourSet at 0x117cddc18>



In [ ]:

K近傍法

jupyter np_kmeans (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

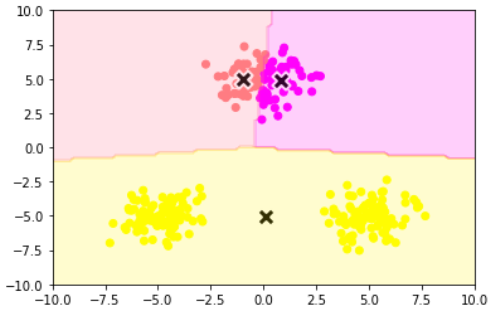
Run Markdown

```
for i, x in enumerate(X_train):
    d = distance(x, centers)
    y_pred[i] = np.argmin(d)
```

In [6]:
```
xx0, xx1 = np.meshgrid(np.linspace(-10, 10, 100), np.linspace(-10, 10, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T
```

In [7]:
```
# 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')
```

Out[7]: <matplotlib.contour.QuadContourSet at 0x1114b0668>



In [ ]:

K平均法