+ コード + テキスト 接続

ロジスティック回帰

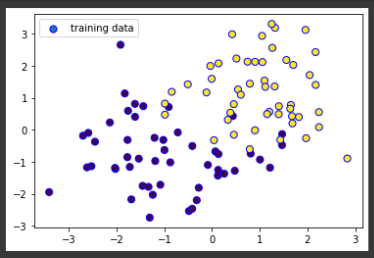
```
[ ]  1 %matplotlib inline
     2 import numpy as np
     3 import matplotlib.pyplot as plt
```

訓練データ生成

```
[ ]  1 n_sample = 100
     2 harf_n_sample = 50
     3 var = .2
     4
     5 #データ生成
     6 #「2行1列の-1よりデータ」と 「2行1列の+1よりデータ」の結合データ及び、その結果値データを作成
     7 def gen_data(n_sample, harf_nsample):
     8     x0 = np.random.normal(size=n_sample).reshape(-1, 2) - 1.
     9     x1 = np.random.normal(size=n_sample).reshape(-1, 2) + 1.
    10     x_train = np.concatenate([x0, x1])
    11     y_train = np.concatenate([np.zeros(harf_n_sample), np.ones(harf_n_sample)]).astype(np.int)
    12     return x_train, y_train
    13 #データプロット関数
    14 def plt_data(x_train, y_train):
    15     plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, facecolor="none", edgecolor="b", s=50, label="training data")
    16     plt.legend()
```

```
[ ]  1 n_sample = 100
     2 x0 = np.random.normal(size=n_sample).reshape(-1, 2) - 1.
     3 x1 = np.random.normal(size=n_sample).reshape(-1, 2) + 1.
     4 x_train = np.concatenate([x0, x1])
     5 x_train
```

```
[ ]  1 #データ作成
     2 x_train, y_train = gen_data(n_sample, harf_n_sample)
     3 #データ表示
     4 plt_data(x_train, y_train)
```



ロジスティック回帰モデル

識別モデルとして$p(y=1|\boldsymbol{x};\boldsymbol{w})=\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})$を用いる。

ただし、$\sigma(\cdot)$はシグモイド関数であり、$\sigma(h)=\frac{1}{1+\exp(-h)}$で定義される。

また、陽には書かないが、$\boldsymbol{x}$には定数項のための1という要素があることを仮定する。

+ コード + テキスト 接続

学習

訓練データ$X=[\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_n]^{\mathrm{T}},\boldsymbol{y}=[y_1,y_2,\ldots,y_n]^{\mathrm{T}}(y_i=\{0,1\})$に対して尤度関数$L$は以下のように書ける。

$L(\boldsymbol{w})=\prod_{i=1}^{n}p(y_i=1|\boldsymbol{x}_i;\boldsymbol{w})^{y_i}(1-p(y_i=1|\boldsymbol{x}_i;\boldsymbol{w}))^{1-y_i}$

負の対数尤度関数は

$-\log L(\boldsymbol{w})=-\sum_{i=1}^{n}\left[y_i\log p(y_i=1|\boldsymbol{x}_i;\boldsymbol{w})+(1-y_i)\log\left(1-p(y_i=1|\boldsymbol{x}_i;\boldsymbol{w})\right)\right]$

のように書ける。 これを最小化する$\boldsymbol{w}$を求める。

$\frac{d\sigma(h)}{dh}=\sigma(h)(1-\sigma(h))$と書けることを考慮し、負の対数尤度関数を$\boldsymbol{w}$で偏微分すると、

$$\frac{\partial}{\partial\boldsymbol{w}}(-\log L(\boldsymbol{w}))=-\sum_{i=1}^{n}\left[y_i(1-\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i))-(1-y_i)\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i)\right]\boldsymbol{x}_i$$

$$=\sum_{i=1}^{n}\left(\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i)-y_i\right)\boldsymbol{x}_i$$

この式が0となる$\boldsymbol{w}$は解析的に求められないので、今回は$-\log L(\boldsymbol{w})$の最小化問題を最急降下法を用いて解く。

最急降下法では学習率をηとすると、以下の式で$\boldsymbol{w}$を更新する。

$\boldsymbol{w}\leftarrow\boldsymbol{w}-\eta\frac{\partial}{\partial\boldsymbol{w}}(-\log L(\boldsymbol{w}))$

```
[ ]  1 def add_one(x):
     2     return np.concatenate([np.ones(len(x))[:, None], x], axis=1)
```

```
    return np.concatenate([np.ones(len(x))[:, None], x], axis=1)
```

```
def sigmoid(x):
  return 1 / (1 + np.exp(-x))

#stochastic gradient decent
#max_iter分学習
def sgd(X_train, max_iter, eta):
  w = np.zeros(X_train.shape[1])
  for _ in range(max_iter):
    w_prev = np.copy(w)
    sigma = sigmoid(np.dot(X_train, w))
    grad = np.dot(X_train.T, (sigma - y_train))
    w -= eta * grad

    if np.allclose(w, w_prev):
      return w

  return w


X_train = add_one(x_train)
max_iter=100
eta = 0.01
w = sgd(X_train, max_iter, eta)

```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: RuntimeWarning: divide by zero encountered in true_divide
```

```
###★★  np.allcloseとnp.isclosehа別途再調査
# import numpy as np
# a = np.array([0.1, 0.2])
# b = np.array([0.1, 0.2])
# np.allclose(a,b)
```

+ コード + テキスト　　接続

```
# a = np.array([0.1, 0.2])
# b = np.array([0.1, 0.2])
# np.allclose(a,b)
```

予測

入力に対して、$y = 1$である確率を出力する。よって

$p(y = 1|\boldsymbol{x}; \boldsymbol{w}) = \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})$の値が

0.5より大きければ1に、小さければ0に分類する。

```
##################################
##meshgrid()→第一引数を第二引数行数分（今回は100行）に拡張展開
#           →第二引数を第二引数列数分（今回は100列）に拡張展開
#縦横軸の作成に便利？
##################################
xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T
```

```
X_test = add_one(xx)
proba = sigmoid(np.dot(X_test, w))
y_pred = (proba > 0.5).astype(np.int)
```

```
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
plt.contourf(xx0, xx1, proba.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))

```

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

#データ生成
#「2行1列の-1よりデータ」と 「2行1列の+1よりデータ」の結合データ及び、その結果値データを作成
def gen_data(n_sample, harf_nsample):
    x0 = np.random.normal(size=n_sample).reshape(-1, 2) - 1.
    x1 = np.random.normal(size=n_sample).reshape(-1, 2) + 1.
    x_train = np.concatenate([x0, x1])
    y_train = np.concatenate([np.zeros(harf_n_sample), np.ones(harf_n_sample)]).astype(np.int)
    return x_train, y_train

#データプロット関数
def plt_data(ax1,x_train, y_train):
```

+ コード + テキスト　　接続

```
def plt_data(ax1,x_train, y_train):
    ax1.scatter(x_train[:, 0], x_train[:, 1], c=y_train, facecolor="none", edgecolor="b", s=50, label="training data")
    ax1.legend()


#データ作成
x_train, y_train = gen_data(n_sample, harf_n_sample)
#データ表示
fig = plt.figure()
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
plt_data(ax1, x_train, y_train)


#skl実装（学習）
from sklearn.linear_model import LogisticRegression
```

```
model=LogisticRegression(fit_intercept=True)
model.fit(x_train, y_train)

#予測データ
xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

#skl実装（予測）
proba = model.predict_proba(xx)
y_pred = (proba > 0.5).astype(np.int)


ax2.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
ax2.contourf(xx0, xx1, proba[:, 0].reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))

```

```
<matplotlib.contour.QuadContourSet at 0x7f1ab8be9d90>
```