

Homework 4

Lecturer: Jiapeng Liu

Scribe: 吴博成

Student Number: 2193211134

1 HW 1

1.1 问题重述

证明西瓜书上公式

$$\ell(\beta) = \sum_{i=1}^n (-y_i \beta^T \mathbf{x}_i + \log(1 + \exp(\beta^T \mathbf{x}_i)))$$

关于 β 的梯度 $\nabla \ell$ 和 Hessian 矩阵 \mathbf{H} 分别为

$$\nabla \ell = \mathbf{X}^T (\boldsymbol{\mu} - \mathbf{y})$$

$$\mathbf{H} = \mathbf{X}^T \mathbf{S} \mathbf{X}$$

其中

$$\begin{aligned} \mathbf{X} &= [\mathbf{x}_1, \dots, \mathbf{x}_n]^T \\ \boldsymbol{\mu} &= (\mu_1, \dots, \mu_n)^T \\ \mathbf{y} &= (y_1, \dots, y_n)^T \\ \mathbf{S} &= \text{diag}(\mu_1(1 - \mu_1), \dots, \mu_n(1 - \mu_n)) \\ \mu_i &= \frac{1}{1 + \exp(-\beta^T \mathbf{x}_i)}, i = 1, \dots, n \end{aligned}$$

进一步证明矩阵 \mathbf{H} 是正定矩阵。

1.2 问题求解

先求解梯度 $\ell(\beta)$ 关于 β 的梯度 $\nabla \ell$, 即 $\frac{\partial \ell(\beta)}{\partial \beta}$:

$$\begin{aligned} \nabla \ell &= \sum_{i=1}^n -y_i \mathbf{x}_i + \sum_{i=1}^n \frac{\mathbf{x}_i \exp(\beta^T \mathbf{x}_i)}{1 + \exp(\beta^T \mathbf{x}_i)} \\ &= \sum_{i=1}^n -y_i \mathbf{x}_i + \sum_{i=1}^n \frac{\mathbf{x}_i}{1 + \exp(-\beta^T \mathbf{x}_i)} \\ &= -\mathbf{X}^T \mathbf{y} + \sum_{i=1}^n \mathbf{x}_i \mu_i \\ &= \mathbf{X}^T (\boldsymbol{\mu} - \mathbf{y}) \end{aligned}$$

再求解 Hessian 阵, 即求解 $\nabla^2 l(\beta) = \frac{\partial^2 l}{\partial \beta^2}$:

$$\begin{aligned} \mathbf{H} &= \frac{\partial \mathbf{X}^T (\boldsymbol{\mu} - \mathbf{y})}{\partial \beta} \\ &= \frac{\partial \mathbf{X}^T \boldsymbol{\mu}}{\partial \beta} \\ &= \mathbf{X}^T \nabla \boldsymbol{\mu} \\ &= \mathbf{X}^T \sum_{i=1}^n \frac{\mathbf{x}_i \exp(-\boldsymbol{\beta}^T \mathbf{x}_i)}{[1 + \exp(-\boldsymbol{\beta}^T \mathbf{x}_i)]^2} \\ &= \mathbf{X}^T \sum_{i=1}^n \mathbf{x}_i (\mu_i (1 - \mu_i)) \\ &= \mathbf{X}^T \mathbf{S} \mathbf{X} \end{aligned}$$

最后证明矩阵 \mathbf{H} 是正定矩阵, 若要证明矩阵 \mathbf{H} 是正定矩阵, 则要证明:
对于任意的 n 维非零向量 \mathbf{z} , 都有

$$\mathbf{z}^T \mathbf{H} \mathbf{z} = \mathbf{z}^T \mathbf{X}^T \mathbf{S} \mathbf{X} \mathbf{z} = (\mathbf{X} \mathbf{z})^T \mathbf{S} \mathbf{X} \mathbf{z} > 0$$

将 $\mathbf{X} \mathbf{z}$ 记为 $\mathbf{A} = [a_1, a_2, \dots, a_n]$, 则有:

$$\mathbf{z}^T \mathbf{H} \mathbf{z} = \mathbf{A}^T \mathbf{S} \mathbf{A} = \sum_{i=1}^n \mu_i (1 - \mu_i) a_i^2$$

因为 $\mu_i \in (0, 1), a_i^2 > 0$,

所以 $\mu_i (1 - \mu_i) a_i^2 > 0$, 所以 \mathbf{H} 是正定矩阵。

2 HW 2

2.1 问题重述

对于 Breast Cancer Wisconsin 数据集, 根据上述公式, 用 Python 实现梯度下降法和牛顿法估计 Logistic Regression 模型的参数 β 。进一步对比计算得到的结果与作业 2 中由 sklearn 算出的结果之间的异同。

2.2 问题求解

```
1 # %% [markdown]
2 # # Section 1 : 数据预处理
3
4 # %%
5 import pandas as pd
6 import numpy as np
7 # 首先将缺失值中? 表示替换为nan
8 data = pd.read_csv("breast-cancer-wisconsin.data", names=["Sample code number", "Clump Thickness", "
    Uniformity of Cell Size", "Uniformity of Cell Shape", "Marginal Adhesion", "Single Epithelial Cell Size", "Bare
    Nuclei", "Bland Chromatin", "Normal Nucleoli", "Mitoses", "Class"])
```

```

9 data.replace("?", np.nan , inplace = True)
10 data["Bare Nuclei"] = data["Bare Nuclei"].astype("float")
11 cleaned_data = data.dropna()
12 normalized_data = pd.DataFrame()
13 def Standard_Score(arr):
14     mean = np.mean(arr)
15     std = np.std(arr)
16     return ((arr - mean)/std)
17 for (columnName, columnData) in cleaned_data.iteritems():
18     normalized_data[columnName] = Standard_Score(columnData)
19 normalized_data["Sample code number"] = cleaned_data["Sample code number"]
20 normalized_data["Class"] = cleaned_data["Class"]
21 normalized_data.to_csv("normalized-breast-cancer-wisconsin.data",header=False,index=False,sep=',')
22 display(normalized_data)
23
24 X = normalized_data.iloc[:,1:-1]
25 #Class 一列中, 4表明患癌症, 2表示不患癌症。转化成1-0
26 y = normalized_data["Class"].replace([4,2],[1,0]).values
27
28 # %% [markdown]
29 # # Section 2 : 分割训练集和测试集
30 # 逻辑回归通过定义 $\beta = \begin{bmatrix} w \\ b \end{bmatrix}$ ,  $\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$ , 将  $\mathbf{\omega}^{\text{top}} \mathbf{x} + b$  简写为  $\mathbf{\beta}^{\text{top}} \hat{\mathbf{x}}$ , 所以通过给 $\mathbf{X}$ 加一列全1向量变成增广矩阵 $\hat{\mathbf{X}}$ 
31
32 # %%
33 X_hat = np.concatenate((X.values,np.ones([X.shape[0],1])),axis = 1)
34 from sklearn.model_selection import train_test_split
35 X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=20)
36 X_hat_train = np.concatenate((X_train.values,np.ones([X_train.shape[0],1])),axis = 1)
37 X_hat_test = np.concatenate((X_test.values,np.ones([X_test.shape[0],1])),axis = 1)
38
39 # %% [markdown]
40 # # Section 3 : 梯度下降法估计Logistic Regression模型的参数 $\beta$ 
41 # $$
42 # \ell(\boldsymbol{\beta}) = \sum_{i=1}^m \left( -y_i \boldsymbol{\beta}^{\text{T}} \hat{\boldsymbol{x}}_i + \ln \left( 1 + e^{\boldsymbol{\beta}^{\text{T}} \hat{\boldsymbol{x}}_i} \right) \right)
43 # $$
44 # $$
45 # \boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\arg \min } \ell(\boldsymbol{\beta})
46 # $$
47
48 # %% [markdown]
49 # ### 对于梯度下降法而言:
50 # $$

```

```

51 # \boldsymbol{\beta}^{t+1}=\boldsymbol{\beta}^t- s \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}
52 #
53 # \nabla \ell = \hat{\mathbf{X}}^{\text{top}}(\boldsymbol{\mu}-\mathbf{y})
54 # $$
55 # ### 其中:
56 # $$\boldsymbol{\mu}=\left(\mu_1, \ldots, \mu_n\right)^{\mathrm{T}}$$
57 # $$
58 # \mu_i=\frac{1}{1+\exp \left(-\boldsymbol{\beta}^{\mathrm{T}} \hat{\mathbf{x}}_i\right)}, i=1, \ldots, n$$
59
60 # %%
61 #梯度下降法
62 def target_function(X_hat,beta,y):
63     result = 0
64     for i in range(X_hat.shape[0]):
65         result += -y[i]* beta.T @ X_hat[i,:] + np.log( 1+ np.exp(beta.T @ X_hat[i,:]))
66     return result
67
68 def mu(X_hat,beta,y):
69     mu = []
70     for i in range(X_hat.shape[0]):
71         mu.append(float(1/(1+np.exp(-beta.T @ X_hat[i,:]))))
72     mu = np.array(mu)
73     return mu
74
75 def gradient(X_hat,beta,y):
76     m = mu(X_hat,beta,y)
77     return X_hat.T @ (m - y)
78
79 def Gradient_Decent(X_hat,y,step,eps):
80     beta = np.zeros(X_hat.shape[1])
81     t=0 #计数器
82     err = np.inf
83     while err > eps and t < 1e6:
84         original_lx = target_function(X_hat,beta,y)
85         beta=beta-step*gradient(X_hat,beta,y)
86         err = abs(target_function(X_hat,beta,y) - original_lx)
87         t += 1
88     return beta,t,target_function(X_hat,beta,y)
89 GD_beta,GD_t,GD_target = Gradient_Decent(X_hat_train , y_train , step = 1e-2, eps = 1e-6)
90 print( "使用梯度下降法迭代次数为:{t} \n目标函数最优值为: {target} \n最优解为: {beta}\n".format(t = GD_t,
    target = GD_target,beta = GD_beta))
91
92 # %% [markdown]
93 # # Section 4 : 牛顿法估计Logistic Regression模型的参数$\beta$

```

```

94 # $$
95 # \ell(\boldsymbol{\beta})=\sum_{i=1}^m\left(-y_{i} \boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_{i}+\ln \left(1+\mathrm{e}^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_{i}}\right)\right)
96 # $$
97 # $$
98 # \boldsymbol{\beta}^*=\underset{\boldsymbol{\beta}}{\arg \min } \ell(\boldsymbol{\beta})
99 # $$
100 # ### 对于牛顿法而言:
101 # $$
102 # \boldsymbol{\beta}^{t+1}=\boldsymbol{\beta}^t-\left(\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathrm{T}}}\right)^{-1} \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}=\boldsymbol{\beta}^t-\mathbf{H}^{-1} \nabla \ell
103 # $$
104 # ### 其中:
105 # $$\boldsymbol{\mu}=\left(\mu_1, \ldots, \mu_n\right)^{\mathrm{T}}$$
106 # $$
107 # \mu_i=\frac{1}{1+\exp \left(-\boldsymbol{\beta}^{\mathrm{T}} \hat{\mathbf{x}}_i\right)}, i=1, \ldots, n$$
108 # $$\nabla \ell=\hat{\mathbf{X}}^{\mathrm{T}}(\boldsymbol{\mu}-\mathbf{y})$$
109 # $$\mathbf{H}=\hat{\mathbf{X}}^{\mathrm{T}} \mathbf{S} \hat{\mathbf{X}}$$
110 # $$\mathbf{S}=\operatorname{diag}\left(\mu_1\left(1-\mu_1\right), \ldots, \mu_n\left(1-\mu_n\right)\right)$$
111
112 # %%
113 def Hessian(X_hat,beta,y):
114     S = np.zeros([X_hat.shape[0],X_hat.shape[0]])
115     m = mu(X_hat,beta,y)
116     for i in range(X_hat.shape[0]):
117         S[i,i] = m[i]*(1-m[i])
118     return X_hat.T @ S @ X_hat
119
120 def Newton_Method(X_hat,y,eps):
121     beta = np.zeros(X_hat.shape[1])
122     t=0 #计数器
123     err = np.inf
124     while err > eps and t < 1e6:
125         original_lx = target_function(X_hat,beta,y)
126         beta=beta - np.linalg.inv(Hessian(X_hat,beta,y)) @ gradient(X_hat,beta,y)
127         err = abs(target_function(X_hat,beta,y) - original_lx)
128         t += 1
129     return beta,t,target_function(X_hat,beta,y)
130
131 newton_beta,newton_t,newton_target = Newton_Method(X_hat_train , y_train , eps = 1e-6)
132 print( "使用牛顿法迭代次数为:{t} \n目标函数最优值为: {target} \n最优解为: {beta}\n".format(t = newton_t,
    target = newton_target,beta = newton_beta))

```

```

133
134 # %% [markdown]
135 # # Section 5 : Sklearn 求解 Logistic Regression 模型的参数  $\beta$ 
136
137 # %%
138 from sklearn.linear_model import LogisticRegression
139 logistic_regression = LogisticRegression().fit(X_train,y_train)
140 sklearn_beta = np.concatenate((logistic_regression.coef_.flatten(),logistic_regression.intercept_))
141 print("目标函数最优值为: {target} \n使用sklearn模型得到的参数beta为: {beta}\n".format(target =
    target_function(X_hat,sklearn_beta,y),beta = sklearn_beta))
142
143 # %% [markdown]
144 # # Section 6 : 对比各种回归方式异同
145 # 将梯度下降法和牛顿法得到的系数放入sklearn对象中，方便使用sklearn中函数进行对比
146
147 # %%
148 gradient_decent = LogisticRegression().fit(X_train,y_train)
149 gradient_decent.coef_ = GD_beta[:-1].reshape(1,GD_beta.shape[0]-1)
150 gradient_decent.intercept_ = GD_beta[-1]
151 newton = LogisticRegression().fit(X_train,y_train)
152 newton.coef_ = newton_beta[:-1].reshape(1,newton_beta.shape[0]-1)
153 newton.intercept_ = newton_beta[-1]
154
155
156 # %% [markdown]
157 # ### 对比回归系数  $\beta$ 
158
159 # %%
160 print("GD方法系数: {gd}\nNewton方法系数: {nt}\nSklearn方法系数: {sk}\n".format(gd = GD_beta,nt =
    newton_beta,sk = sklearn_beta))
161
162 # %% [markdown]
163 # ### 对比准确率
164
165 # %%
166 GD_score = gradient_decent.score(X_train,y_train) #由于eps设置的小，所以测试集算出准确率相同
167 newton_score = newton.score(X_train,y_train)
168 sk_score = logistic_regression.score(X_train,y_train)
169 print("GD方法准确率: {gd}\nNewton法准确率: {nt}\nSklearn方法准确率: {sk}\n".format(gd = GD_score,nt =
    newton_score,sk = sk_score))
170
171 # %% [markdown]
172 # ### 结论
173 # 由上述结论可知，模型跟迭代次数有一定关系:
174 #
175 # 牛顿法的目标函数值是最小的，效果是最好的。

```

```
176 #  
177 # sklearn的目标函数值是最大的，效果最差。  
178 #
```

可以看到，使用牛顿法和梯度下降法求解的结果基本一致，而这两个与 sklearn 的结果有所不同。总的来说，几种方法结果的差距不算很大，各特征回归系数的相对大小基本一致。

详见附件 Homework4.ipynb