Optimization Theory and Algorithm II **Homework 4 - 13/11/2021**

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

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

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

$$\begin{aligned}\nabla\ell&=\mathbf{X}^{\mathrm{T}}(\boldsymbol{\mu}-\mathbf{y})\\ \mathbf{H}&=\mathbf{X}^{\mathrm{T}}\mathbf{S}\mathbf{X}\end{aligned}$$

其中

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

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 $\ell(\boldsymbol{\beta})$ 关于 $\boldsymbol{\beta}$ 的梯度 $\nabla\ell$, 即 $\frac{\partial\ell(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}}$:

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

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

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

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

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

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

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

因为 $\mu_i\in(0,1), a_i{}^2>0$,
所以 $\mu_i\left(1-\mu_i\right)a_i{}^2>0$, 所以 $\boldsymbol{H}$ 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 $\boldsymbol{\beta}$ 。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

```python
# %% [markdown]
# # Section 1 : 数据预处理

# %%
import pandas as pd
import numpy as np
# 首先将缺失值中? 表示替换为nan
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"])
```

```
data.replace("?", np.nan , inplace = True)
data["Bare Nuclei"] = data["Bare Nuclei"].astype("float")
cleaned_data = data.dropna()
normalized_data = pd.DataFrame()
def Standard_Score(arr):
    mean = np.mean(arr)
    std = np.std(arr)
    return ((arr - mean)/std)
for (columnName, columnData) in cleaned_data.iteritems():
    normalized_data[columnName] = Standard_Score(columnData)
normalized_data["Sample code number"] = cleaned_data["Sample code number"]
normalized_data["Class"] = cleaned_data["Class"]
normalized_data.to_csv("normalized-breast-cancer-wisconsin.data",header=False,index=False,sep=',')
display(normalized_data)

X = normalized_data.iloc[:,1:-1]
#Class 一列中，4表明患癌症，2表示不换癌症。转化成1-0
y = normalized_data["Class"].replace([4,2],[1,0]) .values

# %% [markdown]
# # Section 2：分割训练集和测试集
# 逻辑回归通过定义$\beta = \begin{bmatrix}w\\b \end{bmatrix},\hat{\mathbf{x}}=\begin{bmatrix}\mathbf{x}\\\mathbb{1}\end{bmatrix}$ ， 将 $\mathbf{ \omega}^\top \mathbf{x} +{ b}$ 简写为 $\mathbf{\beta}^\top\hat{\mathbf{x}}$, 所以通过给$\mathbf{X}$加一列全1向量变成增广矩阵$\hat{\mathbf{X}}$

# %%
X_hat = np.concatenate((X.values,np.ones([X.shape[0],1])),axis = 1)
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=20)
X_hat_train = np.concatenate((X_train.values,np.ones([X_train.shape[0],1])),axis = 1)
X_hat_test = np.concatenate((X_test.values,np.ones([X_test.shape[0],1])),axis = 1)

# %% [markdown]
# # Section 3：梯度下降法估计Logistic Regression模型的参数$\beta$
# $$
# \ell(\boldsymbol{\beta})=\sum_{i=1}^{m}\left(-y_{i} \boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_{i}+\ln \left(1+e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_{i}}\right)\right)
# $$
# $$
# \boldsymbol{\beta}^{*}=\underset{\boldsymbol{\beta}}{\arg \min } \ell(\boldsymbol{\beta})
# $$

# %% [markdown]
# ### 对于梯度下降法而言：
# $$
```

```python
# \boldsymbol{\beta}^{t+1}=\boldsymbol{\beta}^{t}- s \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\\
#
# \nabla \ell =\hat{\mathbf{X}}^{\top}(\boldsymbol{\mu}-\mathbf{y})
# $$
# ### 其中:
# $$\boldsymbol{\mu}=\left(\mu_{1}, \ldots, \mu_{n}\right)^{\mathrm{T}}$$
# $$
# \mu_{i}=\frac{1}{1+\exp \left(-\boldsymbol{\beta}^{\mathrm{T}} \hat{\mathbf{x}_{i}}\right)}, i=1, \ldots, n$$

# %%
#梯度下降法
def target_function(X_hat,beta,y):
    result = 0
    for i in range(X_hat.shape[0]):
        result += -y[i]* beta.T @ X_hat[i,:] + np.log( 1+ np.exp(beta.T @ X_hat[i,:]))
    return result

def mu(X_hat,beta,y):
    mu = []
    for i in range(X_hat.shape[0]):
        mu.append(float(1/(1+np.exp(-beta.T @ X_hat[i,:]))))
    mu = np.array(mu)
    return mu

def gradient(X_hat,beta,y):
    m = mu(X_hat,beta,y)
    return X_hat.T @ (m - y)

def Gradient_Decent(X_hat,y,step,eps):
    beta = np.zeros(X_hat.shape[1])
    t=0 #计数器
    err = np.inf
    while err > eps and t < 1e6:
        original_lx = target_function(X_hat,beta,y)
        beta=beta-step*gradient(X_hat,beta,y)
        err = abs(target_function(X_hat,beta,y) - original_lx)
        t += 1
    return beta,t,target_function(X_hat,beta,y)
GD_beta,GD_t,GD_target = Gradient_Decent(X_hat_train , y_train , step = 1e-2, eps = 1e-6)
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))

# %% [markdown]
# # Section 4 : 牛顿法估计Logistic Regression模型的参数$\beta$
```

```python
# $$
# \ell(\boldsymbol{\beta})=\sum_{i=1}^{m}\left(-y_{i} \boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_{i}+\ln \left(1+e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_{i}}\right)\right)
# $$
# $$
# \boldsymbol{\beta}^{*}=\underset{\boldsymbol{\beta}}{\arg \min } \ell(\boldsymbol{\beta})
# $$
# ### 对于牛顿法而言:
# $$
# \boldsymbol{\beta}^{t+1}=\boldsymbol{\beta}^{t}-\left(\frac{\partial^{2} \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathrm{T}}}\right)^{-1} \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}=\boldsymbol{\beta}^{t} - \mathbf{H} \nabla \ell
# $$
# ### 其中:
# $$\boldsymbol{\mu}=\left(\mu_{1}, \ldots, \mu_{n}\right)^{\mathrm{T}}$$
# $$
# \mu_{i}=\frac{1}{1+\exp \left(-\boldsymbol{\beta}^{\mathrm{T}} \hat{\mathbf{x}_{i}}\right)}, i=1, \ldots, n$$
# $$\nabla \ell=\hat{\mathbf{X}}^{\mathrm{T}}(\boldsymbol{\mu}-\mathbf{y})$$
# $$\mathbf{H}=\hat{\mathbf{X}}^{\mathrm{T}} \mathbf{S} \hat{\mathbf{X}}$$
# $$\mathbf{S}=\operatorname{diag}\left(\mu_{1}\left(1-\mu_{1}\right), \ldots, \mu_{n}\left(1-\mu_{n}\right)\right) $$

# %%
def Hessian(X_hat,beta,y):
    S = np.zeros([X_hat.shape[0],X_hat.shape[0]])
    m = mu(X_hat,beta,y)
    for i in range(X_hat.shape[0]):
        S[i,i] = m[i]*(1-m[i])
    return X_hat.T @ S @ X_hat

def Newton_Method(X_hat,y,eps):
    beta = np.zeros(X_hat.shape[1])
    t=0 #计数器
    err = np.inf
    while err > eps and t < 1e6:
        original_lx = target_function(X_hat,beta,y)
        beta=beta - np.linalg.inv(Hessian(X_hat,beta,y)) @ gradient(X_hat,beta,y)
        err = abs(target_function(X_hat,beta,y) - original_lx)
        t += 1
    return beta,t,target_function(X_hat,beta,y)

newton_beta,newton_t,newton_target = Newton_Method(X_hat_train , y_train , eps = 1e-6)
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))
```

```python

# %% [markdown]
# # Section 5 : Sklearn 求解 Logistic Regression 模型的参数 $\beta$

# %%
from sklearn.linear_model import LogisticRegression
logistic_regression = LogisticRegression().fit(X_train,y_train)
sklearn_beta = np.concatenate((logistic_regression.coef_.flatten(),logistic_regression.intercept_))
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))

# %% [markdown]
# # Section 6 : 对比各种回归方式异同
# 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

# %%
gradient_decent = LogisticRegression().fit(X_train,y_train)
gradient_decent.coef_ = GD_beta[:-1].reshape(1,GD_beta.shape[0]-1)
gradient_decent.intercept_ = GD_beta[-1]
newton = LogisticRegression().fit(X_train,y_train)
newton.coef_ = newton_beta[:-1].reshape(1,newton_beta.shape[0]-1)
newton.intercept_ = newton_beta[-1]


# %% [markdown]
# ### 对比回归系数 $ \beta$

# %%
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))

# %% [markdown]
# ### 对比准确率

# %%
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
newton_score = newton.score(X_train,y_train)
sk_score = logistic_regression.score(X_train,y_train)
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))

# %% [markdown]
# ### 结论
# 由上述结论可知，模型跟迭代次数有一定关系:
#
# 牛顿法的目标函数值是最小的，效果是最好的。
```

```
#
# sklearn的目标函数值是最大的，效果最差。
#
```

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