Introduction to Machine Learning
Homework 2

181860155 朱晓晴

2020 年 10 月 20 日

1 [30pts] Multi-Label Logistic Regression

In multi-label problem, each instance $\boldsymbol{x}$ has a label set $\boldsymbol{y} = \{y_1, y_2, ..., y_L\}$ and each label $y_i \in \{0, 1\}, \forall 1 \le i \le L$. Assume the post probability $p(\boldsymbol{y} \mid \boldsymbol{x})$ follows the conditional independence:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{i=1}^{L} p(y_i \mid \boldsymbol{x}). \tag{1.1}$$

Please use the logistic regression method to handle the following questions.
(1) [15pts] Please give the log-likelihood function of your logistic regression model;
(2) [15pts] Please calculate the gradient of your log-likelihood function and show the parameters updating step using gradient descent.

解：(1)$\boldsymbol{x_i}$ 表示第 i 个示例，$\boldsymbol{y_i}$ 表示 $\boldsymbol{x_i}$ 的标记集合，y_{ij} 表示 $\boldsymbol{y_i}$ 中的第 j 个标记。对数似然函数为

$$\begin{aligned}
&\ell(\boldsymbol{w}, b) = \sum_{i=1}^{m} \ln p(\boldsymbol{y}_i | \boldsymbol{x}_i; \boldsymbol{w}, b) \\
&= \sum_{i=1}^{m} \ln \prod_{j=1}^{L} p(y_{ij} | \boldsymbol{x}_i; \boldsymbol{w}_j, b_j) \\
&= \sum_{i=1}^{m} \sum_{j=1}^{L} \ln p(y_{ij} | \boldsymbol{x}_i; \boldsymbol{w}_j, b_j)
\end{aligned}$$

令 $\boldsymbol{\beta}_j = (\boldsymbol{w}_j; b_j)$，$\hat{\boldsymbol{x}}_i = (\boldsymbol{x}_i; 1)$，有

$\ln p(y_{ij}|\boldsymbol{x}_i;\boldsymbol{w}_j,b_j)$

$= \ln[y_{ij}p_1(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}_j)+(1-y_{ij})p_0(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}_j)]$

$= y_{ij}\boldsymbol{\beta}_j^{\mathrm{T}}\hat{\boldsymbol{x}}_i - \ln(1+e^{\boldsymbol{\beta}_j^{\mathrm{T}}\hat{\boldsymbol{x}}_i})$

因此，
$\ell(\boldsymbol{\beta}) = \sum\limits_{i=1}^{m}\sum\limits_{j=1}^{L}[y_{ij}\boldsymbol{\beta}_j^{\mathrm{T}}\hat{\boldsymbol{x}}_i - \ln(1+e^{\boldsymbol{\beta}_j^{\mathrm{T}}\hat{\boldsymbol{x}}_i})]$

(2) 梯度为

$\frac{\partial\ell(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}_j} = \sum\limits_{i=1}^{m}(y_{ij}\hat{\boldsymbol{x}}_i - \frac{\hat{\boldsymbol{x}}_i e^{\boldsymbol{\beta}_j^{\mathrm{T}}\hat{\boldsymbol{x}}_i}}{1+e^{\boldsymbol{\beta}_j^{\mathrm{T}}\hat{\boldsymbol{x}}_i}})$
$\bigtriangledown\ell(\boldsymbol{\beta}) = [\frac{\partial\ell(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}_1},...,\frac{\partial\ell(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}_L}]$

梯度下降法的更新公式为
$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t - \alpha^t \bigtriangledown \ell(\boldsymbol{\beta})$
其中，α^t 为该步的学习率。

2 [70pts] Logistic Regression from scratch

Implementing algorithms is a good way of understanding how they work in-depth. In case that you are not familiar with the pipeline of building a machine learning model, this article can be an example (link).

In this experiment, you are asked to build a classification model on one of UCI data sets, Letter Recognition Data Set (click to download). In particular, the objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The detailed statistics of this data set is listed in Table 1. The data set was then randomly split into train set and test set with proportion 7 : 3. Also, letters from 'A' to 'Z' are mapped to digits '1' to '26' respectively as represented in the last column of the provided data set.

表 1: Statistics of the data set.

Property	Value	Description
Number of Instances	20,000	Rows of the data set
Number of Features	17	Columns of the data set
Number of classes	26	Dimension of the target attribute

In order to build machine learning models, you are supposed to implement Logistic Regression (LR) algorithm which is commonly used in classification tasks. Specifically, in this experiment, you have to adapt the traditional binary class LR method to tackle the multi-class learning problem.

(1) [**10pts**] You are encouraged to implement the code using *Python3* or *Matlab*, implementations in any other programming language will not be graded. Please name the source code file (which contains the main function) as *LR_main.py* (for python3) or *LR_main.m* (for matlab). Finally, your code needs to print the testing performance on the provided test set once executed.

(2) [**30pts**] Functions required to implement:

- Implement LR algorithm using gradient descent or Newton's method.

- Incorporate One-vs-Rest (OvR) strategy to tackle multi-class classification problem.

(3) [**30pts**] Explain implementation details in your submitted report (source code should not be included in your PDF report), including optimization details and hyper-parameter settings, etc. Also, testing performance with respect to Accuracy, Precision, Recall, and F_1 score should be reported following the form of Table 2.

表 2: Performance of your implementation on test set.

Performance Metric	Value (%)
accuracy	00.00
micro Precision	00.00
micro Recall	00.00
micro F_1	00.00
macro Precision	00.00
macro Recall	00.00
macro F_1	00.00

NOTE: Any off-the-shelf implementations of LR or optimization methods are **NOT ALLOWED** to use. When submitting your code and report, all files should be placed in the same directory (without any sub-directory).

实验报告

1 算法介绍

1.1 总体思路

本次实验要求使用 Logistic Regression 算法实现分类模型，用于辨别 26 个大写字母。训练部分，首先处理训练集数据，再根据 OvR 策略构建 26 个分类器，并在每个分类器中进行学习。测试部分，首先处理测试集数据，然后在每个分类器中预测测试样例的分类。

1.2 核心算法

核心算法为 OvR 策略和牛顿法。相较于梯度下降法，牛顿法收敛速度更快，且不存在“之”字形下降的问题，因此本次实验选择采用牛顿法进行优化。

OvR

OvR 每次将一个字母所在类作为正例，其他 25 个字母作为反例，共训练 26 个分类器。预测时，考虑各分类器结果的预测置信度，选择置信度最大的类别标记作为分类结果。

牛顿法

选用的损失函数为

$$J(\boldsymbol{\beta}) = -\frac{1}{m}\sum_{i=1}^{m}(y_i\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i - \ln(1+e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}))$$

损失函数的梯度（一阶导数）为

$$\bigtriangledown J(\boldsymbol{\beta}) = \frac{1}{m}\sum_{i=1}^{m}\hat{\boldsymbol{x}}_i(p_1(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}) - y_i)$$

Hessian 矩阵（二阶导数）为

$$H = \frac{1}{m}\sum_{i=1}^{m}\hat{\boldsymbol{x}}_i\hat{\boldsymbol{x}}_i^{\mathrm{T}}p_1(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta})(1-p_1(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}))$$

牛顿法第 $t+1$ 轮迭代解的更新公式为

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^{t} - H^{-1} \bigtriangledown J(\boldsymbol{\beta})$$

牛顿法的伪代码如下：

1.$\boldsymbol{\beta}$ 初始为全零向量；

2. 计算梯度和 Hessian 矩阵，并计算出新的 $\boldsymbol{\beta}$ 值；

3. 重复第 2 步（迭代）若干次。

2 代码实现

2.1 语言

本次实验的代码部分使用 Python 3 完成，存放于 LR_main.py 中。

2.2 模块划分

代码实现中封装了 LRClassifier 类，成员变量的含义如下：

trainSet：训练集数据

dataMatrix：训练集特征数据（每个示例 16 个特征）

label：训练集示例标记

weights：26 个分类器训练出的结果

testSet：测试集数据

testDataMatrix：测试集特征数据

testResult：测试集示例标记

成员函数的功能如下：

normalize(testDataMat)：对 testDataMat 矩阵中的数据进行归一化处理。

getDataSet()：读取训练集和测试集数据，分离示例特征和标记，并对示例特征进行归一化处理。

OvR()：对每个分类器，计算出相应的训练集标记，并进行学习，得到的结果存放入 weights 中。

newtonMethod(X, y, iteration)：X 为训练集特征数据，y 为分类器对应的训练集标记，iteration 为指定迭代次数。该函数使用牛顿法得出某一分类器的结果。

classify()：利用训练出的模型，对每个测试样例进行预测，最后给出分类模型的性能指标。

3 实验结果

使用牛顿法进行优化时，需要调节的参数仅为迭代次数。迭代次数为 5 次时，模型的性能指标如下：

表 3: 模型性能指标（迭代 5 次）

Performance Metric	Value (%)
accuracy	69.97
micro Precision	69.97
micro Recall	69.97
micro F_1	69.97
macro Precision	70.56
macro Recall	70.07
macro F_1	69.22

迭代次数为 10 次时，模型的性能指标如下：

表 4: 模型性能指标（迭代 10 次）

Performance Metric	Value (%)
accuracy	70.77
micro Precision	70.77
micro Recall	70.77
micro F_1	70.77
macro Precision	71.50
macro Recall	70.85
macro F_1	70.27

当迭代次数达到 10 次以上时，上述性能指标没有显著提升。因此，本次实验选择的迭代次数为 10 次，性能指标如表 4 所示。

参考文献

[1] 周志华. 机器学习 [M]. 清华大学出版社, 2016.

[2] 数据标准化/归一化
https://www.cnblogs.com/pejsidney/p/8031250.html

[3] 牛顿法在逻辑回归中的使用
https://blog.csdn.net/m0_37393514/article/details/82708285

[4] 多分类测评标准
https://www.cnblogs.com/wuxiangli/p/10478097.html