CS 571 Homework 1

Yifan Fei

Computer Science
Purdue University

Instructor: Prof. Bruno Ribeiro

Spring 2018

Section 0:

1

Yes, I discussed the homework with others but came up with my own answers. Their name(s) are Yupeng Han and Meng Liu.

2

I have used online resources to help me answer this question, but I came up with my own answers. Here is a list of the websites I have used in this homework:

Euclidean distance: https://en.wikipedia.org/wiki/Euclidean_distance

Logistic Regression from sklearn: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Bayesian linear regression: https://en.wikipedia.org/wiki/Bayesian_linear_regression

Class weight: https://stackoverflow.com/questions/30972029/how-does-the-class-weight-parameter-in-scikit-learn-work

Section 1:

1:

Accuracy of positive class is about 0.030487804878 and accuracy of negative class is 0.999901517683, which is larger than that of positive class. There are 284807 rows in the csv file, but 0.1727% of the class is class 1(positive class), and 99.8273% of the class is class 0(negative class). The sample number of positive class is much smaller than that of negative class.

Bayes's theorem can explain this. Bayes's theorem is strongly influenced by priors since it describes a positive relationship between posterior and prior probability. If you start with a very low prior, even in the face of strong evidence the posterior probability will be closer to the prior (lower). Since the dataset number of positive class is much smaller than that of negative class, the "belief" that only 0.1727% of the class is positive, is the strong reason for a low posterior positive accuracy.

2:Two effective solutions

1. Make the number of test data of positive class and negative class balanced. In this case, we can up-sample Minority Class(class 1) or down-sample Majority Class(class 0) in training.
2. Try different penalized models. Here, increasing the C value can have less powerful regularization, and more accuracy on positive class.
3. Use weighted data as in subquestion 4 and 6

3:

C is inverse of regularization strength. Smaller C values specify stronger regularization. When C is bigger, say we set $C = 0.1$, the accuracy of positive class increases to 0.607723577236, while the accuracy of negative class goes down

a little to 0.999873379878. When C is smaller, say we set $C = 10^{-12}$, Accuracy of positive class is 0.030487804878, and accuracy of negative class is 1.0. The table below can shown this accuracy change.

Table 1: **Accuracy with different C value**

C	0.1	10^{-5}	10^{-8}	10^{-12}
Positive accuracy	0.607723577236	0.227642276423	0.030487804878	0.030487804878
Negative accuracy	0.999873379878	0.999475933384	0.999901517683	1.0

Relationship between C and σ^2:

In lecture 7, slide 17:

$$\hat{\beta} = argmin_{\beta} \sum_{i=1}^{n} (y_i - \boldsymbol{x_i^T \beta})^2 + \frac{\sum_{j=1}^{p} {\beta_j}^2}{2\sigma^2}$$

Check the sklearn official file, as an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

Divided by C does not change the cost function:

$$\min_{w,c} \frac{1}{2C} w^T w + \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

So we can see:

$$C = \sigma^2$$

Use Bayes theorem to explain:

From the table we can see that positive class is affected more by the regularization than the negative class. Bayes's theorem is strongly influenced by priors since it describes a positive relationship between posterior and prior probability. If you start with a very low prior, even in the face of strong evidence the posterior probability will be closer to the prior (lower).

Since the dataset number of positive class is much smaller than that of negative class, the "belief" that only 0.1727% of the class is positive, is the strong reason for a low posterior positive accuracy. By increasing C value, regulation is less stronger and overfitting are more likely to happen. This benefits positive class more since the sample space is smaller.

4:

Correcting for bias in the likelihood function by automatically adjust weights inversely proportional to class frequencies in the input data. Let π_i be the probability of sampling example i in the training data, here i equals to 0 or 1.

$$\pi_i = \frac{sample\, number\, of\, class\, i}{total\, number\, of\, sample}$$

$$logp(t \mid w) = \sum_i \frac{1}{\pi_i} (t_i logy(\phi_i) + (1t_i) log(1 - y(\phi_i))$$

Then we can use the gradient:

$$\bigtriangledown_w logp(t \mid w) = \sum_i \frac{1}{\pi_i}(t_i - y(\phi_i))\phi_i$$

The second derivative (Hessian):

$$H = \Phi^T R \Phi$$

where $R_i = \frac{1}{\pi_i} y(\phi_i)(1 - y(\phi_i))$

and then we can use the iterative parameter update (Newton-Raphson) to update similarly with unweighted method.

5:

If I use class weight defined in subquestion 4(Let π_i be the probability of sampling example i in the training data), the result is(C = 1e-8):

Accuracy of positive class: 0.989837398374

Accuracy of negative class: 0.315962928442

If I use class weight as 'balanced'(C = 1e-8):

Accuracy of positive class: 0.993902439024

Accuracy of negative class: 0.195445192832

The difference is that in sklearn official file, the 'balanced' weight is defined as $n_{samples}/(n_{classes} * np.bincount(y))$, but we actually ignore class number and just use $n_{samples}/np.bincount(y))$. If we add class number in the denominator, the results should be the same.

```
# hw1q1_5.py
import pandas as pd

train_df = pd.read_csv('creditcard.csv')
grouped = train_df.groupby(train_df['Class'])
class0 = grouped.get_group(0) # 284315
class1 = grouped.get_group(1) # 492
num0, num1 = class0.shape[0], class1.shape[0]
print(num0, num1)
weight = {0:(num0+num1)/num0, 1:(num0+num1)/num1}
X_train = train_df.drop(['Class', 'Time'], axis=1)
Y_train = train_df["Class"]

from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
# Logistic Regression
logreg = LogisticRegression(penalty='l2',C=1e-8,class_weight = weight)
logreg.fit(X_train, Y_train)
pred_train = logreg.predict(X_train)
# print(round(logreg.score(X_train, Y_train) * 100, 2)) # print score

CM = confusion_matrix(Y_train, pred_train)
print("Accuracy of positive class:",CM[1,1]/(CM[1,0]+CM[1,1]))
print("Accuracy of negative class:",CM[0,0]/(CM[0,0]+CM[0,1]))
```

6:

Rather than using objective function $argmin_{w,b}\{\frac{1}{2}w^2 + C\sum_{y_i=-1,1}\xi_i\}$ The new objective function should be:

$$argmin_{w,b}\{\frac{1}{2}w^2 + C_1\sum_{y_i=+1}\xi_i + C_2\sum_{y_i=-1}\xi_i\}$$

By Lagrange method, we should minimize:

$$L(a) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(x_m, x_n)$$

S.T.

$$0 \leq a_n^+ \leq C_1$$
$$0 \leq a_n^- \leq C_2$$
$$\sum_{n=1}^{N} a_n t_n = 0$$

where a_n^+, a_n^- represent the Lagrangian multipliers of positive and negative examples, respectively. This dual optimization problem can be solved in the same way as solving the normal SVM optimization problem.

you can choose constants C_1 and C_2 inversely proportional to the class sizes. That is, if you have l_1 training samples in class 1 and l_2 – in class 2, take C_1 and C_2 such that $\frac{C_1}{C_2} = \frac{l_2}{l_1}$. In that case, the minority class uses a higher misclassification penalty.

Section2: Linear regression with priors

1

$$\varepsilon \, to \, be \, \boldsymbol{N}(0, \lambda \boldsymbol{I})$$

So $y_i - \beta_0 1 - \beta_1 x_{i1} - ... - \beta_p x_{ip} = y_i - \boldsymbol{x_i^T}\boldsymbol{\beta}$ is also gussian distribution with

$$N(0, \lambda)$$

which means:

$$P(y_i \mid x_i, \beta) = \frac{1}{(\sqrt{2\pi}\lambda)} e^{-\frac{(y_i - \boldsymbol{x_i^T}\boldsymbol{\beta})^2}{2\lambda^2}}$$

$$P(y \mid x, \beta) = \frac{1}{(2\pi\lambda)^{n/2}} e^{\sum_{i=1}^{n} -\frac{(y_i - \boldsymbol{x_i^T}\boldsymbol{\beta})^2}{2\lambda^2}}$$

2

$$\boldsymbol{\beta}\, to\, be\, \boldsymbol{N}(0, \sigma \boldsymbol{I})$$

So

$$P(\beta) = \frac{1}{(\sqrt{2\pi}\sigma)} e^{(-\frac{\beta^T \beta}{2\sigma^2})}$$

$$P(\beta \mid y) = \frac{P(y \mid \beta, P(\beta)}{P(y)}$$

Maximum a posterior estimate:

$$\hat{\beta} = argmax_\beta \frac{1}{(2\pi\lambda)^{n/2}} e^{\sum_{i=1}^{n} -\frac{(y_i - \boldsymbol{x_i^T \beta})^2}{2\lambda}} \frac{1}{(\sqrt{2\pi}\sigma)} e^{(-\frac{\beta^T \beta}{2\sigma^2})}$$

$$\hat{\beta} = argmax_\beta log \frac{1}{(2\pi\lambda)^{n/2}} e^{\sum_{i=1}^{n} -\frac{(y_i - \boldsymbol{x_i^T \beta})^2}{2\lambda}} \frac{1}{(\sqrt{2\pi}\sigma)} e^{(-\frac{\beta^T \beta}{2\sigma^2})}$$

$$\hat{\beta} = argmax_\beta \sum_{i=1}^{n} -\frac{(y_i - \boldsymbol{x_i^T \beta})^2}{2\lambda} - \frac{\beta^T \beta}{2\sigma^2}$$

$$\hat{\beta} = argmin_\beta \sum_{i=1}^{n} \frac{(y_i - \boldsymbol{x_i^T \beta})^2}{2\lambda} + \frac{\sum_{j=1}^{p} {\beta_j}^2}{2\sigma^2}$$

3:Decision Boundary and Classification

The decision boundary between the two classes:

$$y = (x - \mu_-)^T (x - \mu_-) - (x - \mu_+)^T (x - \mu_+)$$

So that when x is near μ_+, y>0, and when x is near μ_-, y<0.

Then by simplifying:

$$y = \mu_-^T \mu_- - 2x^T \mu_- - \mu_+^T \mu_+ + 2x^T \mu_+$$
$$y = 2x^T (\mu_+ - \mu_-) + (\mu_-^T \mu_- - \mu_+^T \mu_+)$$

Compared with $y = w^T x + b$, we can get:

$$w = 2(\mu_+ - \mu_-)$$
$$b = \mu_-^T \mu_- - \mu_+^T \mu_+$$