Papan Yongmalwong

13 Followers About Follow

Logistic Regression From Scratch with Gradient Descent and Newton's Method

Papan Yongmalwong May 20, 2019 · 3 min read

Logistic Regression From Scratch with Gradient Descent and Newton's Method

Click here to open colaboratory

Introduction to Logistic Regression

From Statistics Point of View

Given explanatory matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ and response vector $\mathbf{Y} \in \mathbb{R}^{n}$

where element $x_{ij} \in \mathbf{X}$ takes any real value and element $y_i \in \mathbf{Y}$ takes only binary value (0,1)

$\mathbf{X}$				$\mathbf{Y}$
x_{11}	x_{12}	$\cdots$	x_{1p}	y_1

⋮	⋮	⋱	⋮	⋮
x_{n1}	x_{n2}	⋯	x_{np}	y_n

The goal is to find a model that is able to predict the probability of $y^* = 1$ given new explanatory variable $\mathbf{x}^* = \begin{bmatrix} x_1^* & x_2^* & \dots & x_p^* \end{bmatrix}^T$.

The model that is suitable for this problem is the logit model where we treat each response variable Y_i, conditioned on explanatory variable $\mathbf{x}_i = \begin{bmatrix} x_{i1} & x_{i2} & \dots & x_{ip} \end{bmatrix}^T$ as a random variable such that $Y_i|\mathbf{x}_i \sim \text{Bernoulli}(p_i)$ where $p_i = \Pr(Y_i = 1|\mathbf{x}_i) = \frac{e^{\mathbf{x}_i^T\boldsymbol{\beta}}}{1+e^{\mathbf{x}_i^T\boldsymbol{\beta}}}$ and that $Y_i|\mathbf{x}_i$'s are independent, for every row $i = 1, \dots, n$.

It is noteworthy that $Y_i|\mathbf{x}_i$'s are random variables, $\mathbf{x}_i$'s are $p \times 1$ constant vectors and $\boldsymbol{\beta}$ is an unknown $p \times 1$ vector.

The goal then becomes solving for coefficients $\boldsymbol{\beta}$ that satisfies the logit model assumption on $\mathbf{X}$ and $\mathbf{Y}$.

Solving for $\boldsymbol{\beta}$ can be done by maximum likelihood estimation.

$$\widehat{\boldsymbol{\beta}} = \text{argmax}_{\boldsymbol{\beta}}\, \mathcal{L}(\boldsymbol{\beta})$$

$$\mathcal{L}(\boldsymbol{\beta}) = \Pr(Y_1 = y_1, Y_2 = y_2, \dots, Y_n = y_n|\mathbf{X})$$

$$= \prod_{i=1}^{n} \Pr(Y_i = y_i|\mathbf{x}_i)$$

$$= \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}$$

$$= \prod_{i=1}^{n} \left(\frac{e^{\mathbf{x}_i^T \boldsymbol{\beta}}}{1 + e^{\mathbf{x}_i^T \boldsymbol{\beta}}} \right)^{y_i} \left(\frac{1}{1 + e^{\mathbf{x}_i^T \boldsymbol{\beta}}} \right)^{1-y_i}$$

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\boldsymbol{\beta}} - \ln(\mathcal{L}(\boldsymbol{\beta}))$$

$$- \ln(\mathcal{L}(\boldsymbol{\beta})) = \sum_{i=1}^{n} [\ln\left(1 + e^{\mathbf{x}_i^T \boldsymbol{\beta}}\right) - y_i \mathbf{x}_i^T \boldsymbol{\beta}]$$

Here, $\hat{\boldsymbol{\beta}}$ is a maximum likelihood estimator for $\boldsymbol{\beta}$.

There is no closed form for finding $\hat{\boldsymbol{\beta}}$, thus iterative method is required.

This colab will guide you to finding $\hat{\boldsymbol{\beta}}$ with 2 iterative methods, which are **Gradient Descent** and **Newton's Method**.

From Operations Research Point of View

The logistic regression model can be viewed as a convex program. Solving for $\boldsymbol{\beta}$ by maximum likelihood estimation can be viewed as solving for optimal solution in a convex program.

Let $f(\boldsymbol{\beta}) = -\ln(\mathcal{L}(\boldsymbol{\beta}))$

Decision variables: $\boldsymbol{\beta} \in \mathbb{R}^p$

Convex program: $\min f(\boldsymbol{\beta})$
s.t. $\boldsymbol{\beta} \in \mathbb{R}^p$

It can be proven that the objective function $f(\boldsymbol{\beta})$ is a convex function, which is a requirement for a program to be convex.

Both **Gradient Descent** and **Newton's Method** guarantee an optimal solution for $\boldsymbol{\beta}$ in a convex program.

Please feel free to skip the proof if you are not into mathematics

The proof requires a theorem which states that for a twice differentiable function $f: \mathbb{R}^p \to \mathbb{R}$, $f(\mathbf{x})$ is a convex function if and only if $\nabla^2 f(\mathbf{x})$ is positive semi-definite

From

$$f(\boldsymbol{\beta}) = -\ln(\mathcal{L}(\boldsymbol{\beta})) = \sum_{i=1}^{n} \left[\ln\left(1 + e^{\mathbf{x}_i^T\boldsymbol{\beta}}\right) - y_i \mathbf{x}_i^T \boldsymbol{\beta}\right]$$

$$\nabla f(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(\frac{e^{\mathbf{x}_i^T\boldsymbol{\beta}}}{1 + e^{\mathbf{x}_i^T\boldsymbol{\beta}}} - y_i\right)\mathbf{x}_i$$

$$\nabla^2 f(\boldsymbol{\beta}) = \sum_{i=1}^{n} \frac{e^{\mathbf{x}_i^T\boldsymbol{\beta}}}{1 + e^{\mathbf{x}_i^T\boldsymbol{\beta}}} \frac{1}{1 + e^{\mathbf{x}_i^T\boldsymbol{\beta}}} \mathbf{x}_i \mathbf{x}_i^T$$

Consider $\mathbf{A}_i = \frac{e^{\mathbf{x}_i^T\boldsymbol{\beta}}}{1+e^{\mathbf{x}_i^T\boldsymbol{\beta}}} \frac{1}{1+e^{\mathbf{x}_i^T\boldsymbol{\beta}}} \mathbf{x}_i \mathbf{x}_i^T$

For non-zero vector $\mathbf{u} \in \mathbb{R}^p$,

$$\mathbf{u}^T \mathbf{A}_i \mathbf{u} = \mathbf{u}^T \frac{e^{\mathbf{x}_i^T\boldsymbol{\beta}}}{1 + e^{\mathbf{x}_i^T\boldsymbol{\beta}}} \frac{1}{1 + e^{\mathbf{x}_i^T\boldsymbol{\beta}}} \mathbf{x}_i \mathbf{x}_i^T \mathbf{u}$$

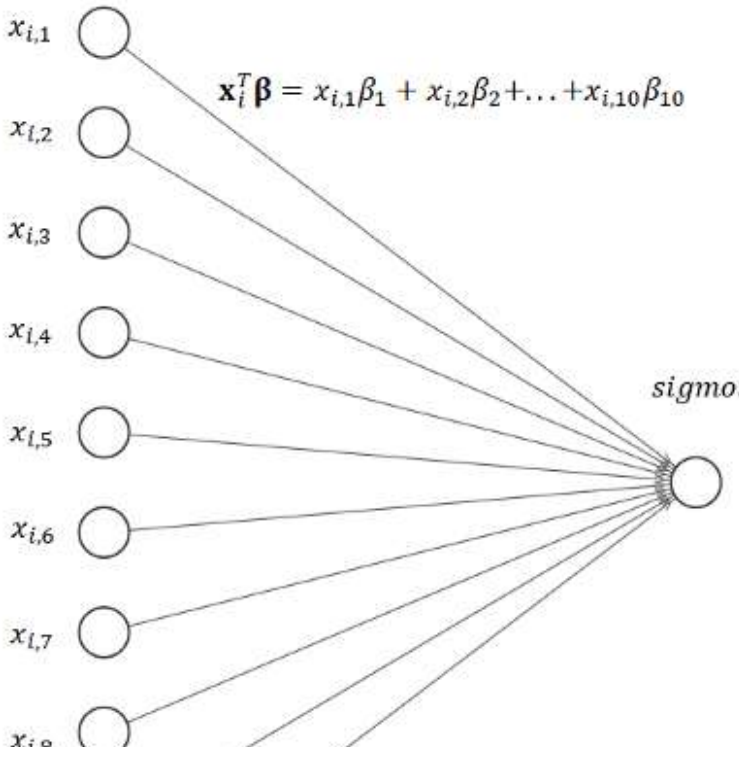
$$= \frac{e^{\mathbf{x}_i^T\boldsymbol{\beta}}}{1 + e^{\mathbf{x}_i^T\boldsymbol{\beta}}} \frac{1}{1 + e^{\mathbf{x}_i^T\boldsymbol{\beta}}} \left(\mathbf{x}_i^T \mathbf{u}\right)^T \left(\mathbf{x}_i^T \mathbf{u}\right) \geq 0$$

Thus, $\mathbf{A}_i$ is positive semi-definite. Moreover, the sum of positive semi-definite matrices is still a positive semi-definite matrix.

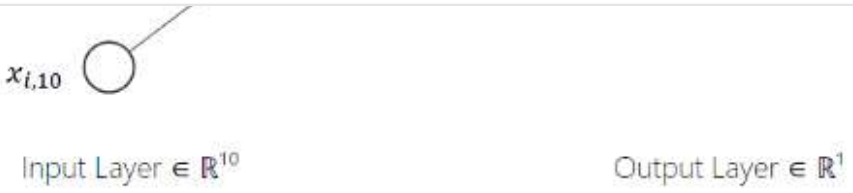
Therefore, $\nabla^2 f(\boldsymbol{\beta})$ is positive semi-definite. As a result, $f(\boldsymbol{\beta})$ is a convex function.

The logistic regression model can also be viewed as a neural network model with 1 input layer and 1 output layer (no hidden layer). Suppose there is explanatory matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ and response vector $\mathbf{Y} \in \mathbb{R}^{n}$. The input layer will consist of p nodes (the number of columns in $\mathbf{X}$) and the output layer will consist of 1 node (the number of columns in $\mathbf{Y}$). The activation function for the node in the output layer is the sigmoid function and the loss function for the model is the negative log-likelihood $f(\boldsymbol{\beta}) = -\ln\big(\mathcal{L}(\boldsymbol{\beta})\big)$ (or binary cross-entropy). Solving for $\boldsymbol{\beta}$ by maximum likelihood estimation can be viewed as training this neural network with input $\mathbf{X}$ and output $\mathbf{Y}$, which produces the coefficients $\boldsymbol{\beta}$.

Suppose you have $p = 10$, the neural network architecture for logistic regression model will be as follows:



$$loss(\boldsymbol{\beta}) = -ln(\mathcal{L}(\boldsymbol{\beta})) = \sum_{i=1}^{n}[ln\left(1 + e^{\mathbf{x}_i^T\boldsymbol{\beta}}\right) - y_i\mathbf{x}_i^T\boldsymbol{\beta}]$$



Neural Network Architecture for Logistic Regression Model with p = 10

Gradient Descent

Gradient descent is an iterative method that solves for minimum of a convex function.

For a convex function $f(\mathbf{x})$, the step taken in each iteration is proportional to the steepest descent direction of $f(\mathbf{x})$, which is $-\frac{f(\mathbf{x})}{\|f(\mathbf{x})\|}$. Please see **Appendix** for further detail on the intuition of gradient descent.

$$\mathbf{x} \leftarrow \mathbf{x} - \lambda \nabla f(\mathbf{x})$$

λ is the step size in each iteration

$\nabla f(\mathbf{x})$ is the gradient of $f(\mathbf{x})$

The method stops when $f(\mathbf{x})$ is close to the minimum, or $\nabla f(\mathbf{x})$ is close to $\mathbf{0}$.

The gradient descent algorithm that solves for $\boldsymbol{\beta}$ in logistic regression will be as follows:

while $\|\nabla f(\boldsymbol{\beta})\| > \epsilon$:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \lambda \nabla f(\boldsymbol{\beta})$$

λ is the parameter you have to specify. It is recommended to take a small step; i.e. $\lambda = 0.001$ is used as an example below. Too small

step size may result in a divergence.

Note that λ is not required to be fixed at every iteration. Instead, λ is allowed to change at every iteration and is chosen via line search. Here, λ is chosen to be fixed just for simplicity.

Newton's Method

Newton's method is an iterative method that solves for minimum of a convex function. Equivalently, the method solves for root of the derivative of the convex function. The idea behind this method is to use a quadratic approximate of the convex function and solve for its minimum at each step. Please see **Appendix** for further detail on the intuition of Newton's method.

For a convex function $f(\mathbf{x})$, the step taken in each iteration is $-\left(\nabla^2 f(\mathbf{x})\right)^{-1}\nabla f(\mathbf{x})$.

$$\mathbf{x} \leftarrow \mathbf{x} - \left(\nabla^2 f(\mathbf{x})\right)^{-1}\nabla f(\mathbf{x})$$

$\nabla f(\mathbf{x})$ is the gradient of $f(\mathbf{x})$

$\nabla^2 f(\mathbf{x})$ is the Hessian matrix of $f(\mathbf{x})$

The method stops when $f(\mathbf{x})$ is close to the minimum, or $\nabla f(\mathbf{x})$ is close to $\mathbf{0}$.

The Newton's method algorithm that solves for $\boldsymbol{\beta}$ in logistic regression will be as follows:

The chosen stopping criterion here is the Euclidean norm of $\nabla f(\boldsymbol{\beta})$ not exceeding a pre-specified tolerance ϵ.

Implementation of Logistic Regression from Scratch

Generate Dataset X and Y


```
import numpy as np

X = np.matrix(np.ones(2484)).T
X = np.append(X,np.matrix([0]*1379+[2]*638+[4]*213+[5]*254).T,axis=1)
Y = np.matrix([1]*24+[0]*1355+[1]*35+[0]*603+[1]*21+[0]*192+[1]*30+
[0]*224).T
```

Define f(β), ∇f(β), and ∇²f(β)

Please see **Appendix** for compact notation of $f(\boldsymbol{\beta})$, $\nabla f(\boldsymbol{\beta})$, and $\nabla^2 f(\boldsymbol{\beta})$

```
def f(beta):
  return np.ravel(np.ones(len(Y))*(np.log(1+np.exp(X*beta)))-
Y.T*X*beta)[0]

def nabla_f(beta):
  return X.T*(1/(1+1/np.exp(X*beta))-Y)

def nabla2_f(beta):
  return X.T*
(np.diag(np.ravel(np.exp(X*beta)/np.power(1+np.exp(X*beta),2)))*X)
```

With initial guess $\boldsymbol{\beta} = [0\ \ 0]^T$, the algorithm converges to $\|f(\boldsymbol{\beta})\| \leq 10^{-10}$ at 771 iterations

$\boldsymbol{\beta} = [-3.8662481\ \ 0.39733662]^T$

```
beta = np.matrix(np.zeros(X.shape[1])).T
TOL = np.power(10.,-10)
lam = 0.001
counter = 0

while np.linalg.norm(nabla_f(beta)) > TOL:
  counter += 1
  beta -= -lam*nabla_f(beta)

print('iter =',counter)
print(beta)
print('norm =',np.linalg.norm(nabla_f(beta)))
```

Implement Newton's Method Algorithm

With initial guess $\boldsymbol{\beta} = [0\ \ 0]^T$, the algorithm converges to $\|f(\boldsymbol{\beta})\| \leq 10^{-10}$ at 7 iterations

$\boldsymbol{\beta} = [-3.8662481\ \ 0.39733662]^T$

```
beta = np.matrix(np.zeros(X.shape[1])).T
TOL = np.power(10.,-10)
counter = 0

while np.linalg.norm(nabla_f(beta)) > TOL:
  counter += 1
  beta -= np.linalg.inv(nabla2_f(beta))*nabla_f(beta)

print('iter =',counter)
print(beta)
print('norm =',np.linalg.norm(nabla_f(beta)))
```

$$\boldsymbol{\beta} = [-3.86624885 \;\; 0.39733451]^T$$

```
from sklearn.linear_model import LogisticRegression

clf =
LogisticRegression(solver='lbfgs',C=1e9,fit_intercept=False).fit(X,np
.ravel(Y))

np.set_printoptions(suppress=True)
print(clf.coef_)
```

Conclusion

In comparing the efficiency of Newton's method and gradient descent for solving the coefficients $\boldsymbol{\beta} \in \mathbb{R}^p$ in logistic regression model given explanatory matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ and response vector $\mathbf{Y} \in \mathbb{R}^n$,the runtime complexity for each iteration in Newton's method is $\mathcal{O}(np(n+p)+p^3)$ while the runtime complexity for each iteration in gradient descent is $\mathcal{O}(np)$. The memory complexity for each iteration in Newton's method is $\mathcal{O}(p^2)$ while the memory complexity for each iteration in gradient descent is $\mathcal{O}(p)$. It can be seen that Newton's method requires more number of operations and more amount of memory than gradient descent at each iteration. However, Newton's method requires less iterations to converge to minimum because it uses the knowledge of the second derivative $\nabla^2 f(\boldsymbol{\beta})$ at each iteration while gradient descent only uses the knowledge of the first derivative $\nabla f(\boldsymbol{\beta})$ at each iteration. This is the trade-off between the complexity and speed of convergence of Newton's method and gradient descent. In fact,

matrix $\left(\nabla^2 f(\boldsymbol{\beta})\right)^{-1}$. Computing the inverse of Hessian matrix can be unavailable for a very large dataset. In real world, ones ought to approximate the inverse of Hessian matrix, leading to less complexity, in exchange for more iterations to converge to minimum. The methods are called the Quasi-Newton methods. It is noteworthy that the solver specified in `sklearn.linear_model.LogisticRegression` is `solver='lbfgs'` or Limited-memory BFGS, an iterative method in the family of Quasi-Newton methods.

Appendix

Intuition of Gradient Descent

Gradient descent is an iterative method that solves for minimum of a convex function. The method moves the guessing point to the next guessing point in the direction that $f(\mathbf{x})$ decreases the fastest.

To move in the direction that $f(\mathbf{x})$ decreases the fastest, consider the theorem

> For $f(\mathbf{x}) \in \mathbb{R}$, a unit vector $-\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}$ is the steepest descent direction of $f(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_0$

In gradient descent, in order to find the minimum of the convex function $f(\mathbf{x})$, we start from our initial guess $\mathbf{x}_0$, move $\mathbf{x}_0$ to the next point in the direction that $f(\mathbf{x})$ decreases the fastest $\mathbf{x}_0 - \lambda_0 \nabla f(\mathbf{x}_0)$, let $\mathbf{x}_1 = \mathbf{x}_0 - \lambda_0 \nabla f(\mathbf{x}_0)$ be the next guess, move $\mathbf{x}_1$ to the next point in the direction that $f(\mathbf{x})$ decreases the fastest $\mathbf{x}_1 - \lambda_1 \nabla f(\mathbf{x}_1)$, let $\mathbf{x}_2 = \mathbf{x}_1 - \lambda_1 \nabla f(\mathbf{x}_1)$ be the next guess, and do these steps recursively until $f(\mathbf{x}_i)$ is close to minimum. Therefore

follows:

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i - \lambda_i \nabla f(\mathbf{x}_i)$$

for arbitrary value of λ_i

At each iteration, λ_i is chosen such that $f(\mathbf{x}_i - \lambda_i \nabla f(\mathbf{x}_i))$ is smallest. This can be done via line search.

You can see that gradient descent moves from point $\mathbf{x}_i$ to point $\mathbf{x}_{i+1}$ in the direction $-\lambda_i \nabla f(\mathbf{x}_i)$, which is proportional to $-\frac{\nabla f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)\|}$, the steepest descent direction of $f(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_i$.

Intuition of Newton's Method

Compact Notation of f(β), ∇f(β), and ∇²f(β)

Machine Learning Optimization Logistic Regression Data Science Convex Optimization