The impact of replacing the number e in Logistic Regression.

Aidyn Kadyr
MATH 540 Statistical Learning
Nazarbayev University

Abstract

This project aims at considering alternative constants to e in the sigmoid function applied in the logistic regression and analyse the model behaviour. Changing the constant resulted in feature scaling and the traditional choice of e is justified by its convenience and simplicity. The source code is available at the Github.

Introduction

Logistic Regression uses a sigmoid function given as $\sigma(x) = \frac{1}{1+e^{-x}}$ to squish the linear combination of independent variables into the 0 to 1 probability interval. However, what would happen if instead of e, other constants would be chosen? What would be the differences and how would the accuracy change?
This project sets alternative values for e in the sigmoid function, the loss function is redefined from the Maximum Likelihood Estimator perspective and Stochastic Gradient Descent is used for optimization.
Training and testing is done on the Breast Cancer dataset.

Loss function

The loss function for the binary Logistic Regression is given by:

$$L(w) = -\sum y_i \cdot ln(p_i) + (1 - y_i) \cdot ln(1 - p_i)$$

where

$$p_i = \sigma(w^T x_i) = \frac{1}{1 + e^{-w^T x_i}}$$

and the gradient is defined by:

$$\nabla_w L(w) = -\sum y_i \cdot \nabla_w ln(p_i) + (1 - y_i) \cdot \nabla_w ln(p_i)$$

$$-\sum \frac{y_i}{p_i} \nabla_w p_i - \frac{1 - y_i}{1 - p_i} \nabla_w p_i$$

$$-\sum (\frac{y_i}{p_i} - \frac{1 - y_i}{1 - p_i}) \cdot \nabla_w p_i$$

where

$$\nabla_w p_i = \nabla_w \sigma(w^T x) = p_i(1 - p_i)x_i$$

After simplification:

$$\nabla_w L(w) = -\sum (y_i - p_i)x_i$$

Now, let's set e to some constant C. Then:

$$\nabla_w p_i = \nabla_w \frac{1}{1 + C^{-w^T x_i}} = \nabla_w (1 + C^{-w^T x_i})^{-1}$$

$$= -(1 + C^{-w^T x_i})^{-2} \cdot ln(C) \cdot C^{-w^T x_i} \cdot (-x_i)$$

$$= \frac{ln(C) \cdot C^{-w^T x_i} \cdot x_i}{(1 + C^{-w^T x_i})^2}$$

$$= \frac{C^{-w^T x_i}}{1 + C^{-w^T x_i}} \cdot \frac{1}{1 + C^{-w^T x_i}} \cdot ln(C) \cdot x_i$$

$$= (1 - p_i) \cdot p_i \cdot ln(C) \cdot x_i$$

Substituting into $\nabla_w L(w)$:

$$\nabla_w L(w) = -\sum ln(C) \cdot (y_i - p_i) \cdot x_i$$

and the sigmoid function:

$$p_i = \sigma(w^T x_i) = \frac{1}{1 + C^{-w^T x_i}}$$

Baseline result

The baseline result is done using $SGDClassifer$ from the sklearn linear model library in Jupyter environment. This model was chosen over $LogisticRegression$ from the same library, since the latter does not support SGD optimization. Using $SGDClassifier$ the loss was set to log to match the logistic regression loss function, $alpha$ of 0 to run without the regularization term and constant $learning\ rate$ of 0.001. The accuracy score for the test set was 0.9649:

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(loss = 'log', max_iter = 2000, \
learning_rate = 'constant', eta0 = 0.001, alpha = 0)\
.fit(X_train, y_train)
sgd.score(X_test,y_test)
```

```
0.9649122807017544
```

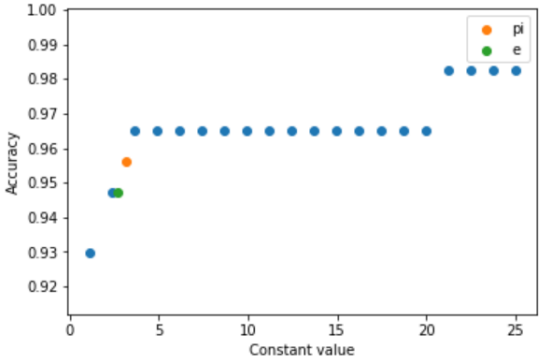
Python implementation

The no library approach was taken by defining the $gradienterror$ function which outputs

$$\nabla_w L(w) = -ln(C) \cdot (y_i - p_i) \cdot x_i$$

for a specific point for the Stochastic Gradient Descent approach. The function takes $w, n, base$ as parameters, where w is the weight vector, n is the n^{th} datapoint and $base$ is for the constant: `def gradient_error(w, n, base):`

Number of iterations was set to 2000, learning rate to 0.001, and initial weights to a zero vector. Since the domain of logarithm is non-zero, constants were set to a list of 20 values between 1.1 and 25 using $numpy.linspace$ function. Note, if the constant is set to 1, then sigmoid itself will equal a constant $\frac{1}{2}$. The scatter plot of the Accuracy vs. the Constant Value:



Analysis

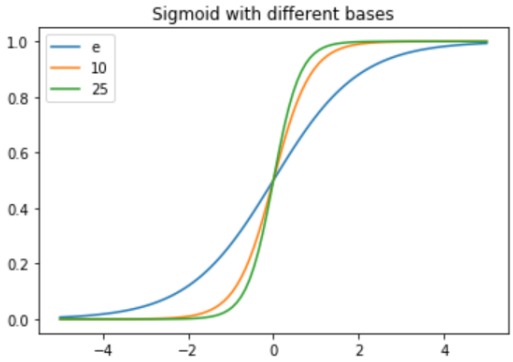
The scatter plot above shows an increasing trend for the accuracy with an increase in the constant value. The traditional choice of e showed an accuracy of 0.94736, which slightly deviates from the baseline result due to the stochastic nature of the algorithm. Accuracy for 10 is 0.9649 and for 25 is 0.9824.

Effect on the gradient of the error

Since the gadient of the error function with a general constant C is a scaled version of the traditional one by the $log(C)$, it gets multiplied by the learning rate in the update rule. Therefore, the initial learning rate of 0.001 gets scaled by the $log(C)$.

Effect on the sigmoid function

1 An increase in the base value decreases the numerator resulting in higher value for the sigmoid function:



2 Following the effect on the gradient of error, weights different from the traditional approach are obtained, which returns different scores for the sigmoid function.

$$p_i = \sigma(w^T x_i) = \frac{1}{1 + C^{-w^T x_i}}$$

where

$$C^{-w^T x_i} = e^{-w^T x_i ln(C)}$$

can be expressed as a traditional logistic regression with features scaled by a factor of $ln(C)$

Conclusion

Setting other constants instead of e in the sigmoid function for logistic regression resulted in feature scaling and change in learning rate for the SGD (scaled gradient.)

The increase in accuracy of the model is probably accounted for the feature scaling, since it speeds up the SGD convergence if the number of iterations is held constant.

Let's show that accuracy of 0.94736 with the traditional approach can be obtained by using base 10 instead of e. Since using base C is the same as scaling features by $ln(C)$, multiplying features by a factor of $\frac{1}{ln(C)}$ produced the same accuracy of 0.94736 for base C.

This way, changing the e constant in the Logistic Regression has the same effect as doing a feature scaling. It may or may not increase the model performance. In the case of the Breast Cancer dataset, the model accuracy was increased by taking larger constants.