Final Project: Logistic Regression with IRLS vs Neural Network with Gradient Descent

Daniel Smith 205212977 STATS205

University of California, Los Angeles

Abstract

The purpose of this analysis is to compare and contrast the models used to predict a binary class. This is done by having both models predict the probability of binary class using several features from a given data set. Neither one of these models is inherently better than the other, however, when working with real world data, it is important to consider constraints such as memory used, time to train, and interpretability of the model, as well as the accuracy of the results. This study will go over two common models and methods of training the models: Logistic Regression with IRLS and Neural Networks with Gradient Descent

Contents

amsmath

1 Introduction

In order to compare the two models, we first must find an appropriate data set and define our task. For this analysis, I used this kaggle data set. This data set is a comprehensive description of individuals and their health data with, most importantly, a label of their obesity level as categorical variables. This class is then compressed into a binary class where anything overweight and above counts as obese (1), while anything normal or below is considered healthy (0).

1.1 Data Exploration

The data set comprises of 2111 observations with 17 features and 0 N/A or Null values. Of these values 8 of them are categorical features, while 9 of them are numerical features. One feature is dropped (NObeyesdad) which is a categorical variable comprising of obesity levels of the observation. The function created to encode the label is given by formula 1

$$Encode(L) = \begin{cases} 0, & \text{if } L \in \{\text{Insufficient Weight}\} \\ 1, & \text{else} \end{cases} \qquad \text{(formula 1)}$$

The remaining numerical values are scaled using StandardScaler and the categorical features are one-hot encoded. This results in a data set with 2111 observations and 24 features.

2 Logistic Regression with IRLS

Exploring the IRLS algorithm

2.1 Logistic Regression

Logistic Regression operates in a similar principle to Linear Regression. We have a model that multiplies each feature by a weight and uses that sum passed through a function bounded by (0,1) in order to assign it a probability.

2.2 Iteratively Reweighted Least Squares

IRLS or Iteratively Reweighted Least Squares is an algorithm that uses the Newton-Raphson method for updating the model parameters until the prior and current parameters have an acceptable difference below our level of tolerance. Below is the algorithm for IRLS

1. **Initialize** $\beta^{(0)}$, often with zeros or small random values
2. Compute **probabilities** $\mathbf{p} = \sigma(\mathbf{X}\beta^{(t)})$. This value is the predicted probability of our model with the current weights passed through a sigmoid function that bounds the value between (0,1)
3. Construct the **weight matrix W** with $W_{ii} = p_i(1 - p_i)$. This matrix is a diagonal matrix of the probabilities
4. Compute **z** as the **adjusted response**, defined as:

$$\mathbf{z} = \mathbf{X}\beta^{(t)} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$$

5. Solve the Newton Raphson Update Equation:

$$\beta^{(t+1)} = (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{z}$$

6. Repeat until **convergence** (e.g., when $||\beta^{(t+1)} - \beta^{(t)}||$ is below a given level of tolerance).

The resulting Beta is then used to make the final probability prediction

3 Application Interface

Using Neural Networks and Gradient Descent

3.1 Neural Network Architecture

The model being created is a Fully Connected Network with a 1 dimensional output layer that predicts the probability that an observation is overweight. Below is a diagram of the Neural Network (figure 1), as well as the PyTorch code (figure 2) that signifies how the model is connected.

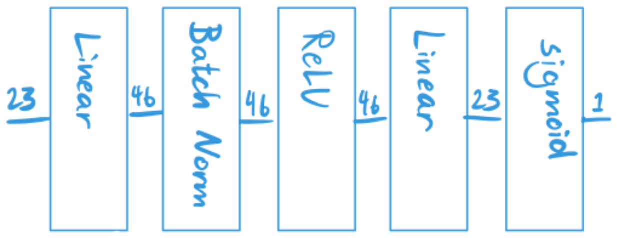
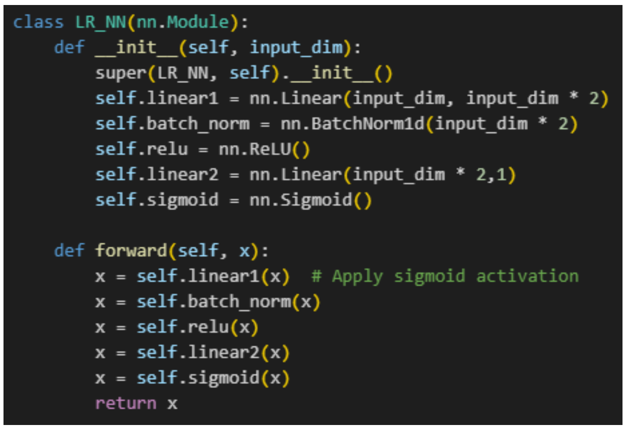


Figure 1: Neural Network Diagram

Figure 1 shows the corresponding dimensions that go into each layer and the corresponding dimensions that come out.



```
class LR_NN(nn.Module):
    def __init__(self, input_dim):
        super(LR_NN, self).__init__()
        self.linear1 = nn.Linear(input_dim, input_dim * 2)
        self.batch_norm = nn.BatchNorm1d(input_dim * 2)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(input_dim * 2,1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.linear1(x)  # Apply sigmoid activation
        x = self.batch_norm(x)
        x = self.relu(x)
        x = self.linear2(x)
        x = self.sigmoid(x)
        return x
```

Figure 2: Pytorch Code

3.2 Forward Pass

Each of the layers in the neural network corresponds to a mathematical operation. The first layer Linear is just a matrix multiplication where the input vector is matrix multiplied to get a resulting vector with a separate amount of dimensions, in this case we are turning 1 by 23 dimensional vector to 1 by 46 dimensional vector. The next layer, Batch Norm works twofold. The first method is by reducing Internal Covariate shift, which is done after the Linear Layer. The second is by smoothing the Loss Function used in Gradient Descent. The ReLU Layer (formula 2) is the method used to prevent the Neural Network from becoming Linear as Neural Networks are excellent at approximating non linear functions.

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \qquad \text{(formula 2)}$$

$$\hat{y} = \sigma(f(x)) = \frac{1}{1 + e^{-f(x)}} \tag{1}$$

$$J(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right] \tag{2}$$

By passing through each layer we end at our final result for each observation, which is probability score between 0 and 1 defined in equation 1, which than is scored against the real values in our data (equation 2). This is known as the Loss Function

3.3 Backward Propagation and Gradient Descent

As we can tell computing the forward pass with weights that are randomly initialized, will not result in an accurate score, similar to IRLS we need a method to update the parameters. This method is known as gradient descent with an update rule (equation 3).

$$\theta := \theta - \alpha \nabla_\theta J(\theta) \tag{3}$$

where alpha is the learning rate hyperparameter, theta are our parameters, and J is the Loss Function.

The next step is to fund the derivative of the Loss Function. As the Loss Function is defined it is our final result of the Forward pass. This results in series of functions chained together to produce 1 output. The method of finding the derivatives of nested functions is known as the chain rule (equation 4).

$$\nabla_\theta J(\theta) = \frac{\partial J}{\partial \mathbf{a}^{(J)}} \cdot \frac{\partial \mathbf{a}^{(J)}}{\partial \mathbf{z}^{(J)}} \cdot \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{a}^{(L-1)}} \cdots \tag{4}$$

.

Combining all these steps we

1. Calculate the value of $J(\theta)$.

2. Calculate the derivative value $\nabla_\theta J(\theta)$.

3. Use gradient descent and the update rule to update each parameter.

4 Comparisons

IRLS vs Gradient Descent

4.1 Results

Comparing both models we can use K-Fold Cross Validation to split our data into 5 different components and use 4 of them to train the model while the other 1 is used for testing our accuracy. Each component will then rotate so that each fold will be used to calculate accuracy. Doing for both models we get the results shown below (figure 3).

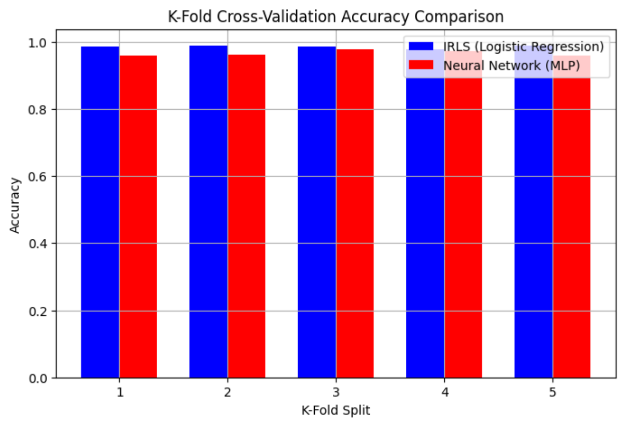


Figure 3: Results of Accuracy

comparing the accuracy results, all folds have an accuracy above 97 percent. We can see that for all folds IRLS is the more accurate method for updating the parameters, however the difference in accuracy is less than 1 percent.

The next characteristic we can measure is the time it takes to train the each model. The models will each see the data the same number of times. This is measured using epochs, which is defined as one full pass through the data set. Each model and fold has been trained for 50 epochs

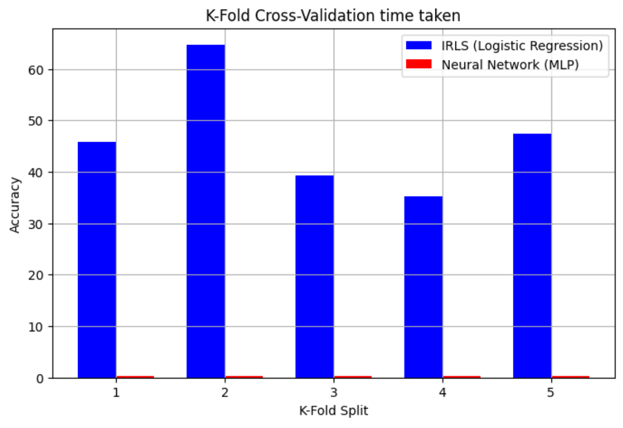


Figure 4: Results of Time taken to train

Looking at figure 4 we can see there is a huge difference between the two models in terms of time it takes to train. The IRLS Logistic Regression model has 24 parameters, while the Neural Network with Gradient Descent has 1345 parameters. This means in terms of space and memory used, Logistic Regression with IRLS is exponentially more efficient than a Neural Network, however we can see that in terms of Time, Neural Networks beat Logistic Regression with IRLS. This is due to several factors, the most important being that IRLS has a second order derivative and involves finding the inverse of a matrix. Taking the Inverse of a matrix is often slow and inefficient and cannot be computed in Parallel. With Gradient Descent in Neural Networks however, the gradients of each batch in Gradient Descent can be trained in Parallel which, although requires more calculations ends up training faster

4.2 Analysis

With both models trained, one feature we can do with Logistic Regression with IRLS is find the coefficient of the feature that has the highest affect on our model's results. This feature is calculated to be FCVC (Frequent calories volume consumption), which is a binary categorical variable that determines whether the observation eats high volume calories often. Anecdotally this makes sense as the biggest advice that athletes are given to manage their weight is to be precice in the food they consume.

5 Future Work and Conclusion

5.1 Future Work

While I am satisfied with the results of both models, there are some other methods that can be used to optimize the training further. Some involve fine tuning hyperparameters using more K-Fold Cross Validation, while others involve some form of regularization to prevent the model's from overfitting. This is primarily a concern with Neural Networks as what has been shown is that the Neural Netowrk has more parameters and thus is prone to overfitting more. While there are many methods of regularization such dropout, the method that can be used is adding a L1/L2 penalty (equation 5)

$$\text{L2} = \lambda \sum_i |w_i^2| \tag{5}$$

This value can be appended to the loss function J and subsequently will then be incorporated into the weights via Gradient Descent and updates. This penalty is particularly useful since it results in a final model with weights that have no value. As such the final model with a penalty has fewer weights than the vanilla model, and thus prevents overfitting

5.2 Conclusion

Overall this project aimed to analyze the difference between two different methods of updating weights and model structures. To overview, Logistic Regression with IRLS will result in a more interpretable model that occupies less memory and has a slight, but consistent edge in accuracy. While the main benefit of a Neural Network with Gradient Descent is that the model can be trained vastly quicker.

Depending on the situation either models and methods can be used, however, time constraints, I would use the Logistic Regression with IRLS as the model in this case will tell us which feature has the largest effect on our desired prediction in addition to being more accurate than the Neural Network.

Thank you so much Professor Li for teaching this course, I was concerned as a Computer Science Graduate I would not be able to keep pace with the course, however I feel I learned a lot in this class during my final quarter at UCLA.

All code can be found at GitHub Link

References