Machine Learning | Final Report

Abstract:

For this project, the model that was used is Regularized Logistic Regression with Cross-Validation. The Logistic Regression model was optimized using the Iterative Re-weighted Least Squares (IRLS) algorithm. More generally the Logistic Regression model was chosen because the goal of the dataset sourced for this project was to predict a patient's mortality given information such as clinical, health, and lifestyle information. Since the target of the dataset, a feature called death event, was given as binary values the machine learning problem we will be tackling is a binary classification problem. Therefore, Logistic Regression, a binary classification algorithm, was chosen as the primary algorithm that would be implemented. Regularization and cross-validation were added to prevent the Logistic Regression model from overfitting during training. Overfitting will cause the model to have a high training performance, but then perform poorly on data it has not seen previously during training.

Once the algorithm was implemented, the average testing error rate for All-Features after 100 iterations is ~21 – 19% and for Two-Features it was ~18 – 16%. The average error rates are presented because before every run of the algorithm the entire dataset is randomly shuffled and then split into a 2/3 training and 1/3 testing set. With the training set being shuffled and subdivided further in a training and validation set. Due to this random shuffling and splitting of the dataset there is variability in the resulting error rate calculated by the model. To account for this variability the algorithm was ran for 100 iterations and the final error rate was then calculated by averaging the error rate for all the runs.

Theoretical Description:

Logistic Regression as implied by the algorithm's name fits an "S" shaped logistic sigmoid curve to the dataset [1]. However, despite having the word *Regression* within its name, this algorithm is not used to predict a continuous value instead it is used for classification. The logistic sigmoid function (eqn. 1) in this case is also known as the squashing or activation function in this instance because it squashes all the data points between 0 and 1 [2]. Due to this squashing performed by the logistic sigmoid the density estimations of the model, theta parameters, can no longer be calculated in closed form [3].

$$y = \frac{1}{1+e^{-\theta^T \phi(x)}} \qquad \text{[eqn. 1]}$$

Therefore, the iterative optimization approach is used when estimating the optimal theta parameters. The iterative approach used for determining theta parameters was Newton-Raphson's cross-entropy optimization, which was implemented within the IRLS algorithm. The IRLS algorithm is used to update and compute the Maximum Likelihood Estimate (MLE) theta values [2].

Once the MLE theta is determined, these parameters along with the features $\phi(x)$ of the dataset can be passed into the activation function for a classification prediction. The value returned by the activation function is the probability that this specific data sample is predicted to belong to class 1. If a value above 0.5 is returned we know that the predicted class is 1, else the predicted class is 0.

Although we have covered the theoretical basics of training a Logistic Regression model, it is important that while training the model we prevent it from overfitting. If the model overfits it is difficult for the model to generalize to unseen data and would often fail to make accurate predictions. To avoid overfitting of the model a regularization parameter (λ) can be introduced to the cost or objective function which can control the complexity of the polynomial or penalizing the large theta parameters. Before the regularization parameter was introduced the negative log-likelihood or cross-entropy objective function (eqn. 2) is the equation used to determine model error [4]. After squaring the initial loss function and adding the regularization parameter we then obtain a regularized objective function (eqn. 3) which penalizes large theta values [5].

$$E(\theta) = -\sum_{n=1}^{N} t^{(n)} ln\sigma\left(\theta^T \phi^{(n)}\right) + (1 - t^{(n)}) \ln\left[1 - \sigma(\theta^T \phi^{(n)})\right] \quad \text{[eqn. 2]}$$

$$E(\theta) = -\frac{1}{2}\sum_{n=1}^{N}\left[t^{(n)} ln\sigma\left(\theta^T \phi^{(n)}\right) + \left(1 - t^{(n)}\right) \ln\left[1 - \sigma\left(\theta^T \phi^{(n)}\right)\right]\right]^2 + \frac{\lambda}{2}||\theta||^2 \quad \text{[eqn. 3]}$$

Subsequently, with the change of the loss or objective function to train a Regularized Logistic Regression model the regularization parameter also needs to be accounted for when calculating the gradient (eqn. 4) and hessian (eqn. 5) of the objective function [4]. As they are used to determine the step size of the theta parameters.

$$\nabla_\theta E = \sum_{n=1}^{N}[\sigma\left(\theta^T \phi^{(n)}\right) - t^{(n)}]\phi^{(n)} = A^T(y - t) \rightarrow A^T(y - t) + \lambda\theta \quad \text{[eqn. 4]}$$

$$H = \nabla_\theta^2 E = A^T R A \rightarrow A^T R A + \lambda \quad \text{[eqn. 5]}$$

The method that was used to determine the optimal regularization parameter within this project is cross-validation. To implement cross-validation the whole entire dataset is shuffled and divided into 3 different sets, each used for a specific purpose [6]. The first split of the dataset results in a 2/3 training and 1/3 testing set. The 2/3 training set is then taken and further subdivided into in 2/3 training and 1/3 validation. The final 2/3 training set is then used for training the Logistic Regression model and computing the MLE theta parameters within the IRLS algorithm.

Once the MLE parameters are found the theta values are returned within the cross-validation function and used to compute the prediction error on the validation set. The previous and current validation set errors are compared with each other if the current validation set error is smaller the loop continues. Essentially, the goal of the cross-validation function is to determine a value for the regularized parameter (λ) that when utilized to train the model would produce theta values that generalizes well to datasets that the model has not seen previously.

Combining all these components together, implementing a Regularized Logistic Regression model with Cross-Validation would result in a binary classifier that has been restrained during training to not overfit on the training dataset. As a result of not overfitting the model is able to generalize to data samples that is has not seen before and therefore make more accurate predictions.

Implementation Details:

As with any machine learning implementation the first thing to do is to gather data. The first the code does in the main() function is call the read_data() function (see *line 201*). The read_data() function (see *line 8-65*) essentially reads the dataset from the csv file provided, ensure the data imported is of the right type, and creates the feature and target matrices. These matrices are then returned to the main function. At *line 208*, the code enters a 100 iteration for loop to run the Regularized Logistic Regression with Cross-validation algorithm for 100 times. With each iteration and run of the machine learning algorithm the entire dataset is randomly shuffled and split into 2/3 training and 1/3 testing, with the 2/3 training being split further into 2/3 training and 1/3 validation.

Once the dataset has been split the identity basis is applied to the feature matrix of each set (see *line 75-79*) producing the $\phi(x)$ matrix. The cross_validation function is then called to train the logistic regression model and determines the optimal regularization parameter value and MLE theta values for the dataset provided. Stepping into the cross_validation function (see *lines 83-106*) the initial regularization parameter (λ) and iteration count (k) are set to 0. After the initialization these variables the IRLS function is called and given the training set, training set labels, and the initial lambda value. The IRLS function is used to train the Logistic Regression model and return the MLE theta parameters.

Stepping into the IRLS function (see *line 163-181*) before training some hyperparameters are initialized. Theta parameters are set to zero, counter set to 0, and the arbitrarily small number ε is set to 0.001. Before entering the while loop the R matrix is calculated from the initial theta values and train features following what's being in eqn. 6 (see *line 154-159*), the R matrix is calculated because within the while loop it is needed to calculate the hessian later to determine the step size of the gradient descent. Later within the loop it will also be updated with every new theta vector generated by each iteration of the loop until the loop terminates.

$$R_{(n,n)} = y^{(n)}(1 - y^{(n)}) \qquad \text{[eqn. 6]}$$

Within the while loop gradient descent is being performed to find the MLE theta parameters. The hessian and gradient of the regularized objective function are calculated in *lines 171 and 172*, by calling the hessian (see *line 134-136*) and grad_obj (see *lines 127-130*) functions respectively. Within the code the hessian is being calculated with the regularized hessian equation as specified above by eqn. 5 and the gradient is being calculated using the regularized gradient equation specified in eqn. 4.

Once the hessian and gradient matrices are calculated the new theta value is determined by using eqn. 7, where the inverse of the Hessian matrix is matrix multiplied with the gradient matrix to determine the step size (gradient direction) the theta values should move in. Note a try and except block is placed here because when the R matrix is approximately zero the matrix becomes ill-conditioned causing the algorithm to halt [4]. In order to prevent the algorithm from halting if an np.linalg.inv() singular matrix error is throw, the algorithm instead uses the pseudo-inverse of the matrix to compute the step size and update theta.

$$\theta^{(k+1)} = \theta^{(k)} - H^{-1}\nabla_{\theta}E \qquad \text{[eqn. 7]}$$

With each iteration the gradient of the theta values is calculated, once the norm of the gradient becomes significantly small or if the max iteration of 700 is reached the loop will terminate and return the current theta parameters. Using the current theta values the loss for the training set is calculated using the obj_fun function (see *lines 140-150*), which the error is calculated using eqn. 3 the regularized objective function.

Returning from IRLS back in the cross_validation function the newly calculated theta values are then used to calculate the current validation error using the same obj_fun function as referenced in the IRLS for the training set. Except this time, it is used to calculate the error for a dataset that the model has not seen before. This helps check the theta values obtained through training and determine if it has overfitted to the training dataset or not.

Once inside the cross_validation while the lambda is incremented with every iteration, the model is trained using that lambda value and theta parameters are produced. Once the theta parameters return a new validation error is calculated. If the theta values returned were determined to have overfitted or the max iteration of the cross_validation loop has been reached the previous theta, lambda, and error value of the training set will be returned as the optimal parameters found.

The optimal parameters returned by the cross_validation function are the values used to determine the final training, validation, and testing set error rates. However, this time instead of using the objective function to calculate loss or error the logistic sigmoid activation function along with the thetas found and the $\phi(x)$ feature matrix are used to make predictions about the class of the sample given its features (see *line 185-196*). The predictions are between [0, 1], for any prediction that is > 0.5 the predicted class is 1 else the predicted class is 0. These predictions are then check against the actual labels of the data sample. If the prediction is wrong error increases by 1. At aggregating how many samples were assigned the wrong label, the error is then divided by the total number of samples within the set to obtain an error rate.

The same Regularized Logistic Regression model and Cross Validation algorithm is followed by both feature matrices that were tested for this project. The same random shuffling and dataset split were also performed on both feature matrices to make the comparison as objective as possible.

After the 100 iterations of the algorithm the training, validation, and testing error rates for both feature matrices were average to obtain a final averaged error rates because the random shuffling and splitting causes ambiguity and variability in the algorithm performance itself. Generally, taking the average helps with visualizing generally where the error rates lie considering variability.

Reproducibility:

The results can be reproduced by installing the following python libraries: csv, numpy, and matplotlib. Before running the code ensure that the *name* field of the contains the correct name of the file within the same directory as the code or contains the full path of where the ‘heart_failure_clinical_records_dataset.csv’ file is located on the local machine. After those

items have been addressed just running the code should be sufficient in reproducing the results that was mentioned in this report.

If any parameters happen to change within the code file my initial parameters are as follow: *iterations* = 100, max iteration for the IRSL function loop = 700, and max iteration for the cross_validation function loop = 500, cross_validation function's *delta_lambda* = 0.001, and split_data function's *n* = (number of samples * 2/3). Although the random shuffling might prevent the exact same average error rate from returning every time, my executions of the code so far with averaging the error rate of 100 iterations has produced consistent and similar error rate results from one run to the next.

Results:

The average final test error rate over 100 iterations of the algorithm always returns to be ~21 – 19% for All-Features and ~18 – 16% for Two-Features. The All-Features matrix compared to the Two-Features matrix is more time efficient because unlike the Two-Features matrix which rarely converge after passing through 500 iterations of the cross-validation loop or 700 iterations of the IRLS loop. The All-Features matrix usually coverages around 100 – 250 iterations of either loops. Making the All-Features matrix more time efficient to train, while the Two-Features matrix takes twice or sometimes 3x as long to complete training. Although the All-Features matrix does contain 3 – 4 times more features than the Two-Features matrix, memory usage for both matrices are similar. Since one although smaller takes more iterations to train, and the other although larger converges in training relatively quickly.

Within Davide Chicco and Giuseppe Jurman's research paper the authors claimed that using serum creatinine and ejection fraction alone was sufficient in predicting a patient's survival with heart failure [7]. However, after implementing the Regularized Logistic Regression with Cross-Validation just giving the machine learning algorithm serum creatinine and ejection fraction did not result in a more accurate prediction model. Instead, given just these two features the model was performing approximately 3% worse on the testing set predictions, with an average error rate of ~23%. The model that utilized all the features was performing better than the model that only had serum creatinine and ejection fraction.

However, it was not until I read the paper more closely did it state that for a model with just serum creatinine and ejection fraction features to perform well the feature follow-up month also needs to be included [7]. After this was determined, changes were made to the code so that in the Two-Features matrix aside from just serum creatinine and ejection fraction, follow-up month was also included. Surprising after the addition of the feature called follow-up month the Two-Features matrix although is still taking quite a long time to train and converge is indeed outperforming the All-Features matrix by roughly 3% given that with each run the same set of data samples are randomly shuffled and passed to each training algorithm respectively.

With this find the title of the article this dataset was sourced from is misleading. If the feature follow-up month needs to be included as a feature as well for the machine learning model to achieve the desired result, then why was it not mentioned in the title like the other two features? Nevertheless, when the algorithm is feed the correct features, the performance is

consistent even after 100 different iterations or runs of the same algorithm with a randomly shuffled and split dataset.

Comparing the average training and validation rates that have been returned by the Regularized Logistic Regression with Cross-Validation algorithm, it can be observed that generally the training error rate for the Two-Features matrix is slightly higher than the All-Features matrix model. Although the training error rate is slightly higher it can be observed that the validation and testing error rates are significantly better than the All-Features model. Which can mean that the All-Features model might have too many features, causing the model to overfit a little more on the training dataset. This slight overfitting translates into a significant decrease in validation and testing set prediction accuracy.

This correlation that can be observed comes to me as a surprise because I did not think that overfit by just slightly with a lot more parameters than you truly need can cause such a degradation in the accuracy of your model. Ultimately, this shows that it is important to be able to provide just the right number of features and overloading your model with features that do not assist in its prediction might cause your model more harm in terms of decreasing prediction accuracy.

References:

[1] *Logistic Regression in Machine Learning*, JavaTpoint, Accessed on: Dec. 1, 2021. [Online]. Available: https://www.javatpoint.com/logistic-regression-in-machine-learning

[2] S. S. Azam, *Classification and Logistic Regression*, Machine Learning Medium, Aug. 31, Accessed on: Nov. 27, 2021. [Online]. Available: https://machinelearningmedium.com/2017/08/31/classification-and-representation/

[3] C. L. Wyatt, Class Lecture, Topic: "Meeting #42: Course Review." ECE4424/CS4824, The Bradly Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA., Dec. 8, 2021.

[4] C. L. Wyatt, Class Lecture, Topic: "Meeting #18: Perceptron." ECE4424/CS4824, The Bradly Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA., Oct. 6, 2021.

[5] S. S. Azam, *Regularized Logistic Regression,* Machine Learning Medium, Sep. 15, Accessed on: Nov. 27, 2021. [Online]. Available: https://machinelearningmedium.com/2017/09/15/regularized-logistic-regression/

[6] C. L. Wyatt, Class Lecture, Topic: "Meeting #20: Linear Regression." ECE4424/CS4824, The Bradly Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA., Oct. 11, 2021.

[7] D. Chicco, G. Jurman, *Machine Learning can Predict Survivial of Patients with Heart Failure from Serum Creatinine and Ejection Fraction Alone*, BioMed Central Ltd, Accessed on: Nov. 12, 2021. [Online]. Available: https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-020-1023-5

Data Link:
https://drive.google.com/file/d/1N4hBeGuDhWWvFNK1W2gAaWdFTXRUVZaA/view?usp=sharing

Appendix:

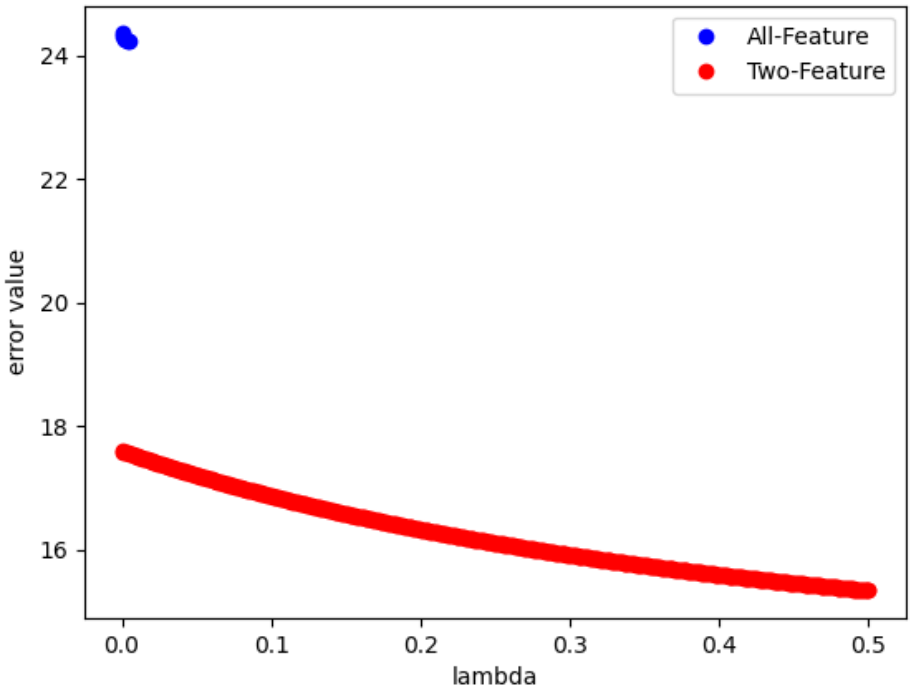

Figure 1. Last Iteration Training Error Value vs. Lambda (Run 1)

```
All-Feature Average Errors
Training Error:  0.1485606060606061
Validation Error:  0.19820895522388057
Testing Error:  0.19609999999999997
Two-Feature Average Errors
Training Error:  0.15530303030303033
Validation Error:  0.17208955223880593
Testing Error:  0.1698
```

Figure 2. Error Rates (Run 1)

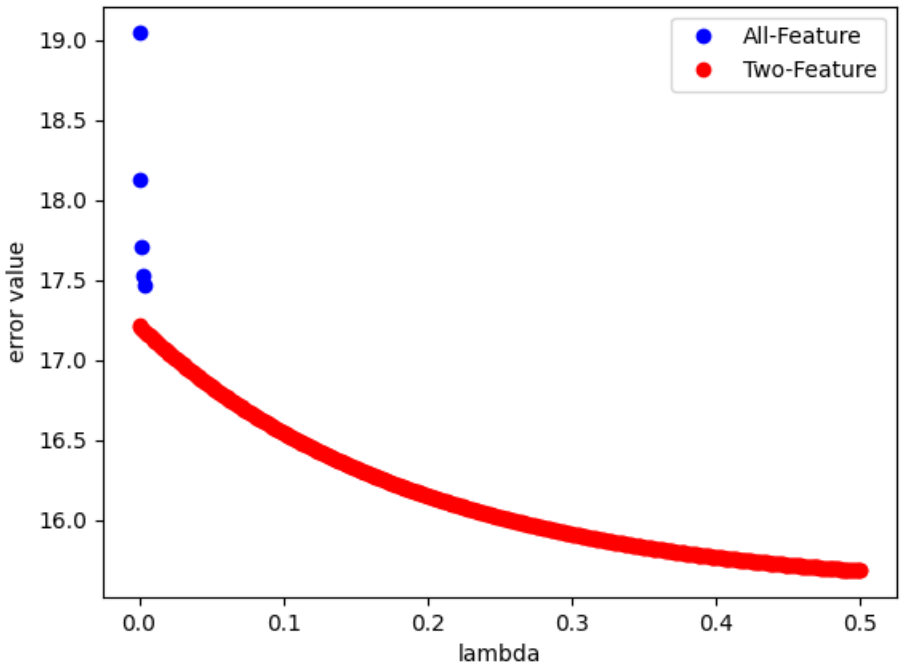

Figure 3. Last Iteration Training Error Value vs. Lambda (Run 2)

```
All-Feature Average Errors
Training Error:  0.150075757575758
Validation Error:  0.19641791044776116
Testing Error:  0.19380000000000003
Two-Feature Average Errors
Training Error:  0.15886363636363635
Validation Error:  0.16686567164179106
Testing Error:  0.16849999999999998
```

Figure 4. Error Rates (Run 2)

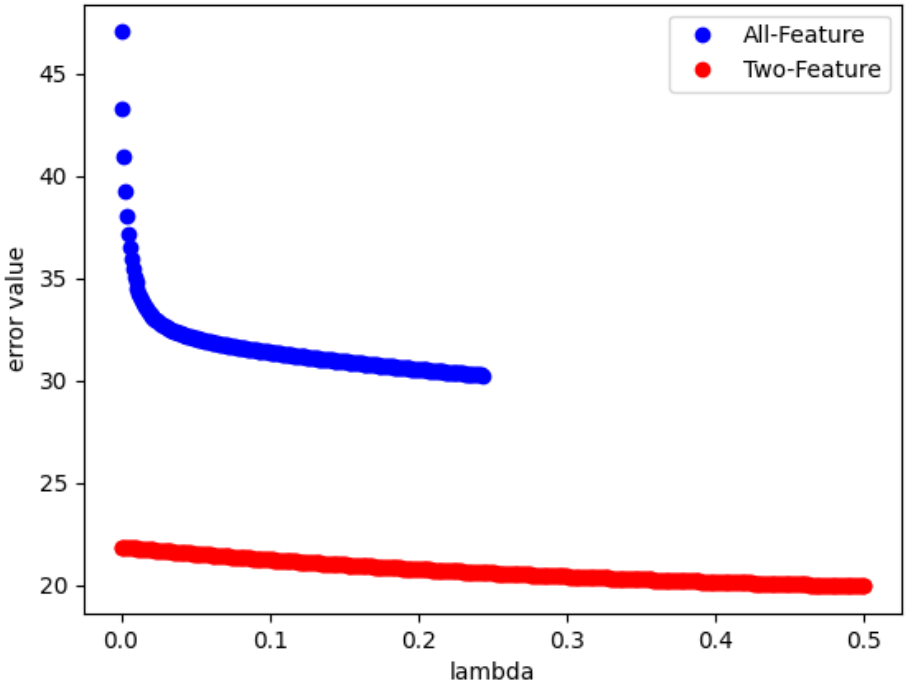

Figure 5. Last Iteration Training Error Value vs. Lambda (Run 3)

```
All-Feature Average Errors
Training Error:  0.1509848484848485
Validation Error:  0.19298507462686565
Testing Error:  0.1987
Two-Feature Average Errors
Training Error:  0.15719696969696972
Validation Error:  0.1705970149253731
Testing Error:  0.17520000000000008
```

Figure 6. Error Rates (Run 3)