CS235 - Data Mining

Scratch Implementation of Logistic Regression
and Comparison with Scikit implementation

Project - Fall 2022

1 Logistic Regression

The following Equation 1.1 expresses sigmoid function that is used in logistic regression. Using this equation, we will get value of y ranging from 0 to 1, which is later converted to either 0 or 1 based on its closeness. x represents the regression equation.

$$y = \frac{1}{1 + (e)^{-x}} \tag{1.1}$$

For two class, probability of a datapoint, x, belonging to a class, C_1, is -

$$P(C_1|x) = f(x) = \sigma(w^T x + b) \tag{1.2}$$

here, $x = [x_1, x_2, x_3, \dots, x_m]^T$ is a data point of m dimension, $w = [w_1, w_2, w_3, \dots, w_m]^T$ is weight vector and b is bias, σ is the activation function shown in Equation 1.1.

However, for multi-class classification we cannot use the sigmoid function due to its restriction of quantifying between 0 and 1. We can use **softmax function** in this case. Now, probability of a datapoint, x, belonging to a class, C_i, from k number of classes is -

$$P(C_i|x) = f(x) = \frac{e^{a_i}}{\sum_{j=1}^{k} e^{a_j}} \tag{1.3}$$

$$a_i = w_i^T x + b_i, (i = 1, 2, \dots, k) \tag{1.4}$$

where, $w_i = [w_{i1}, w_{i2}, \dots, w_{im}]$ are the weights for class i, $a = a_1, a_2, \dots, a_k$ are the regression for each class, and $b = b_1, b_2, \dots, b_k$ are the bias for each class.

2 Regularization Technique

Regularization is a technique used in Machine Learning methods to address the **overfitting** issue. This is a simple procedure of adding a penalty in the cost function. Two types of regularization are very common -

1. Lasso/L1 regularization
2. Ridge/L2 regularization

2.1 L1 regularization

It adds the absolute magnitude ($\lambda \sum_{j=1}^{m} |w_j|$) of the coefficients as penalty to the cost function. So, our cost function will become Equation 2.1. Here λ is regularization parameter that controls the degree of overfitting.

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} Cost(h(x^i) - y^i) + \frac{\lambda}{n} \sum_{j=1}^{m} |w_j| \tag{2.1}$$

It does not prioritize any model parameters to minimize where L2 prioritize to minimize large parameters. In L1, the shrinking is constant and so we will get lots of zeros (sparse matrix) for the ones which are least useful for minimizing loss.

2.2 L2 regularization

It adds the squared magnitude ($\lambda \sum_{j=1}^{m} w_j^2$) of the coefficients as penalty to the cost function. So, our cost function will become Equation 2.2.

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} Cost(h(x^i) - y^i) + \frac{\lambda}{2n} \sum_{j=1}^{m} w_j^2 \tag{2.2}$$

And the gradient will be calculated using Equation 2.3. We see that large value of w_i will cause large penalty and smaller value will cause small penalty. Large value of λ will cause w_i to shrink close to 0.

$$\frac{\delta}{\delta w_i} J(w) = \frac{1}{n} [\sum_{j=1}^{n} (h(x^j) - y^j) x_i^j + \lambda w_i] \tag{2.3}$$

3 Report

1. **Off-the-shelf baseline**:

   For the off-the-shelf implementation of Logistic Regression, Sklearn python library is used. The details of this library is available at this link - `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

   Input settings:

   - *random_state*: 0 was set for reproduction of the same result.
   - *penalty*: 'l1' or 'l2'.
   - *solver*: 'liblinear' which supports both penalty.

2. **Potential modifications to the existing dataset and task**: Describe any modifications you made to the general problem definition. For instance, if you only focused on a specific set of features, or you turned a classification problem to regression or vice-versa, state it here. Otherwise add "nothing to report".

3. **Preprocessing**:

(a) Normalization: It transfroms multi-scaled data to a single scale. Two types of normalization was applied to the dataset and compared to select one.

i. *MinMax Normalization*: It rescales the values of a feature within a range [0, 1] using the max and min value of a feature in the dataset. The formula is shown in Equation 3.1.

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

ii. *z - normalization*: It transforms the feature values with a mean of 0 and standard deviation 1. The transformed feature vector follows the original distribution. The formula is shown in Equation 3.2

$$x_{new} = \frac{x - \mu}{\sigma}; here, \mu = mean, \sigma = std.deviation \tag{3.2}$$

(b) Feature Selection:

i. *Correlation:* Correlation between all the features were calculated. If there is correlation value is equal or more than 0.9 between two feature, only one will be used for training. The other one will be removed from the dataset.

(c) Feature Representation Learning:

i. *Principal Component Analysis (PCA):* It is an unsupervised linear dimension reduction method that transforms a set of correlated variables into a smaller set of uncorrelated variables maintaining the variation of the original dataset.

ii. *t-distributed Stochastic Neighbor Embedding (t-SNE):* t-SNE is a non-linear dimension reduction technique that is commonly used for data visualization.

iii. *Factor Analysis (FA):* This not only reduces dimensions, but also try to find latent variables that can be derived from other variables in the dataset.

iv. *Linear Discriminant Analysis (LDA):* LDA is supervised dimension reduction technique applicable for multi-class dataset. It tries to find out linear combination of input features which separates the classes better.

(d) Sampling:

i. *Stratified:* The dataset was divided into train and test set so that the distribution of data from each classes in train and test set follows the original distribution.

4. **Experimental setting**: Describe the exact set-up of your experimental evaluation. What metrics did you use and why, how were the results you are presenting computed (did you do cross-validation?) etc.

5. **Results**: Here you should include tables/figures that show the performance of your chosen baseline across the chosen performance metrics. Please make sure you include error-bars in any plot, or ± standard deviation in any table, as we discussed in class.

4 Logistic Regression implemented by Faisal Bin Ashraf

- **Algorithm**: I have implemented multi-class logistic regression using gradient descent. I have set the parameters as follows -
  - Number of iteration, maxiter=1000,
  - Learning Rate, eta=0.1,
  - Regularization weight, mu=0.01

  The steps are given below.

  1. calculate the product of X and W. X is the sample set, and W is weight matrix.
     $Z = -X.W$
  2. Calculate the softmax of each row in X. It represents the probability of the sample in each class.
     $P_i = softmax(Z_i)$
  3. calculate the loss
     $loss(W) = -\frac{1}{N} log(P(Y|X,W))$
     $= \frac{1}{N}(\sum_{i=1}^{N}(X_i W_{k=Y_i} + log \sum_{k=0}^{c} exp(-X_i W_k)))$

     Add 'l1' or 'l2' regularization terms with this loss based on the given parameter. For'l2', add $\mu||W||^2$ and, for 'l1' add dd $\mu||W||$
  4. calculate the gradient of W and Update W.
  5. Go to Step 1 and follow the whole process for a given number of iterations.
- **Baseline**: I have used sklearn library for off-the-shelf logistic regression baseline (link).
- **Potential discrepancies**: I have used the default configuration of the parameters for this baseline implementation except setting solver to 'liblinear', so that 'l1' and 'l2' penalty can be introduced. I changed the parameter of 'penalty' to 'l1' and 'l2' to check their effects. Table 1 shows the discrepancies.

Table 1: Discrepancies between off-the-shelf and scratch implementation

Parameter	Baseline	My Implementation
tolerance	1×10^{-4}	None
fit_intercept	True	No bias
solver	*liblinear*	None
max_iter	100	1000
multi_class	*ovr*	$softmax$

- **Measures of comparison**: I have calculated the performance of both implementations using - Accuracy, F1-score, Precision, and Recall. Our multi-class dataset is not balanced, which is why our main comparison metric is **F1-score** which considers precision and recall to check if the samples from small classes are correctly predicted.
- **Experimental results**: Table 2 and Table 3 shows the 10-fold cross-validation results of baseline off-the-shelf classifier with L1 and L2 regularization respectively. Performance evaluation metrics for scratch implementation with both L1 and L2 regularization are

shown in Table 4 and Table 5. Figure 1 shows the comparison of my implementation with the baseline for both regularizations.

Table 2: Baseline scores of Logistic Regression with L1 Regularization

Data Representation	Accuracy	F1	Precision	Recall
Original Data	0.989 ± 0.0007	0.9653 ± 0.0026	0.9666 ± 0.0025	0.9666 ± 0.0025
z-Normalized data	0.9918 ± 0.0008	0.9724 ± 0.0029	0.9732 ± 0.0028	0.9732 ± 0.0028
MinMax Normalized	0.9918 ± 0.0008	0.9725 ± 0.0029	0.9732 ± 0.0028	0.9732 ± 0.0028
PCA	0.9425 ± 0.0016	0.7946 ± 0.0081	0.8032 ± 0.0094	0.8032 ± 0.0094
LDA	0.9871 ± 0.0009	0.9612 ± 0.0029	0.9657 ± 0.0025	0.9657 ± 0.0025
tSNE	0.7829 ± 0.0021	0.2195 ± 0.0003	0.2016 ± 0.0006	0.2016 ± 0.0006
FA	0.9875 ± 0.0008	0.9609 ± 0.0024	0.9647 ± 0.0022	0.9647 ± 0.0022

Table 3: Baseline scores of Logistic Regression with L2 Regularization

Data Representation	Accuracy	F1	Precision	Recall
Original Data	0.9891 ± 0.0009	0.9658 ± 0.0033	0.9674 ± 0.0032	0.9674 ± 0.0032
z-Normalized data	0.991 ± 0.0009	0.9707 ± 0.0030	0.9719 ± 0.0031	0.9719 ± 0.0031
MinMax Normalized	0.991 ± 0.0009	0.9707 ± 0.0030	0.9719 ± 0.0031	0.9719 ± 0.0031
PCA	0.9422 ± 0.0016	0.7936 ± 0.0081	0.8026 ± 0.0096	0.8026 ± 0.0096
LDA	0.9869 ± 0.0009	0.9602 ± 0.0029	0.9647 ± 0.0025	0.9647 ± 0.0025
tSNE	0.7831 ± 0.0021	0.2195 ± 0.0003	0.2016 ± 0.0006	0.2016 ± 0.0006
FA	0.9869 ± 0.0008	0.9602 ± 0.0023	0.9647 ± 0.0019	0.9647 ± 0.0019

Table 4: Scratch implementation scores of Logistic Regression with L1 Regularization

Data Representation	Accuracy	F1-Score	Precision	Recall
z-Norm	0.985 ± 0.001	0.953 ± 0.003	0.949 ± 0.004	0.959 ± 0.003
min-max Norm	0.985 ± 0.001	0.953 ± 0.003	0.949 ± 0.004	0.959 ± 0.003
PCA	0.949 ± 0.002	0.824 ± 0.007	0.815 ± 0.008	0.839 ± 0.005
LDA	0.990 ± 0.001	0.968 ± 0.003	0.969 ± 0.003	0.968 ± 0.003
FA	0.986 ± 0.007	0.957 ± 0.002	0.962 ± 0.002	0.954 ± 0.003
Original	0.955 ± 0.008	0.894 ± 0.009	0.916 ± 0.011	0.885 ± 0.007

- **Justification of discrepancies**: From Fig 1 we can see that the performance of the scratch implementation is less than the off-the-shelf classifiers. However the performance degradation is not drastic and for some data representation (LDA, FA) it is quite similar to the sklearn. The discrepancies of model parameters and the use of optimizers in sklearn are the reasons behind this performance gap. Figure 2 shows the difference between sklearn performance and my implementation. The lines are close to zero (0), indicate that the difference is very low. Moreover, in some cases (PCA, LDA) my implementation performs better than the library implementation.

Table 5: Scratch implementation scores of Logistic Regression with L2 Regularization

Data Representation	Accuracy	F1-Score	Precision	Recall
z-Norm	0.975 ± 0.001	0.925 ± 0.005	0.919 ± 0.005	0.937 ± 0.005
min-max Norm	0.975 ± 0.001	0.925 ± 0.005	0.919 ± 0.006	0.937 ± 0.006
PCA	0.945 ± 0.002	0.813 ± 0.009	0.806 ± 0.008	0.831 ± 0.008
LDA	0.989 ± 0.001	0.965 ± 0.003	0.966 ± 0.003	0.966 ± 0.003
FA	0.986 ± 0.001	0.957 ± 0.003	0.962 ± 0.002	0.955 ± 0.003
Original	0.921 ± 0.006	0.647 ± 0.043	0.647 ± 0.055	0.701 ± 0.036

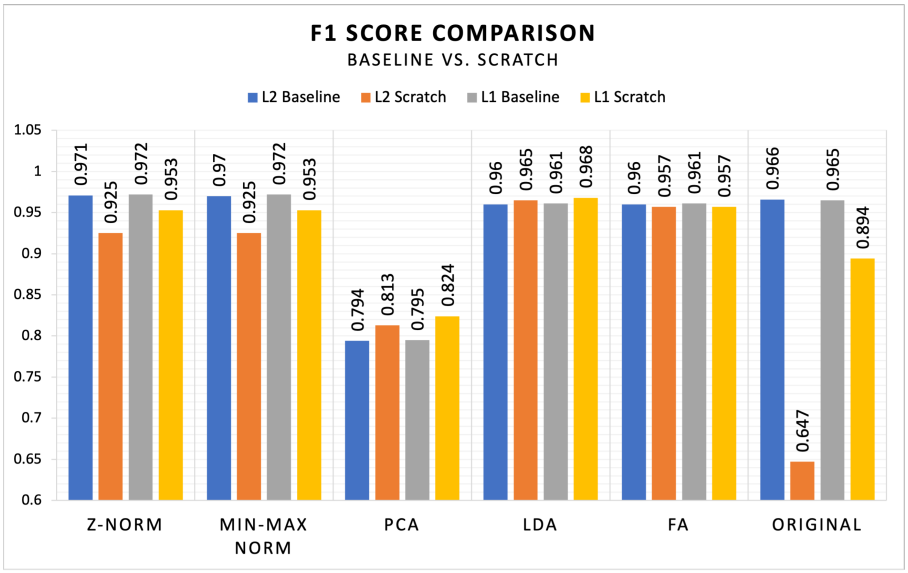

Figure 1: F1-score comparison between off-the-shelf and scratch implementation of Logistic Regression Model

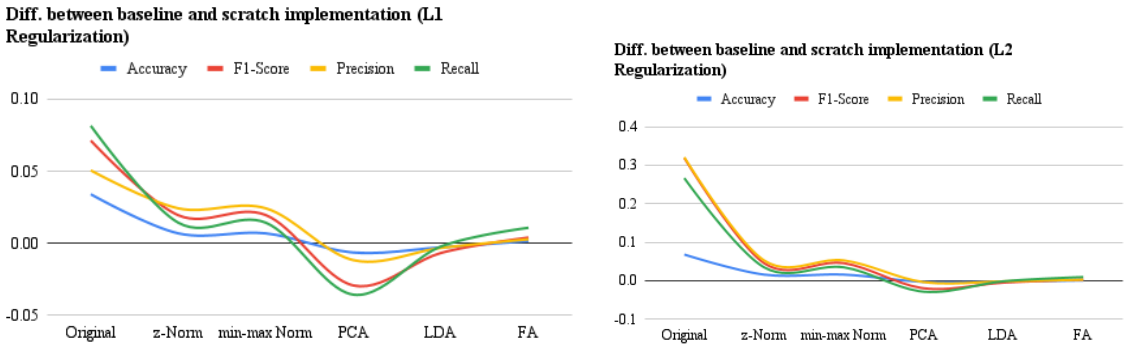

Figure 2: Difference of performance between baseline and scratch (baseline score - scratch score)