Robust Machine Learning Models for Long Tailed Distributions.

Navid Madani
University at Buffalo
Buffalo, New York, U.S.A
smadani@buffalo.edu

Sai Chandra Rachiraju
University at Buffalo
Buffalo, New York, U.S.A
srachira@buffalo.edu

Abstract

Modern real-world large-scale data-sets often have long-tailed label distributions.As the frequency of a particular category decreases, the average model performance gets effected. It is often difficult, to find well-balanced labelled data,but it's critical for algorithms to perform well on such edge cased classes too. Imagine an auto-pilot vehicle misclassifying an object, due to lack of training data. This could lead to fatal consequences and thus training accurate traditional machine learning models and Deep Neural Networks in presence of long tailed and noisy labels is of great importance in the current Machine Learning. scenario. For this project, we chose M.N.I.S.T [1]*. (Modified National Institute of Standards and Technology) handwritten digits to construct various long tailed, noisy data to train it on conventional Machine Learning Models and D.L.L along with a few contributions.*

1. Introduction

1.1. Data processing

M.N.I.S.T Data-set has been downloaded from PyTorch Data-sets and have been randomly splitted into training and testing with sizes of 60k and 10k respectively.

To train the data on PyTorch, we had to construct tensors from the array.Once these tensors are loaded, the pixel data can take values from 1 to 255. When the ranges of values of the input data is not the same, the higher range values dominates the other values, tending the slopes of its weights and biases to incline more. To prevent this, the data has been normalized with a mean and a standard deviation of 0.1307 and 0.3081 respectively.To train our models on long tailed distributions with noise, we had to first create balanced and imbalanced data-sets. The original M.N.I.S.T seems to have a fairly even distribution amongst its classes and thus we considered the original data-set as balanced.

[1]http://yann.lecun.com/exdb/mnist/

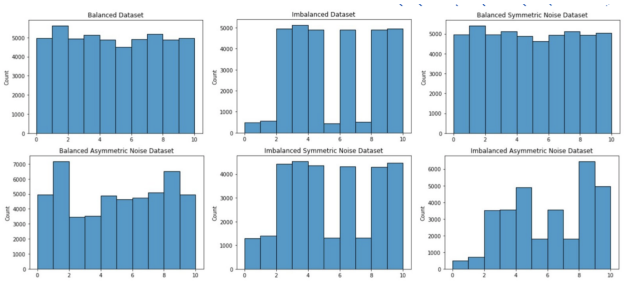

Figure 1. Data distribution along with noise.

To create an imbalance a predefined weights dictionary of imbalanced_weights has been made. The predefined weights are skewed to a large extent to create Long tailed distributions. The classes 0,1,5,7 are skewed to 10% and showed great imbalance. All the labels with a probability of 30% are flipped in a symmetric way to create symmetric noise.

For example if 1 is mislabelled as 9, then 9 is also mislabelled to be 1. By flipping only 30 percent of this data, noise is being introduced, while major portion of the labels are left intact. The following transformation is used to create asymmetric noise - $2 \rightarrow 7$, $3 \rightarrow 8$, $5 \leftrightarrow 6$ and $7 \rightarrow 1$. It has to be noted that the labels 5 and 6 are symmetrically flipped to introduce symmetric noise even inside the asymmetric noise.

Once the datasets are created baseline Machine Learning models such as Logistic Regression and Random Forests are trained. LDAM-DRW [2].

and SL are two Deep Learning Models trained and observed. Once these results are collected, they are analysed and improved to create proposed ML and proposed DL models. The code for reproducing all the datasets, training baseline models and proposed models and generating evaluation metrics are public on our github repository[3].

[2]https://arxiv.org/abs/1906.07413
[3]https://github.com/navidmdn/cse555-final-project

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Figure 2. Cost function for Logistic Regression

2. Baseline Models

In this section we will give and overview of the baseline models that we used in this project. For baseline ML models, we decided to use Logistic Regression and Random Forest which we explain in more detail in 2.1 and 2.2 respectively. Also to baseline two deep learning based models we used label-distribution aware margin loss and symmetric learning via symmetric cross entropy which are explained in detail in 2.3 and 2.4 respectively.

2.1. Logistic Regression

The logistic function was developed in the 1830's and 1840's to model population growth and was named Logistic by Pierre Verhulst.It is a statistical method used for classification problems. Dependent variable having two categorical levels are called as binary logistic regression model, and more then two are called as multinomial logistic regression. M.N.I.S.T being a digit recognition problem has 10 digits or classes and thus multinomial logistic regression is used. This model is an extension of the linear model, where the linear relationship is assumed and is passed into a logit function, that outputs the probabilities of the input falling into a group.

The cost of a learning algorithm is the formal mathematical representation of an objective the learning algorithm is trying to achieve.It checks what the average error is between the actual class and the predicted class. The cost function not only penalized big errors but also errors which are too confident. So our objective to train the model is to achieve the minimum cost possible by adjusting the weights and biases of the model.

Gradient descent is a method of changing weights based on the loss function for each output. Taking the partial derivative of the weights and the biases we update the existing values.

It is the responsibility of the gradient descent algorithm we use to not stuck at any saddle points, but to find the global minimum cost involved in training the algorithm. One big factor here is the learning rate, the rate at which the gradient descent algorithm processes.So higher rate might overshoot the learning process and pass by the minimum point and never converge back. Lower learning rate might never converge to the bottom of the valley and might cost processing time in this.

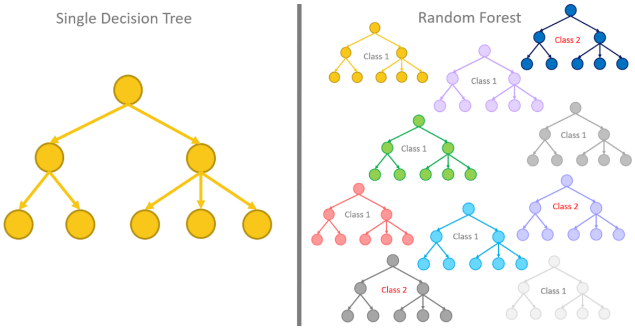

Figure 3. Random Forest vs Single Decision Tree

2.2. Random Forest

Random forests come under the classification umbrella in Machine Learning problems, where decision trees are the fundamental building block behind it. A random forest model consists of large number of these decision trees working as an ensemble. Splitting a data set into different parts, can be done on many of the features, and hence splitting criterion such as Gini Impurity, Information Gain and Chi-Square has been introduced. On each split, the random forest ensures, that homogeneity can be achieved,and thus reducing the entropy or randomness of the given data.On the theory that a large number of relatively uncorrelated trees operating together can perform any individual trees, Random Forest can be used to classify our data with a criterion and depth. Due to this non-linear splitting of data, RF's at times produces far more accuracy than any linear fitting model.With 100 estimators and a max depth of 10,all the given datasets, have been trained and tested. Random forest uses both bagging and feature randomness to create an uncorrelated forest of decision trees.The algorithm has three different hyper-parameters which include the node size, number of trees and number of features sampled. So in our random forest we not only get trees that are trained on different sets of data but also use different features to make decisions. Random forests have lower generalization errors. This algorithm is relatively robust to outliers and noise and is faster than bagging or boosting. Random forests are simple and easily parallelized.

2.3. Label-Distribution-Aware Margin Loss

The idea of this paper [1] is motivated by minimizing a margin-based generalization bound and replacing it as a loss function instead of the standard cross-entropy objective during training. The idea is a bit similar to the idea of SVM but different in the sense that the margin would be dynamic with respect to how balanced the training dataset classes are. In the paper, they mathematically prove that if we wish to improve the generalization of minority classes, we should enforce bigger margins γ_j's for them. However, since this can

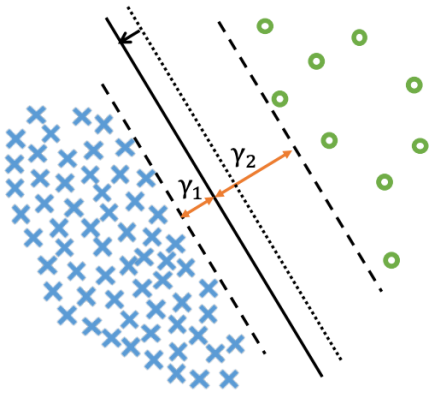

Figure 4. For binary classification with a linearly separable classifier, the margin γ_i of the i-th class is defined to be the the minimum distance of the data in the i-th class to the decision boundary. We show that the test error with the uniform label distribution is bounded by a quantity that scales in $\frac{1}{\gamma_1\sqrt{n_1}} + \frac{1}{\gamma_2\sqrt{n_2}}$ As illustrated here, fixing the direction of the decision boundary leads to a fixed $\gamma_1+\gamma_2$, but the trade-off between γ_1 and γ_2 can be optimized by shifting the decision boundary

hurt the generalization of more frequent classes, we should come up with an optimal point to address this trade-off. In the paper, they provide proofs that for the binary case, the optimal point will be given by 1. In which C is a hyper parameter and n_j is the number of samples in class j.

$$\gamma_j = \frac{C}{n_j^1/4} \tag{1}$$

They extend this work by applying the same margin to multi-class setup and providing the extended loss function in 2 for standard cross-entropy.

$$\begin{aligned} L_{LDAM}((x,y);f) &= -log\frac{e^{z_y-\Delta_y}}{e^{z_y-\Delta_y}+\Sigma_{j\neq y}e^{z}_{j}} \\ where\Delta_j &= \frac{C}{n_j^1/4} for j \in 1,...,k \end{aligned} \tag{2}$$

2.4. Symmetric Learning via Symmetric Cross Entropy loss

In this paper, authors argue that learning DNNs with Cross Entropy loss exhibits overfitting to noisy labels on some classes and also it suffers from significant under learning on some other classes. They suggest that by adding a noise tolerant term, we can avoid underfitting and overfitting in these two classes. Inspired by symmetric KL-divergence, [4] they propose a **Symmetric cross entropy learning** method. The intuition behind the symmetric loss comes from the idea behind symmetric KL divergence. That is, when we are calculating the KL divergence between two probability distributions as shown in 3

$$KL(q||p) = H(q,p) - H(q) \tag{3}$$

where H denotes the cross-entropy, considering that in the classification case q presents the ground truth distribution and p presents the prediction distribution, in case of having noise in labels the q distribution is far from the ground truth distribution and at some point the prediction distribution can be more similar to the ground truth. That is why we also take the KL divergence in other direction and build symmetric KL-divergence as 4.

$$Symmetric - KL(q,p) = KL(q||p) + KL(p||q) \tag{4}$$

Motivated by this idea, the authors introduce symmetric cross entropy by defining loss as 5.

$$l_{sce} = \alpha H(p,q) + \beta H(q,p) \tag{5}$$

where α accounts for the overfitting issue of CE while β accounts for the flexibility of exploration on the robustness of training.

3. Proposed Models

In this section we introduce two proposed approaches that improve the results of our baseline models over imbalanced and noise datasets. 3.1 will introduce a method that embeds the feature space into an embedded space using UMAP [3] and then trains the random forest over it and builds a more robust model against noise and imbalanced data. 3.2 describes the contribution of our project towards the deep learning model introduced in 2.4 by adding a new loss function that better shapes the objective towards noisy and imbalanced datasets.

3.1. Embedded Random Forest

The intuition behind this contribution was that since the embedding generation of data points using UMAP is **unsupervised and it does not require any labels** it is likely that the data points in the embedding space are more robust and consequently, the trained model with noise in the dataset or the lack of data in some classes is more robust and error prune. In 5 we plotted the generated data points by UMAP on the first two dimensions to show the separability of classes. After generating embeddings in 5d space we try to follow the same procedure in 2.2 and train a random forest model on top of the embeddings. Our experiments show a huge boost in accuracy, precision, recall and F1 score of this model comparing to vanilla random forest on all the six datasets.

[4] https://en.wikipedia.org/wiki/KullbackLeibler_divergence

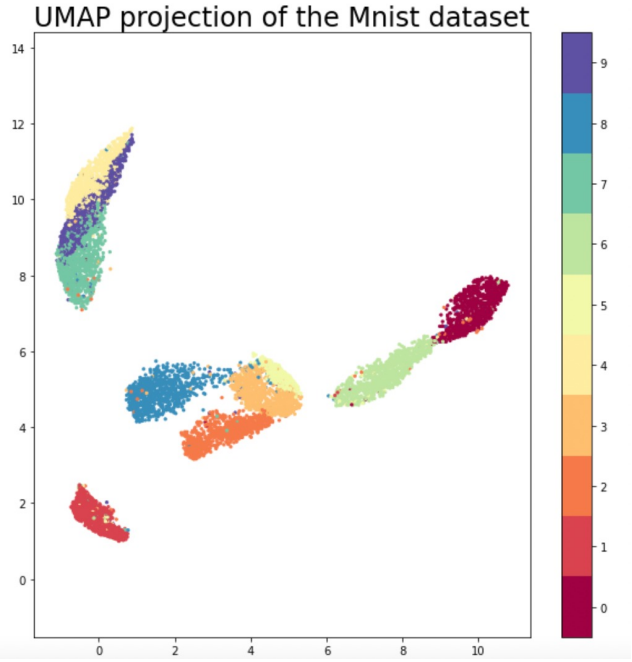

Figure 5. Distribution of data points along the first two dimensions of UMAP embeddings

3.2. Symmetric Cross Entropy with Focal Loss

Focal loss is trick over cross entropy loss proposed in [2]. This new loss function penalizes error in classification of well classified samples less than those that are hard to classify and regulates this by setting a γ factor. Figure 6 compares focal loss and cross entropy with different gammas. To extend the symmetric cross entropy loss with this function, we added focal loss to the loss function and rewrite the objective using the modification in 6 to cross entropy function ($CE(p_t) = -log(p_t)$) for each class p_t

$$Focal - loss(p_t) = -(1 - p_t)^{\gamma} log(p_t) \tag{6}$$

The results show some improvements in three of the six datasets which we'll discuss in 4

4. Experiments

In this section we provide the results of our experiments and the experiment setups that produced them. First, we will provide baseline results and then we will provide the improvements that our proposed models caused.

4.1. Baseline Models

For Logistic regression model, we both implemented it from scratch in pytorch and also implemented it using scikit-learn and the results were completely identical. Both implementations are available in the source code. To set

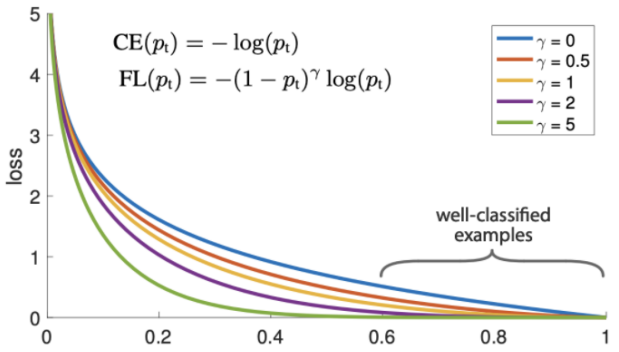

Figure 6. Comparison of focal loss in different gammas with cross-entropy. Setting $\gamma > 0$ reduces the relative loss for well-classified samples.

hyper parameters of our models we randomly chose **10 percent of training dataset of MNIST as validation** set and set hyper-parameters of our model accordingly. For logistic regression model we set a learning rate of 0.1 and L2 regularization and chose to train the model in **one versus rest** fashion that results in 10 sets of parameters for the logistic regression. For random forest model, we chose 100 tree estimators and trees of maximum depth 10. The results of our final model on test dataset are shown in 7. Also, for baseline deep learning models, we used the **early stopping mechanism** on validation dataset to fine-tune the hyper parameters and avoid over-fitting. For both DL models, we used Adam optimizer with an initial learning rate of 1e-4 and L2 regularization. For LDAM model, following the original implementation, we used a resnet-32 architecture and for SCE model, [5] we followed the same custom architecture and hyper parameters that was shown in the original paper and set $\alpha = 0.1$ and $\beta = 1.0$ for loss function. 8 represents the final results of our best model on test dataset. We trained each of the deep learning baseline models on **Nvidia A100 GPU for about an hour**.

4.2. Proposed Models

As described in the previous section, for proposed ML model we ensembled random forest and UMAP methods to create a new approach we call **embedded-rf** the architecture and hyper parameters of the random forest model are remained enact to be comparable to baseline and for UMAP we chose to set euclidean distance and a projection to 5d space. Also, for proposed DL model and the focal loss that we have used, we set gamma to 2 and train the model with all the same hyper parameters to be comparable to baselines. Same as baselines, we trained our models on **Nvidia-A100 GPU for about an hour**. 9 shows the

[5] https://arxiv.org/abs/1908.06112

DL Baselines	Balanced	Imbalanced	Balanced Symmetric	Balanced Asymmetric	Imbalanced Symmetric	Imbalanced Asymmetric
LDAM	Accuracy - 0.9894 Precision - 0.9895 Recall - 0.9894 F1 Score - 0.9894	Accuracy - 0.9894 Precision - 0.9895 Recall - 0.9894 F1 Score - 0.9894	Accuracy - 0.9871 Precision - 0.9874 Recall - 0.9871 F1 Score - 0.9871	Accuracy - 0.9894 Precision - 0.9894 Recall - 0.9894 F1 Score - 0.9894	Accuracy - 0.9823 Precision - 0.9826 Recall - 0.9823 F1 Score - 0.9823	Accuracy - 0.9862 Precision - 0.9863 Recall - 0.9862 F1 Score - 0.9862
SCE Model	Accuracy - 0.9862 Precision - 0.9864 Recall - 0.9862 F1 Score - 0.9862	Accuracy - 0.9952 Precision - 0.9952 Recall - 0.9952 F1 Score - 0.9952	Accuracy - 0.9924 Precision - 0.9925 Recall - 0.9924 F1 Score - 0.9924	Accuracy - 0.9903 Precision - 0.9904 Recall - 0.9903 F1 Score - 0.9903	Accuracy - 0.9900 Precision - 0.9901 Recall - 0.9900 F1 Score - 0.9900	Accuracy - 0.9840 Precision - 0.9841 Recall - 0.9840 F1 Score - 0.9840

Figure 7. Accuracy, precision, recall and F1 score of baseline DL models on the six proposed datasets.

ML Baselines	Balanced	Imbalanced	Balanced Symmetric	Balanced Asymmetric	Imbalanced Symmetric	Imbalanced Asymmetric
ML1 -Random Forest	Accuracy - 0.9476 Precision - 0.9479 Recall - 0.9476 F1 Score - 0.9476	Accuracy - 0.8360 Precision - 0.8712 Recall - 0.8360 F1 Score - 0.8213	Accuracy - 0.9461 Precision - 0.9464 Recall - 0.9461 F1 Score - 0.9461	Accuracy - 0.9333 Precision - 0.9358 Recall - 0.9333 F1 Score - 0.9334	Accuracy - 0.8290 Precision - 0.8679 Recall - 0.8290 F1 Score - 0.8100	Accuracy - 0.8127 Precision - 0.8623 Recall - 0.8127 F1 Score - 0.7932
ML2 - Logistic Regression	Accuracy - 0.9178 Precision - 0.9168 Recall - 0.9166 F1 Score - 0.9166	Accuracy - 0.8813 Precision - 0.8894 Recall - 0.8788 F1 Score - 0.8782	Accuracy - 0.8902 Precision - 0.8904 Recall - 0.8902 F1 Score - 0.8897	Accuracy - 0.8617 Precision - 0.8696 Recall - 0.8617 F1 Score - 0.8612	Accuracy - 0.7725 Precision - 0.8096 Recall - 0.7725 F1 Score - 0.7580	Accuracy - 0.7739 Precision - 0.8015 Recall - 0.7739 F1 Score - 0.7582

Figure 8. Accuracy, precision, recall and F1 score of baseline ML models on the six proposed datasets.

Proposed ML	Balanced	Imbalanced	Balanced Symmetric	Balanced Asymmetric	Imbalanced Symmetric	Imbalanced Asymmetric
Random Forest	Accuracy - 0.9476 Precision - 0.9479 Recall - 0.9476 F1 Score - 0.9476	Accuracy - 0.8360 Precision - 0.8712 Recall - 0.8360 F1 Score - 0.8213	Accuracy - 0.9461 Precision - 0.9464 Recall - 0.9461 F1 Score - 0.9461	Accuracy - 0.9333 Precision - 0.9358 Recall - 0.9333 F1 Score - 0.9334	Accuracy - 0.8290 Precision - 0.8679 Recall - 0.8290 F1 Score - 0.8100	Accuracy - 0.8127 Precision - 0.8623 Recall - 0.8127 F1 Score - 0.7932
Embedded Random Forest	Accuracy - **0.9562** Precision - 0.9569 Recall - 0.9562 F1 Score - 0.9561	Accuracy - **0.8966** Precision - 0.9095 Recall - 0.8966 F1 Score - 0.8932	Accuracy - **0.9543** Precision - 0.9550 Recall - 0.9543 F1 Score - 0.9542	Accuracy - **0.9543** Precision - 0.9549 Recall - 0.9543 F1 Score - 0.9542	Accuracy - **0.8932** Precision - 0.9077 Recall - 0.8932 F1 Score - 0.8894	Accuracy - **0.8956** Precision - 0.9077 Recall - 0.8956 F1 Score - 0.8923

Figure 9. Accuracy, precision, recall and F1 score of proposed ML models on the six proposed datasets. The improved results are shown in bold font

comparison of proposed ML model with its baseline and 10 shows the comparison of proposed DL model with its baseline and highlights the improvements.

5. Contributions

Data Loading and Balanced Symmetric Noise and Balanced Asymmetric Noise is done by- Navid Madani,

Proposed DL	Balanced	Imbalanced	Balanced Symmetric	Balanced Asymmetric	Imbalanced Symmetric	Imbalanced Asymmetric
SCE with Focal Loss	Accuracy - **0.9879** Precision - 0.9881 Recall - 0.9879	Accuracy - 0.9949 Precision - 0.9949 Recall - 0.9949	Accuracy - **0.9929** Precision - 0.9929 Recall - 0.9929	Accuracy - 0.9837 Precision - 0.9839 Recall - 0.9937	Accuracy - 0.9893 Precision - 0.9894 Recall - 0.9893	Accuracy - **0.9916** Precision - 0.9916 Recall - 0.9916
SCE Model	Accuracy - 0.9862 Precision - 0.9864 Recall - 0.9862 F1 Score - 0.9862	Accuracy - 0.9952 Precision - 0.9952 Recall - 0.9952 F1 Score - 0.9952	Accuracy - 0.9924 Precision - 0.9925 Recall - 0.9924 F1 Score - 0.9924	Accuracy - 0.9903 Precision - 0.9904 Recall - 0.9903 F1 Score - 0.9903	Accuracy - 0.9900 Precision - 0.9901 Recall - 0.9900 F1 Score - 0.9900	Accuracy - 0.9840 Precision - 0.9841 Recall - 0.9840 F1 Score - 0.9840

Figure 10. Accuracy, precision, recall and F1 score of proposed DL models on the six proposed datasets. The improved results are shown in bold font.

Creating imbalanced dataset and Imbalanced Symmetric Noise and Imbalanced Asymmetric Noise is done by Sai Chandra.

ML Model 1 - Random Forest is done by Navid.

ML Model 2 - Logistic Regression is done by Sai.

Deep Learning Model1 - LDAM Model is done by Navid.

Deep Learning Model2 - SCE Model is done by Sai.

UMAP Projection is done by Navid and Sai.

Proposed DL SCE with Focal Loss is done by Navid.

References

[1] Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. Learning imbalanced datasets with label-distribution-aware margin loss. In *Advances in Neural Information Processing Systems*, 2019.

[2] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020.

[3] Yisen Wang, Xingjun Ma, Zaiyi Chen, Yuan Luo, Jinfeng Yi, and James Bailey. Symmetric cross entropy for robust learning with noisy labels. In *IEEE International Conference on Computer Vision*, 2019.