Sub-Quadratic AUC Optimization

Kyle Rust

Northern Arizona University

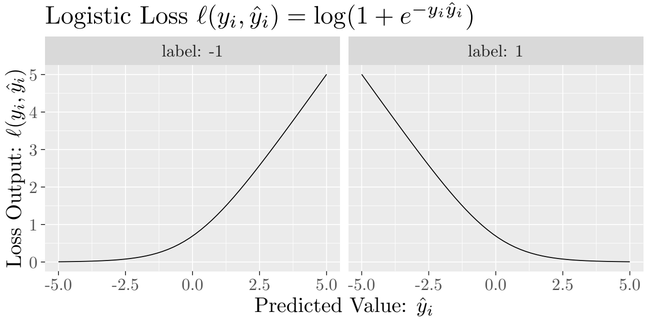
December 8, 2022

Introduction

- Binary classification aims to learn some function $f(x)$ to predict a positive or negative label
- To learn $f(x)$ usually involves some sort of optimization of an objective function
- If the labels in a data set are highly imbalanced, training an effective model can be difficult
- Methods to help handle this class imbalance are SLOW, we proposed methods to do it faster and just as successfully

Standard Process

- To solve a binary classification, we want to learn some function $f : \mathbb{R}^p \rightarrow \mathbb{R}$, where p is the number of features
- The real-valued predictions are computed by $\hat{y}_i = f(x_i)$

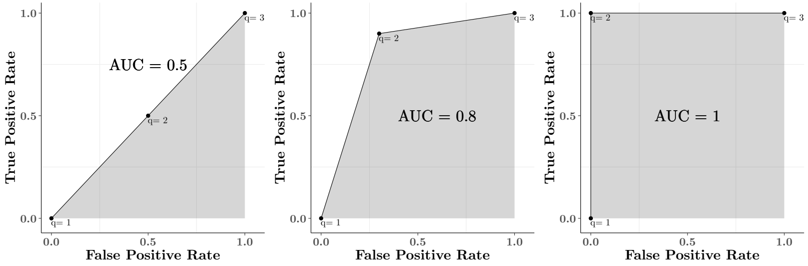


Binary Classification Examples

- Binary classification problems appear in all sorts of different domains
- *Ex:*
  - Classifying emails as spam or not spam
  - Determine whether a image contains a cat or a dog
  - Identifying invasive lake trout in Yellowstone Lake from LiDAR data
- Real world data sets tend to be highly imbalanced
- For the example the fish data set has a ratio of positive to negative labels between 0.001 and 0.003 (Vannoy et al., 2021)

Receiver Operating Characteristic Curve

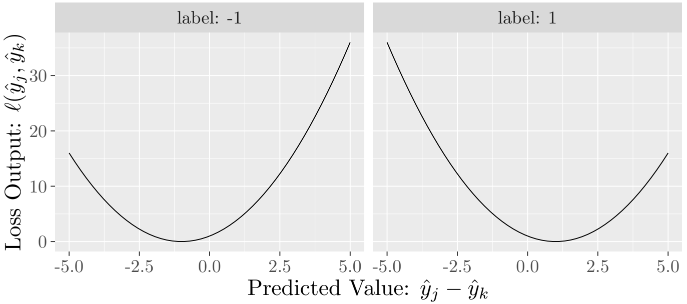
- The Receiver Operating Characteristic Curve (ROC) is a plot of False Positive Rate (FPR) vs True Positive Rate (TPR)
- Bamber (1975) proves that maximizing the AUC is equivalent to minimizing the number of incorrectly ranked pairs
- AUC maximization is an effective way to increase model performance on highly imbalanced data sets (Yuan et al., 2021)



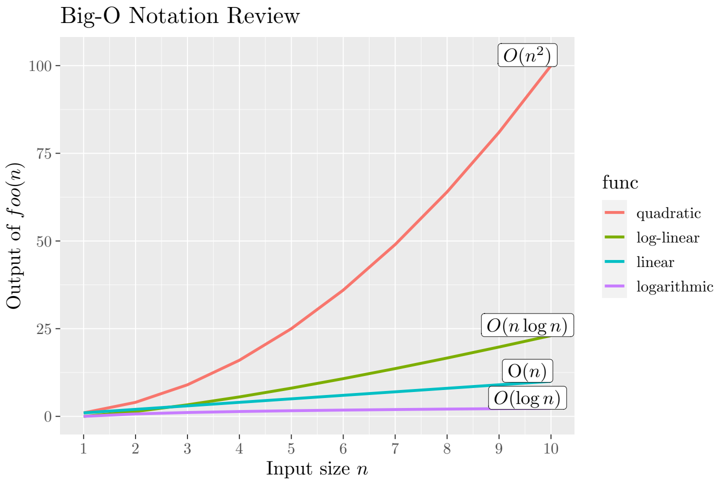
Reason For Research Study

- Pairwise loss functions sum over all pairs of the positive examples and negative examples
- $\mathcal{L}(f) = \sum_{j\in\mathcal{I}^+} \sum_{k\in\mathcal{I}^-} \ell[\hat{y}_j - \hat{y}_k]$

Square Loss $\ell(\hat{y}_j, \hat{y}_k) = (\hat{y}_j - \hat{y}_k)^2$



Big-O Notation



Research Question

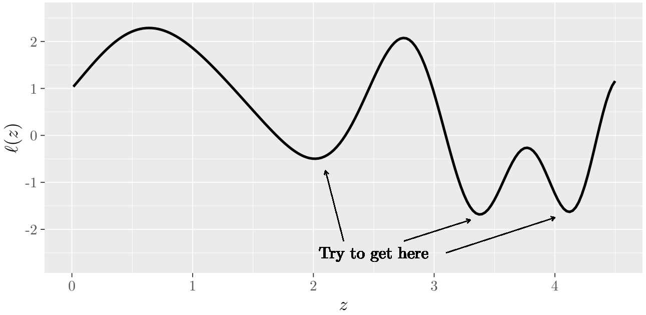
How beneficial is it to compute the square and squared hinge loss in sub-quadratic time?

Objective Function

- Machine learning, particularly deep-learning, involves one form of optimization
- The goal is to either maximize or minimize some function $\ell(x)$ by altering its input x
- This function $\ell(x)$ is referred to as an *objective function*
- If the goal is to minimize $\ell(x)$ this function is specifically referred to as a *loss function*

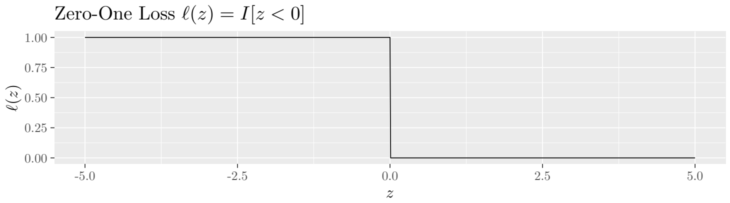
Gradient Descent

- One way to intelligently select $x \to \ell(x)$ is to use derivative information
- The derivative with respect to the loss is taken and that information is used to traverse to the minimum
- The derivative points in the direction of steepest *ascent*, so we go in the opposite direction for steepest *descent*



Convex Surrogate

- In order to make use of gradient for optimization a function must be differentiable and the derivative must be non-zero
- A function that can be optimized in place of the objective function is called a convex surrogate
- A function is convex if its local minima and maxima are also global minima and maxima



Related Work Summary

Sorting Algorithms: (Ferri et al., 2002; Cortes and Mohri, 2004)
Re-weighting: (Freund et al., 2003; Zhao et al., 2011)

Pairwise Approach	Degree	Hinge	Proof	Solution
Pahikkala et al. (2009)	Square	False	False	Functional
Joachims (2005)	Linear	True	True	Functional
Calders et al. (2007)	Polynomial	False	True	Functional
Ying et al. (2016)	Square	False	True	Min-Max
Yuan et al. (2020)	Square	True	True	Min-Max
This Work	Square	True	True	Functional

Represent Square Loss As Sum Over Coefficients

Basic definition of the square loss:

1. $\ell(z) = (m - z)^2$, $z = \hat{y}_j - \hat{y}_k$, $m =$ margin parameter

Substitute the definition of the square loss into the definition of the pairwise loss:

2. $\sum_{j \in \mathcal{I}^+} \ell(\hat{y}_j - \hat{y}_k) = \sum_{j \in \mathcal{I}^+} (m - \hat{y}_j + \hat{y}_k)^2$

Group margin and predicted values, expand the terms:

3. $= \sum_{j \in \mathcal{I}^+} ((m - \hat{y}_j) + \hat{y}_k)((m - \hat{y}_j) + \hat{y}_k)$

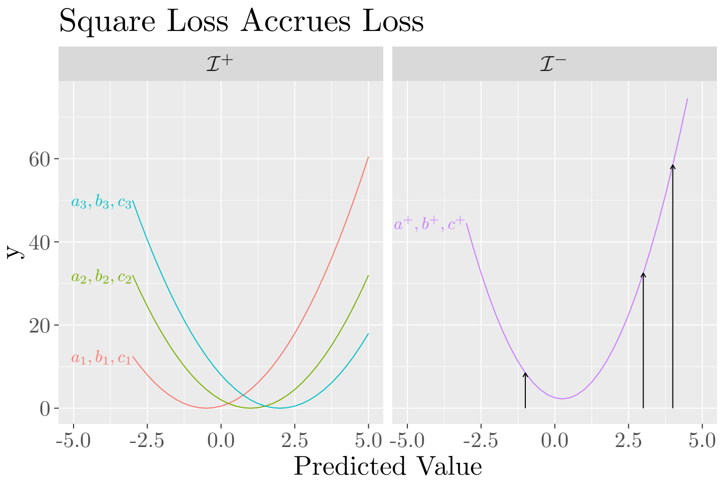
Factor the polynomial:

4. $= \sum_{j \in \mathcal{I}^+} \underbrace{1}_{a_j} \hat{y}_k^2 + \underbrace{2(m - \hat{y}_j)}_{b_j} \hat{y}_k + \underbrace{(m - \hat{y}_j)^2}_{c_j}$

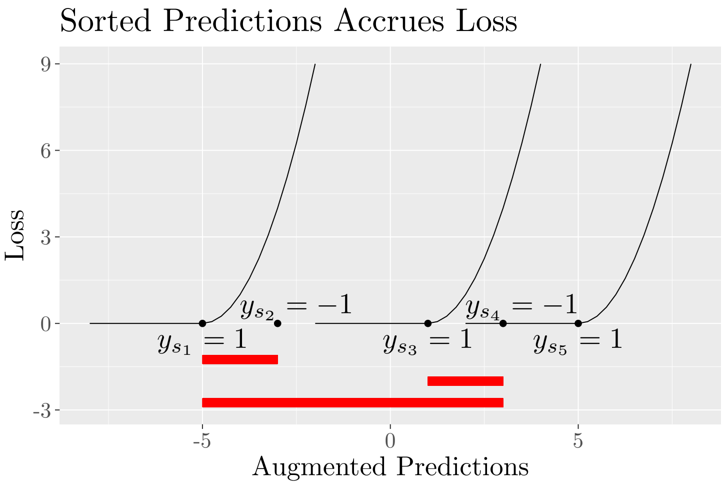
Substitute to coefficients:

5. $= a^+ \hat{y}_k^2 + b^+ \hat{y}_k + c^+$

Geometric View of Square Loss



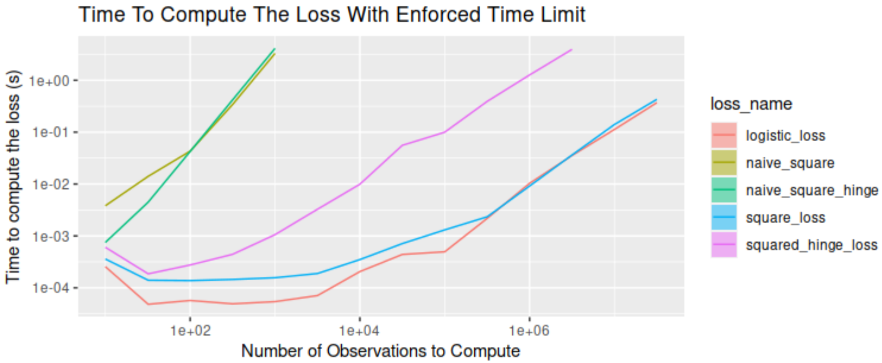
Geometric View of Functional Squared Hinge



Novel Contributions

1. Improved time complexity from $O(n^2)$ to $O(n)/O(n \log n)$
2. Increased possible batch size from $\approx 10^3$ to $\approx 10^7/10^9$
3. Increased model performance for data sets with 0.01 and 0.001 class imbalance

Increased Possible Batch Size



Data Sets

- We trained a model for three data sets
  1. Cat&Dog (Elson et al., 2007)
  2. CIFAR10 (Krizhevsky, 2009)
  3. STL10 (Coates et al., 2011)
- The STL10 and CIFAR10 data sets were converted to binary classification data sets
- The model was trained on each data set for three ratios of positive to negative examples: 0.1, 0.01, 0.001
- To achieve the desired class imbalanced from these sets, positive examples were removed from the train set
- The test set was left balanced between positive and negative examples

Splits

- Each of the following splits was completed 5 times, with 5 different random initializations
- Each data set was first split into 80% and 20%, for training and hold-out test set
- From there the train set was split again 80/20, for gradient descent and hyper-parameter selection

Hyper-Parameters

- The model was tuned for two different hyper-parameters
  1. Batch Size - How many examples are processed in gradient descent
  2. Learning Rate - How big of a step is taken for each batch in gradient descent
- The batch size was selected from 10, 50, 100, 500, 1000, 5000
- The AUCM and logistic losses tested learning rates from $10^{-4}, \ldots, 10^{2}$
- The proposed functional squared hinge loss searched over $10^{-4}, \ldots, 10^{-1}$

Loss Functions

- The proposed functional squared hinge loss was compared to a state-of-the-art and baseline loss functions
- The AUCM loss proposed by Yuan et al. (2020) serves as the state-of-the-art comparison
- The logistic (binary cross entropy) loss serves as the baseline comparison
- For the logistic loss, equal weights were used.

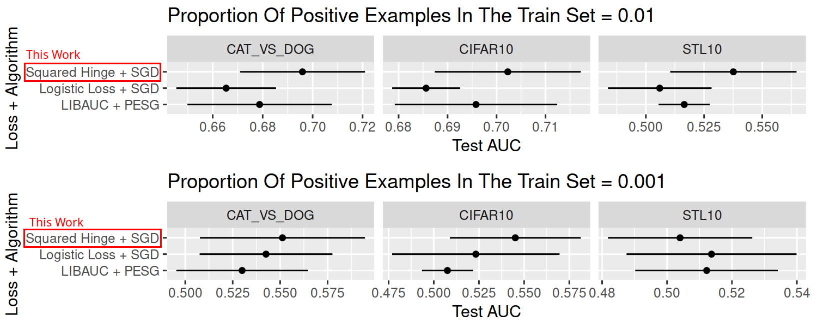
Model Architecture

- The model that was used for training was the ResNet20 architecture (He et al., 2015)
- The model was initially created to solve the CIFAR10 data set
- The 20 layers consist of convolutional layers, activation layers, a softmax layer, a fully connected layer, and a reshape
- This resulted in 270,000 parameters to learn

Increased Possible Batch Size

		CIFAR10		STL10		Cat&Dog	
Imratio	Loss Function	Batch	Learning Rate	Batch	Learning Rate	Batch	Learning Rate
0.1	**This Work**	10	0.0316	10	0.0100	50	0.1000
	LIBAUC	50	0.1000	50	0.1000	50	0.1000
	Logistic Loss	10	0.1000	50	0.1000	50	1.0000
0.01	**This Work**	10	0.0032	100	0.1000	50	0.0316
	LIBAUC	50	0.1000	1000	0.1000	100	0.1000
	Logistic Loss	10	0.1000	1000	0.1000	100	1.0000
0.001	**This Work**	**500**	0.0316	10	0.0001	**1000**	0.3162
	LIBAUC	100	10.0000	10	0.0001	500	10.0000
	Logistic Loss	100	1.0000	100	0.0001	100	1.0000

Increased Model Performance



Conclusion

- In this presentation algorithms for computing the square and squared hinge loss in $O(n)$ and $O(n \log n)$
- Background information and related work were discussed for context around the problem
  1. A preliminary experiment showing the asymptotic time complexity of the proposed algorithms
  2. Demonstrated how it can be advantageous to select larger batch sizes that were not previously feasible
  3. Finally it was shown that these new methods can out perform baseline and state-of-the-art loss functions
- An open source implementation of the proposed square and squared hinge loss can be found *here*

Bibliography I

Bamber, D. (1975). The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *Journal of mathematical psychology*, 12(4):387–415.

Calders, T. and Jaroszewicz, S. (2007). Efficient AUC optimization for classification. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 42–53. Springer.

Coates, A., Ng, A., and Lee, H. (2011). An Analysis of Single Layer Networks in Unsupervised Feature Learning. In *AISTATS*. https://cs.stanford.edu/~acoates/papers/coatesleeng_aistats_2011.pdf.

Cortes, C. and Mohri, M. (2004). AUC optimization vs. error rate minimization. *Advances in neural information processing systems*, 16(16):313–320.

Elson, J., Douceur, J. J., Howell, J., and Saul, J. (2007). Asirra: A captcha that exploits interest-aligned manual image categorization. In *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc.

Ferri, C., Flach, P., and Hernández-Orallo, J. (2002). Learning decision trees using the area under the ROC curve. In *ICML*, volume 2, pages 139–146.

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, 4(Nov):933–969.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Hocking, T. D. (2022). Max generalized auc. https://github.com/tdhock/max-generalized-auc.

HPC, N. H. P. C. (2021). Monsoon details. Northern Arizona University.

Joachims, T. (2005). A support vector method for multivariate performance measures. In *Proceedings of the 22nd international conference on Machine learning*, pages 377–384.

Bibliography II

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. DOI:10.1.1.222.9220.

Pahikkala, T., Tsivtsivadze, E., Airola, A., Järvinen, J., and Boberg, J. (2009). An efficient algorithm for learning to rank from preference graphs - machine learning.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Vannoy, T. C., Belford, J., Rust, K. R., Roddewig, M. R., Churnside, J. H., Shaw, J. A., Whitaker, B. M., et al. (2021). Machine learning-based region of interest detection in airborne lidar fisheries surveys. *Journal of Applied Remote Sensing*, 15(3):038503.

Ying, Y., Wen, L., and Lyu, S. (2016). Stochastic online AUC maximization. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 451–459.

Yuan, Z., Guo, Z., Xu, Y., , and Ying, Yiming Yang, T. (2021). Federated deep auc maximization for heterogeneous datawith a constant communication complexity. Preprint arXiv:2102.04635.

Yuan, Z., Yan, Y., Sonka, M., and Yang, T. (2020). Robust Deep AUC Maximization: A New Surrogate Loss and Empirical Studies on Medical Image Classification. Preprint arXiv:2012.03173.

Zhao, P., Hoi, S. C., Jin, R., and Yang, T. (2011). Online AUC maximization. In *28th international conference on machine learning*.

Cross Validation

- We want to develop a machine learning model that can make predictions on new data
- To do this we have to assume our new data is similar to the data we train on
    1. In statistics this is called independent and identically distributed
- One method for doing so is to use the K-Fold methodology
- To do this the data is first separated in train and test sets
- The train set is then split again into the subtrain and validation set

K-Fold

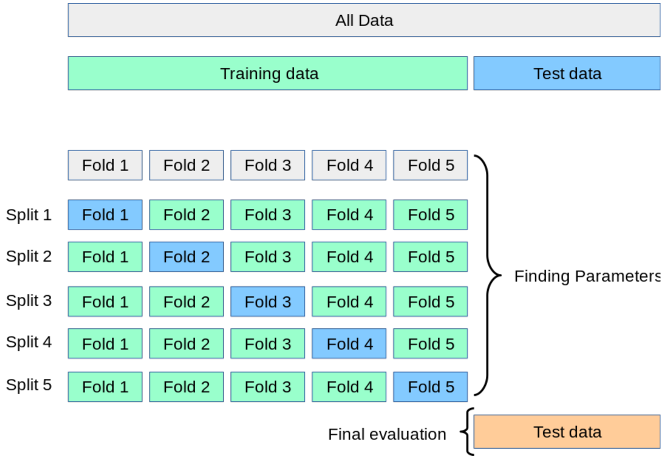
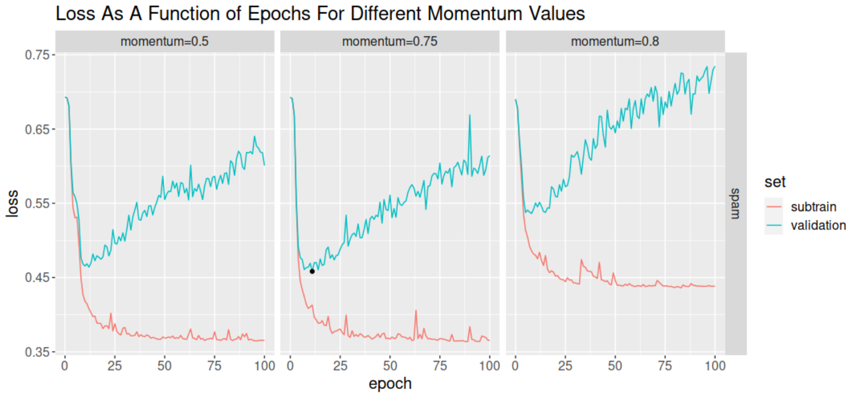


Figure: *(Pedregosa et al., 2011)*

Regularization



Monsoon Cluster

- Timing experiments were carried out on a personal machine with a Intel(R) Core(TM) i5-8600 CPU @ 3.10GHz CPU
- All model training experiments were carried out on NAU's computing cluster Monsoon
- Monsoon consists of 4076 cores, 26TB of memory, and 27 NVIDIA GPUs (HPC, 2021)
- These model performance experiments were computed using AMD EPYC 7542 CPUs

PyTorch

- PyTorch is an open-source machine learning library
- PyTorch provides two main pieces of functionality:
  1. Good performance on GPU's
  2. Automatic gradient calculation
- This is done using a tensor data structure, where derivative information is stored in a graph-like structure
- It can be implemented by simply calling **backward** on the loss value

Last Layer Comparison

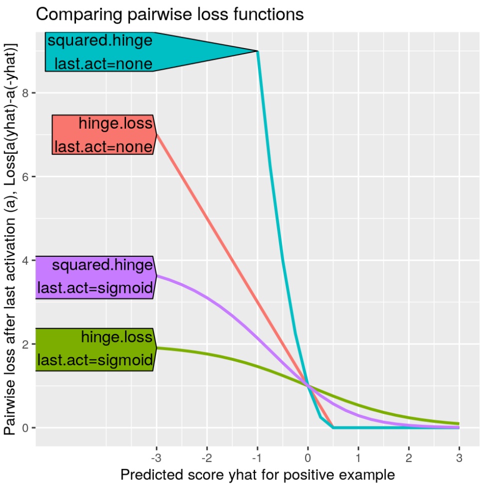
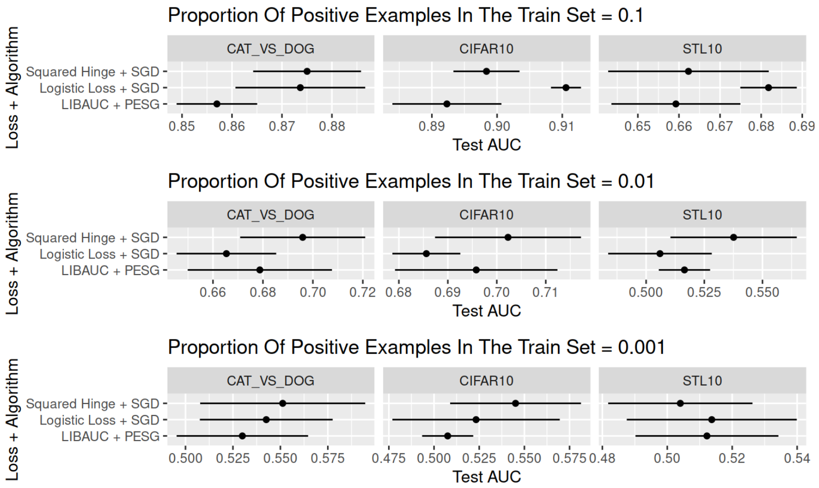


Figure: (Hocking, 2022)

Increased Model Performance Cont.


Proportion Of Positive Examples In The Train Set = 0.1
CAT_VS_DOG
CIFAR10
STL10
Squared Hinge + SGD
Logistic Loss + SGD
LIBAUC + PESG
Test AUC
Loss + Algorithm
Proportion Of Positive Examples In The Train Set = 0.01
Proportion Of Positive Examples In The Train Set = 0.001