Solving Non-Decomposable Objectives Using Linear Programming Layers in General Machine Learning Models - SVMs and Deep Neural Networks

by

Clint Woerishofer

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science

Computer Science

At

The University of Wisconsin–Whitewater

May, 2021

Graduate Studies

The members of the Committee approve the thesis of
(Clinten M. Woerishofer) presented on (May 3rd, 2021)

Dr. Lopamudra Mukherjee, Chair

Dr. Hien Nguyen

Dr. Athula Gunawardena

ACKNOWLEDGMENTS

I am truly grateful for the faculty in the Computer Science department at UW-Whitewater who have provided me knowledge and support along my academic journey. I would like to thank my advisor Dr. Mukherjee for introducing me to machine learning and her guidance throughout this project. I would also like to thank my committee members Dr. Gunawardena and Dr. Nguyen for the feedback that was critical to this project.

Finally, I would like to thank my parents Tim and Yvette Woerishofer for their endless encouragement from a young age to pursue higher education. They have supported me through every step of my life, and I cannot thank them enough.

Solving Non-Decomposable Objectives Using Linear program Layers in General Machine Learning Models - SVMs and Deep Neural Networks

By

Clint Woerishofer

The University of Wisconsin-Whitewater, 2021 Under the Supervision of Dr. Mukherjee

Many domain specific machine learning tasks require more fine tuning with respect to non-decomposable metrics to be effective. In many applications such as medical diagnosis and fraud detection, traditional loss measures do not provide for the best performance. Optimizing for accuracy can give a false sense of effectiveness. In the case of Medical diagnosis, it is typical we want to minimize for false negatives. In the case of fraud detection, the data can often be imbalanced. In the case of imbalanced data, accuracy does not provide the best measure because the model may have high accuracy in the majority class while having low accuracy in the minority class causing the model to appear to have a high effectiveness on the data when in reality the bias of imbalanced data is skewing the results. For such cases, Non-Decomposable measures such a F-Score and AUC provide more detail into the real world performance of the model. Optimizing Non-Decomposable measures has posed a challenge in the past. In this paper, we will investigate using a proposed drag and drop linear programming layer for machine learning methods. We evaluate performance across multiple models and multiple data sets. Our results show increases in Non-Decomposable measures over traditional results and fast convergence.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Many machine learning models suffer from over fitting and high sensitivity to noise when working with large imbalanced data sets [24]. That is, many models will have bias towards the majority class and be unable to effectively classify data in the minority class. In many cases, objectives such as accuracy fall short due to the inability to distinguish between True Positive/Negatives and False Positives/Negatives. A model may have high accuracy in the majority class while having low accuracy in the minority class causing the model to appear to have a high effectiveness on the data when in reality the bias of imbalanced data is skewing the results.

Non-decomposable objective are a driving force behind many machine learning models. Non-decomposable objectives denotes training objectives or loss functions where the learning function when evaluated on a set of examples cannot be decomposed/expressed as the sum of the loss function/objective, applied to each example separately. This is in contrast to decomposable objectives where the loss function can be expressed as a sum over individual examples, such as misclassification error. However, non-decomposable objectives are advantageous because it allows for non-linear trade off between True Positive/Negatives and False

Positives/Negatives, which allows for the representation of wider class of learning losses. Furthermore, several rank based functions such as Area under the curve (AUC), Average Precision (AP) which are commonly used in learning models also fall under the category of non-decomposable objectives as well. To summarize, Non-decomposable objectives are fairly popular in machine learning where imbalanced classification is the norm such as medical diagnosis, fraud detection, and in other applications, where the standard means of measuring learning error (such as misclassification error) falls short. There are a number of recent works which study how to optimize non-decomposable objectives, particularly in context of deep networks. However, such optimizing objectives are in general challenging, since the notion of backpropagation requires the loss function be expressed a function of each input-output pair separately and summed up. This makes optimization of non-decomposable objectives more difficult and consequently deep learning systems are often not trained to optimize the non-decomposable objectives of interest directly. Instead, they are typically trained simply to optimize a surrogate loss or lower bounds of the actual loss function that can be easily optimized. The problem of this procedure is that for many application domains, the chosen loss function differs significantly from the surrogate loss, which is used to replace it. In this paper, our goal is to develop a unified approach that is applicable to a wide range of non-decomposable objectives and that is applicable to large datasets. We propose a re-parameterization such that many non-decomposable objectives can be directly calculated, through the inclusion of linear programming layer, which can be applied to multiple machine learning models.

Chapter 2

Related Work

Non-Decomposable objectives pose their difficulties when trying to optimize for in comparison to traditional loss because they cannot be summed over individual points. Non-Decomposable objectives consider the whole dataset [2, 21]. As such, simply minimizing loss may increase accuracy of the model; however, it will not directly affect measures such as Fscore or AUC. In this section, we will look at current methods for optimizing for such objectives. Neural Networks models learn though the minimization of some loss function. The loss is calculated using the input (target) and the output (predicted) of the model. Backpropigation is employed to minimize the loss. In the simplest form gradient descent is used to take a small step in the direction of the negative gradient [1].

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\left(\mathbf{w}^{(\tau)}\right) \tag{2.1}$$

Where $\nabla E\left(\mathbf{w}^{(\tau)}\right)$ is the gradient of the error of the weight space. And η is the learning rate. Gradient descent methods for optimizing non-decomposable methods have been studied [2]. However, for this method, a surrogate loss is explicit. With our method, an explicit surrogate loss is not needed for the case of neural networks.

Linear programming layers in the context of deep neural networks and machine learning are not new in the literature [11, 22, 23, 24]. [11] gives insight to Domain Specific Languages for convex optimization. The authors show implementation for optimization layers in CVXPY, a popular python based framework. The authors also show their implementation is differentiable when coupled with PyTorch and TensrFlow layers. [23] shows an end to end neural

network framework to treat optimization problems as a layer within a deep neural network. To backpropigate through the layer, the authors provide algorithm by forming Jacobian Matrices.

Physarum [10] is a differntiable LP Solver inspired by the dynamics of slime mold. Physarum is designed to be implemented within Neural Networks. As such, we will use it in our implementation as a solver for our neural network implementation. The authors show positive results in video object segmentation and meta-learning tasks. We seek to extend the use of this into classification problems. Specifically, binary classification to optimize for non-decomposable metrics using a LP layer with the neural network.

[17] provides three basic categories for optimizing support vector machines for non-decomposable objectives. Such categories include, calculating estimates of probabilities of class membership, post-processing tractable performances measures with verification through cross validation, and finally direct optimization of application specific performance. They also propose a SVM framework to optimize Fscore and AUC by formulating a sparse approximation algorithm that iteratively builds constraints. [18] provides a framework for SVM Average Precision optimization by defining AP loss and approximating for AP loss over all samples.

Many of the previous works we looked at in previous works look to optimize for only a small subset of non-decomposable metrics such as F-Score, AUC, Average Precision[12, 13, 14, 17, 18]. In contrast, [9] provides a formulation to optimize for multiple metrics such as AUC, AUCPR, P@R, R@P, and F_β. With this, the authors provide the building blocks for our work. The authors propose a framework built on linear programming to optimize general machine learning models for various rank based objectives including R@P, P@R, AUCPR and F_β. Their results show increases in AUCPR in the CIFAR10 data-set. By using these building blocks and Physarum, we are able to build a drop and drop linear programming layer that can be easily implemented into general machine learning methods.

Chapter 3

Remodeling as LPs

Optimization of non-decomposable objectives has been seen in the literature in the past and is well known for its difficulties to optimize for [2, 21]. To minimize the objective in a neural network, we must compute the gradients for backpropigation. Loss functions such as Cross Entropy loss are used to compute the loss over individual instances. This is in contrast to non-decomposable objectives that define such metrics over the entire dataset. Many objectives include Fscore, F_β Jaccard, Dice, AUC, P@R, R@P and more. In this section, we will introduce the metrics used in this paper and set the building blocks.

3.0.1 Precision & Recall

As independent metric, Precision & Recall are generally not seen as relevant metrics to evaluate the performance of a given model. Precision is an evaluation of the classifications predicted to be positive and what is the proportion with respect to cases that were predicted to be positive. Whereas recall is an evaluation of the classifications predicted to be positive and what is the proportion with respect to cases that are actually positive.

$$Precision = \frac{tp}{tp + fp} \tag{3.1}$$

$$Recall = \frac{tp}{tp + fn} \tag{3.2}$$

Where True Positive = tp, False Positives = fp, True Negatives = tn and False Negatives = fn.

A high precision score for a given class y_i indicates a high percentage of classified instances as belonging to y_i does indeed belong to y_i. However, evaluation of precision does not provide insight to the number of incorrect classifications for y_i. Moreover, a high recall indicates a high percentage of instances belonging to y_i were classified as such. Similarly, this does not provide insight into how many instances from the opposite class were incorrectly labeled (in a binary context).

3.0.2 Fscore

Fscore or F1-measure is used to evaluate the accuracy of a model. However, unlike traditional accuracy, F1 is able to identify if the model has low accuracy with a particular class. A model may be able to have high accuracy because it is able to identify the majority class in many or all cases. However, if it only classifies the minority class few times or not at all, the accuracy can still be high due to the number of examples being much greater in the majority class. If this happens, F1 score is able to identify this imbalance. This allows for a better performance measure than isolated Precision & Recall as Fscore is the harmonic mean of Precision & Recall defined as:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{tp}{tp + \frac{1}{2}(fp + fn)} \tag{3.3}$$

3.0.3 AUC ROC

Area Under Curve or AUC metric is used to define the ability of a model to distinguish classes as the threshold is varied. When AUC is approximately 0.5, the model has no discrimination capacity to distinguish between positive class and negative class. Let $f(x) \in 0, 1$ be a classification model that outputs the prediction of a given feature vector x. Let x^+ define a positive class x^- define a negative class. Given a threshold T, $f(x) > T \implies x^+$ prediction and $f(x) < T \implies x^-$ prediction. Given a positive input x_i $f(x_i) > T$ is a true positive. If x_i is a negative class input $f(x_i) > T$ is a false positive.

Let $P(f(x_i))^+$ be the probability x_i belongs to the positives class and $P(f(x_i))^-$ be the probability x_i belongs to the negative class. We formulate: [4]

$$TruePositiveRate = \int_T^\infty P(f(x_i))^+ dx \tag{3.4}$$

$$FalsePositiveRate = \int_T^\infty P(f(x_i))^- dx \tag{3.5}$$

3.1 Methods

Let n be the number of samples used in training. We use X to denote the explanatory features, which is passed as input to a machine learning that predicts the classification, parametrized by w. Let $Y \in \{0,1\}^n$ to denote the target label, $\hat{Y} \in \{0,1\}^n$ to denote the predicted label (here 0 indicated negative class, whereas 1 indicates positive class), both are in R^n. Since $\hat{Y}$ is the output of the model, we can write $\hat{Y} = \phi(wX)$. With this definitions, we can define True Positive(TP), False Positives(FP), True Negatives(TN) and False Negatives(FN) as follows:

$$\begin{aligned} \mathrm{TP} &= \mathrm{Y}^T \times \hat{Y} & \mathrm{FP} &= (1-\mathrm{Y})^T \times \hat{Y} \\ \mathrm{TN} &= (1-\mathrm{Y})^T \times (1-\hat{Y}) & \mathrm{FN} &= (\mathrm{Y})^T \times (1-\hat{Y}) \end{aligned} \tag{3.6}$$

3.2 Maximizing Fscore

The F-score or F-measure is routinely used as a performance metric for different types of prediction problems, including binary classification and, multi-label classification. Compared to measures like error rate in binary classification and Hamming loss, it enforces a better balance between performance on the minority and the majority classes, and, therefore, it is more suitable in the case of imbalanced data [12]. We construct a variable $\tilde{Y}$ from $\hat{Y}$ such that $\tilde{Y} = \frac{1+\hat{Y}}{2}$. F-score is defined as follows:

$$Fscore(Y, \hat{Y}) = \frac{2P(Y, \hat{Y}) \times R(Y, \hat{Y})}{2P(Y, \hat{Y}) + R(Y, \hat{Y})} \tag{3.7}$$

Where P is the measure of precision defined as

$$P(Y, \hat{Y}) = \frac{TP}{TP + FP} \tag{3.8}$$

and R stands for the measure of recall, given as

$$R(Y, \hat{Y}) = \frac{TP}{TP + FN} \tag{3.9}$$

Plugging this in Eq(1), and replacing the formulations for TP, FP and FN(from Eq 3) we get

$$\text{Fscore}(Y, \hat{Y}) = \frac{2TP}{2TP + FP + FN} = \frac{2\left(\frac{(1+Y)^T}{2} \times \tilde{Y}\right)}{\sum_{i=1}^{n} y_i + \sum_{i=1}^{n} \hat{y}_i} = \frac{2\left(Y^T \times \hat{Y}\right)}{1^T Y + \mathbf{1}^T \hat{Y}} \tag{3.10}$$

where y_i refers to the ith element of Y (same for $\hat{Y}_i$). $\mathbf{1}$ represents an all one vector in R^n. Note that in training, since Y is generally provided, we can assume $\mathbf{1^T}Y = \beta$ which is constant (the number of examples in the positive class in the ground truth) and $\alpha = \beta + b$. We can also represent the the values of $2Y$ as a coefficient matrix c, then the optimization problem for finding F-score can be written as:

$$\begin{aligned} \underset{\hat{Y}}{\text{maximize}} \quad & \frac{c^T \hat{Y}}{\alpha} \\ \text{subject to} \quad & \hat{Y}_i \in [0, 1], i = 1, \ldots, n \end{aligned} \tag{3.11}$$

First, we can relax the constraint on $\hat{Y}_i$ to be in the range of $[0, 1]$. Then the problem can be formulated a a linear fractional objective, by introducing two variables $z = \frac{\beta \hat{Y}}{\alpha}$ and $t = \frac{\beta}{\alpha}$ and $t \in R^1$ and using the relation. The equivalent linear program then can be written as:

$$\begin{aligned} \text{maximize z, t} \quad & \frac{c^T z}{\beta} \\ \text{subject to} \quad & d^T z + \beta t = \beta \\ & z_i \geq 0, t \geq 0, i = 1, \ldots, n. \end{aligned} \tag{3.12}$$

Here, n constraints are needed where n is the number of samples. $d = \mathbf{1}$ is vector in R^n. To recover $\hat{Y}$ from the solution to the linear program, we can compute $\hat{Y} = \frac{z_i}{t}$ and $\hat{Y} = z_i$ otherwise.

3.3 Maximizing AUC

As stated previously, AUC metric is used to define the ability of a model to distinguish classes as the threshold is varied. The output of AUC is a probability with a value between 0 and 1. The higher the output, the higher the probability a positive class is ranged above the negative class. The Wilcoxon-Mann-Whitney(WMW) test is a statistical test to determine the probability a given sample from the positive class is ranked higher than a sample from the negative class [13]. We can estimate AUC using this statistic.

$$A = \frac{\sum_{i=1}^{T} \sum_{j=1}^{T} 1_{f(i)>f(j)}}{|T||N|} \tag{3.13}$$

Here, we assume T is the set of points in the positive class , N is the set of points in the negative class, and $f(x)$ is the classifier function assigning a score to each element $x_i : i \in \{1, 2, ..., n\}$, where $|T| + |N| = n$. With this formulate, use the formulation of [14] to calculate AUC based on the WMW Statistic

$$\begin{aligned} \underset{z_{ij}}{\text{minimize}} \quad & \sum_{i=1}^{T} \sum_{j=1}^{N} z_{ij} \\ \text{subject to} \quad & f(x_i) - f(x_j) \geq \epsilon - z_{ij} \quad x_i \in T, x_j \in N \\ & z_{ij} \geq 0 \quad i = 1, \ldots, T, j = 1 \ldots N \end{aligned} \tag{3.14}$$

Here ϵ is a small user provided constant. The model in (3.14), maximizes AUC indirectly by minimizing the number of pairs (one from the positive class, and one from the negative class) where the positive point is not ranked above the negative point. Note that upon solving this problem using a linear programming layer, we obtain the solution variable z.

Chapter 4

SVM

4.1 Formulation

A Support Vector Machine (SVM) is a supervised learning method for classification. SVMs are often refereed to as large margin classifiers as they attempt to find a hpyerplane that maximizes the distances between examples classes for linearly separable classes. If y belongs to the positive class, $y = 1$ and the SVM will attempt to find a predicted value $\hat{y} \gg 0$. The primal of the linear classfier can be defined as [7]:

$$f(x) = w^T x + b \tag{4.1}$$

and is formulated as solving the optimization problem over w:

$$\min_{\mathbf{w} \in R^d} \|\mathbf{w}\|^2 + C \sum_{i}^{N} \max\left(0, 1 - y_i f\left(\mathbf{x}_i\right)\right) \tag{4.2}$$

The equivalent dual of the classifier can be defined as:

$$\min_{\alpha_j} \sum_{jk} \alpha_j \alpha_k y_j y_k \left(\mathbf{x}_j^\top \mathbf{x}_k\right) \tag{4.3}$$

subject to:

$$y_i \left(\sum_{j=1}^{N} \alpha_j y_j \left(\mathbf{x}_j^\top \mathbf{x}_i\right) + b \right) \geq 1, \forall i \tag{4.4}$$

In order to handle data that is not perfectly linearly separable we introduce:

$$\min_{\mathbf{w} \in R^d, \xi_i \in R^+} \|\mathbf{w}\|^2 + C \sum_{i}^{N} \xi_i \tag{4.5}$$

subject to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \forall i \tag{4.6}$$

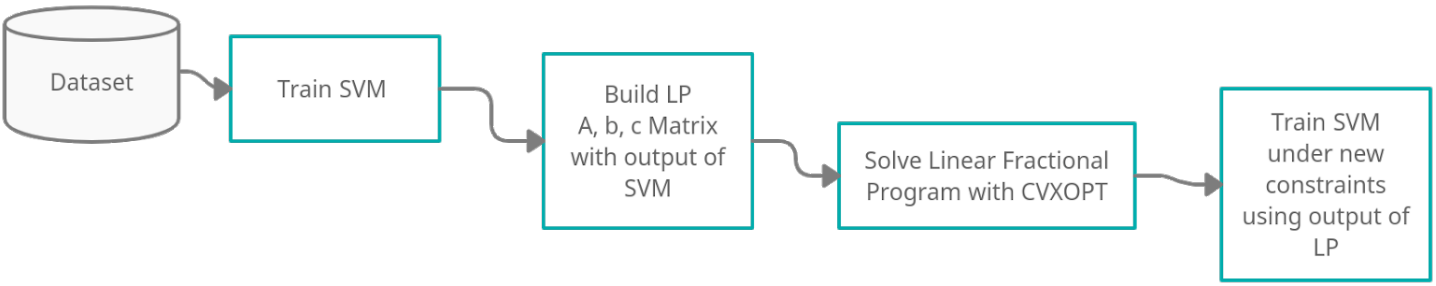

Figure 4.1 Workflow diagram of SVM Implementation

4.1.1 Implementaiton

Using the output of the F-Score optimization LP, we feed θ back into the SVM to influence the decision boundary under the constraint:

$$\theta \left(w^T \mathbf{x}_i + b \right) \geq 1 - \xi_i, \forall i \tag{4.7}$$

Chapter 5

Neural Network

5.1 Formulation

Neural Networks have been widely studied in recent years due to rise in computation power of modern hardware. In general, modern architectures consist of an input layer followed by a hidden layer and finally into an output layer. The input layer of the neural network is the size of the feature vector that denotes all features for the given input. This layer is fed forward into the hidden layer. The hidden layer consists of some activation function such as sigmoid or ReLU. For our implementation, we used ReLU.

$$f(x) = \max\{0, x\} \tag{5.1}$$

A loss function is defined to evaluate the error of the model. The loss is computed as a scalar output and is then used for backpropiagtion to update the weights of the network [15]. In general, the specific loss function is not important as long as the derived can be easily computed. For our implementation, we used Cross-Entropy Loss.

$$L = -\sum y_i log(\hat{y_i}) \tag{5.2}$$

5.1.1 Implementation

In this section, we will discuss how to implement LP into a neural network. To optimize for a given objective, we will use the output of the LP to influence the loss function (L) of the model. In practice, the implementation can be done with only a few lines of code added to the

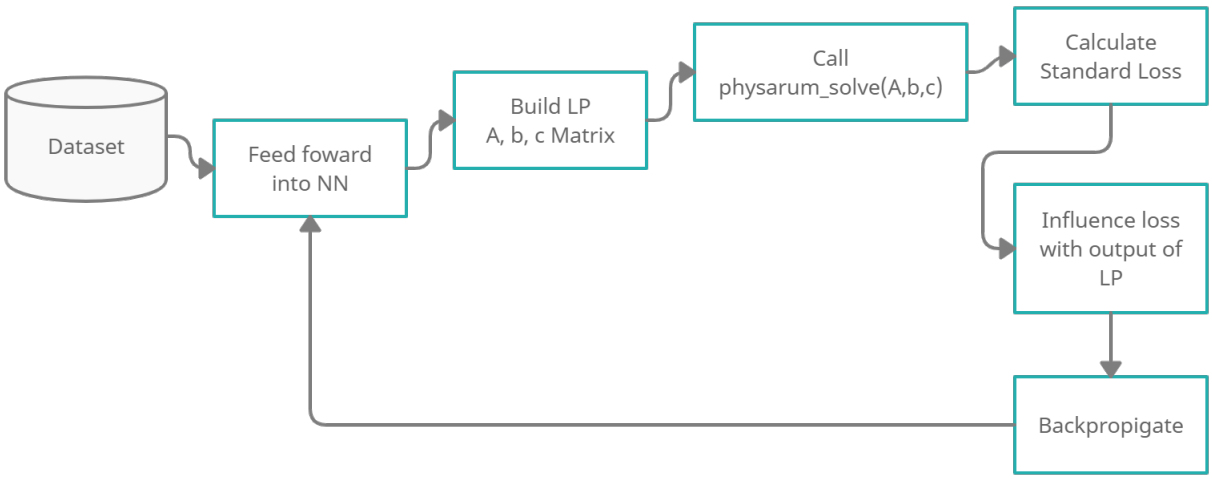

Figure 5.1 Workflow diagram of Neural Network Implementation

training loop. We begin by building the LP w.r.t to the objective we want to optimize for. Let θ be the output of the LP and c be the objective function. Let v be the Hadamard product of $-\theta$ and c. We add the output scalar of the loss function to $\mathbf{1}^{\mathbf{T}}v$.We assign the results to the loss of the neural network and use this for backpropigation. With this, we have influenced the loss function and therefore will influence backpropigation in turn optimizing the decision boundary for our non-decomposable objective.

$$L := L + \mathbf{1}^{\mathbf{T}}v \tag{5.3}$$

Chapter 6

Experiment and Results

For our experiment, we seek to measure the difference in performance metrics of the LP with general machine learning methods against a vanilla implementations. We used real world data sets as well as synthetic data sets to test against a diverse array of scenarios to stress the model. For real world data-sets, we test fraud detection, loan prediction, as well as heart disease prediction. Each scenario poses its own difficulties and differences for optimal performance. We also implement the LP in a CNN to evaluate performance against image classification. We measure Fscore, Accuracy, and AUC of each model against each data-set.

6.1 Datasets

6.1.1 Real-World Datasets

Many real world data sets have an inherent imbalance of data. One such area is in Credit Card Fraud. Only a small percentage of real world transactions are a result of credit card fraud. Furthermore, credit card fraud data sets exhibit this imbalance and pose a challenge to machine learning models. In this paper, we use credit card fraud data set consisting of 28 features. As it is necessary for confidentially, each feature was anonymized and transformed from its original numerical input using PCA. The data set is largely imbalanced with 284,807 transactions with 492 in the class of fraud. Resulting in a 99.83% to 0.17% class imbalance.

We use image classification to evaluate the performance of the CNN implementation introduce diversity into the experiment. The CIFAR10 dataset is a image classification dataset developed by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The dataset consist of 32x32

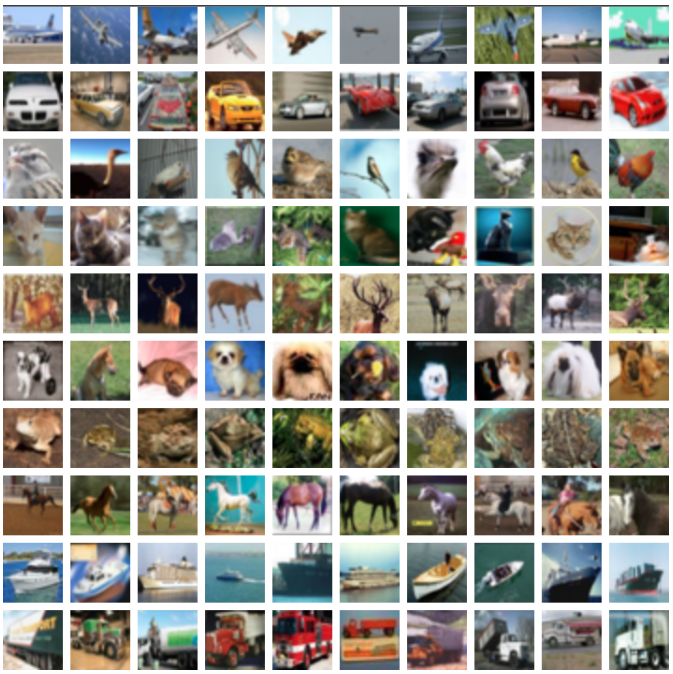

Figure 6.1 CIFAR10 Image Dataset

pixel images. The dataset consists of 60,000 images with 50,000 used for training and 10,000 used for testing. For our experiment, we use one-vs-all classification. The MNIST dataset is a very popular dataset in image classification dating back to the late 1990's and has been used as a general benchmark to evaluate the performance of machine learning models. The dataset consists of 28x28 pixel images of hand written digits. Each class refers to a specific digit.

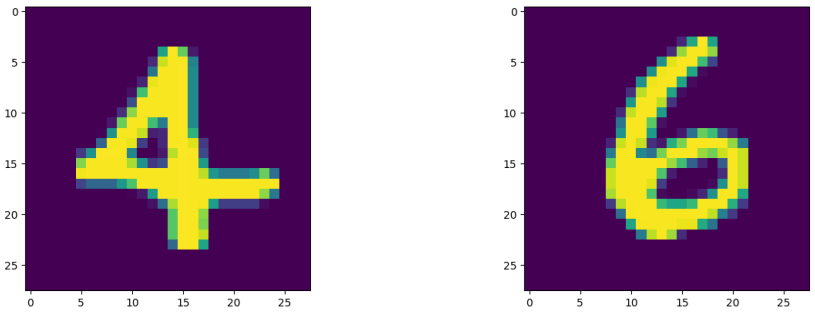

Figure 6.2 MNIST Image Dataset

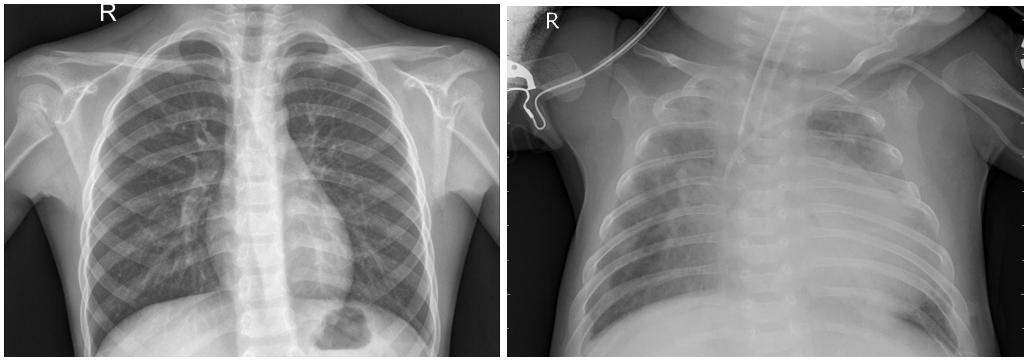

Figure 6.3 Pnuemoinia X-RAY Dataset

6.1.2 Synthetic Datasets

All synthetic generated data sets were made using sklearn.datasets.makeClassification. Each example had multiple data sets generated for each. Results are based on the average across multiple data sets for each parameter to prevent random anomalies and outlier data sets

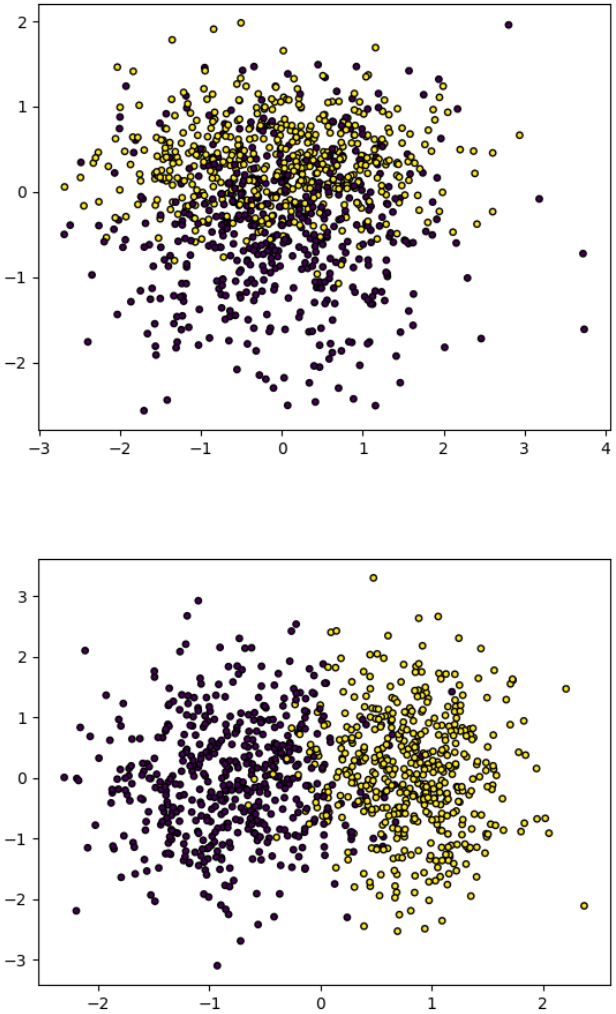

Figure 6.4 Low & High Class Separability (Without Full Separation)

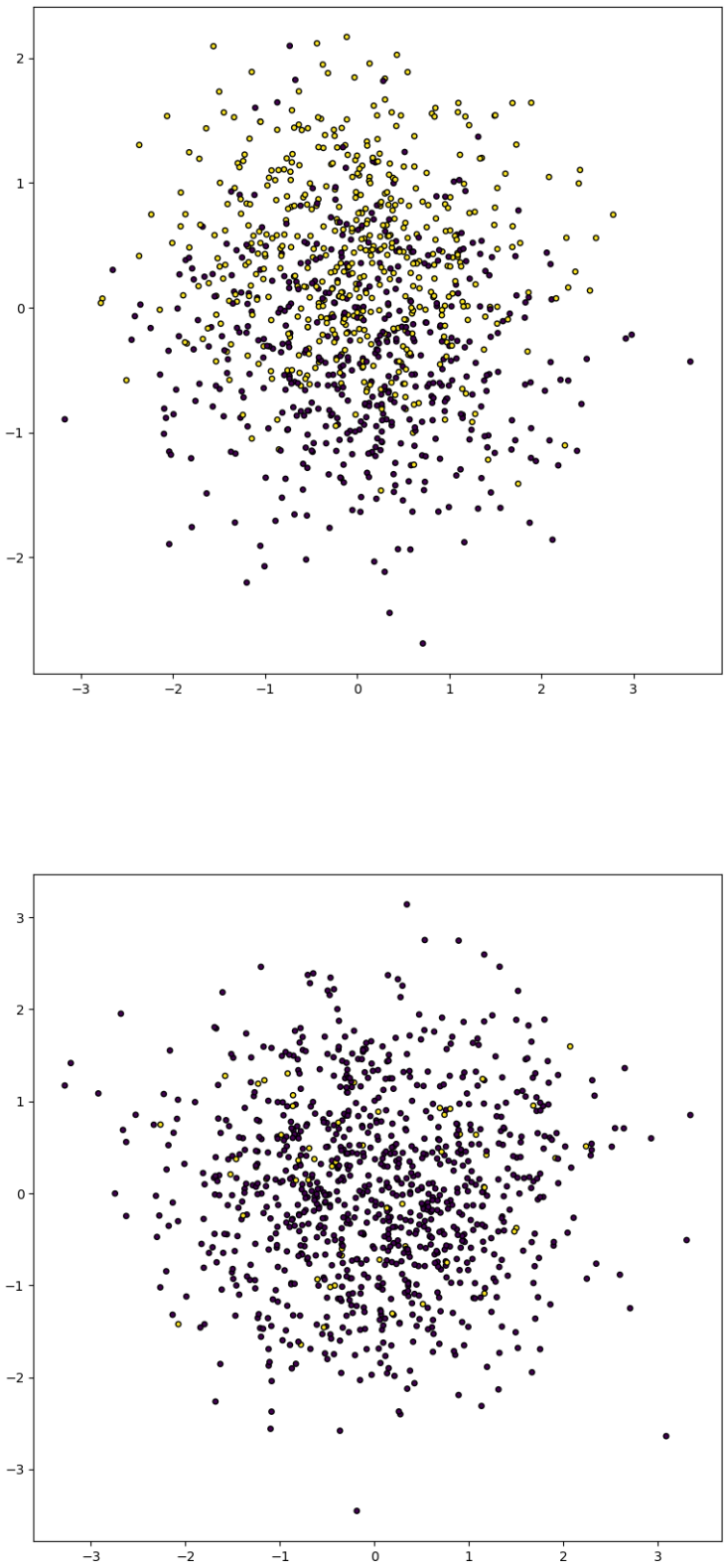

Figure 6.5 Low & High Class Imbalance

6.2 SVM Results

Table 6.1 SVM LP

Dataset	Accuracy	Fscore	AUC
Fraud	99.8	.97	.76
Heart Disease	85	.83	.78
Loan Prediction	94	.93	.71

Table 6.2 SVM

Dataset	Accuracy	Fscore	AUC
Fraud	99.8	.95	.50
Heart Disease	67	.54	.50
Loan Prediction	91	.87	.55

Table 6.3 SVM LP Synthetic Dataset Results

Dataset	Accuracy	Fscore	AUC
Low Class Imbalance	86.0	0.84	0.75
High Class Imbalance	98.0	0.98	0.85
Low Class Separability	82.0	0.82	0.82
High Class Separability	87.0	0.82	0.83
10 Feature Datase	90.0	0.90	0.90
20 Feature Datase	86.0	0.85	0.86
50 Feature Datase	78.0	0.77	0.78

Table 6.4 SVM Synthetic Dataset Results

Dataset	Accuracy	Fscore	AUC
Low Class Imbalance	66.0	0.52	0.5
High Class Imbalance	94.0	0.91	0.5
Low Class Separability	65.0	0.62	0.62
High Class Separability	89.0	0.79	0.73
10 Feature Datase	75.0	0.60	0.61
20 Feature Datase	82.0	0.65	0.60
50 Feature Datase	69.0	0.57	0.59

6.2.1 Neural Network Results

Table 6.5 NN LP

Dataset	Accuracy	Fscore	AUC
Fraud	99.93	0.93	0.98
Heart Disease	72.85	.70	.70
Loan Prediction	97.92	0.94	0.98

Table 6.6 NN

Dataset	Accuracy	Fscore	AUC
Fraud	99.91	.84	.93
Heart Disease	69.2	.63	.65
Loan Prediction	97.48	.90	0.96

Table 6.7 NN LP Synthetic Dataset Results

Dataset	Accuracy	Fscore	AUC
Low Class Imbalance	86.0	.83	.88
High Class Imbalance	97.0	.89	.95
Low Class Separability	79.0	.79	.84
High Class Separability	95.0	.95	.97
10 Features	82.0	.80	.89
20 Features	87.0	.85	.92
50 Features	71.0	.69	.73

Table 6.8 NN Synthetic Dataset Results

Dataset	Accuracy	Fscore	AUC
Low Class Imbalance	85.0	.81	.90
High Class Imbalance	96.0	.78	.70
Low Class Separability	75.0	.74	.84
High Class Separability	94.0	.94	.97
10 Features	74.0	.73	.88
20 Features	84.0	.83	.93
50 Features	67.0	.66	.69

6.2.2 CNN Results

Table 6.9 CNN LP

Dataset	Accuracy	Fscore	AUC
CIFAR10	94.25	.84	.95
MNIST	99.38	.98	.99
X-Ray	80.00	.76	.91

Table 6.10 CNN

Dataset	Accuracy	Fscore	AUC
CIFAR10	92.79	.79	.90
MNIST	98.26	.95	.98
X-Ray	77.24	.71	.88

6.3 Convergence

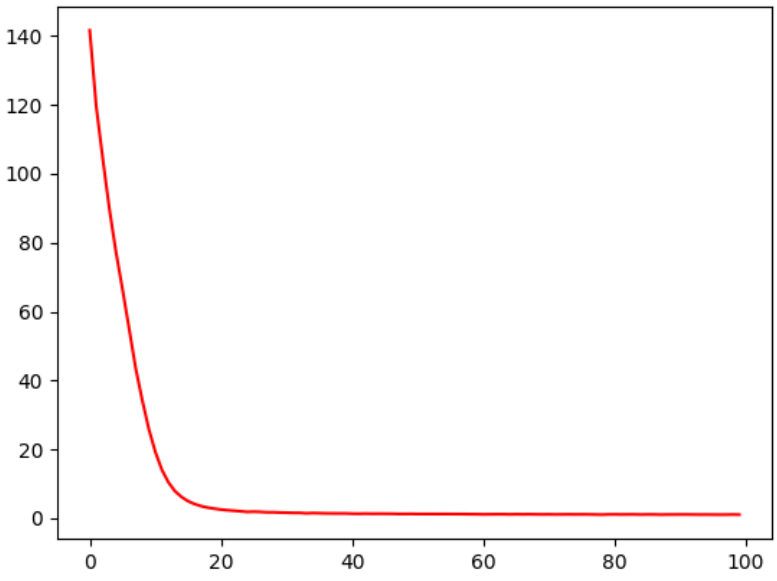

Figure 6.6 Example LP Convergence

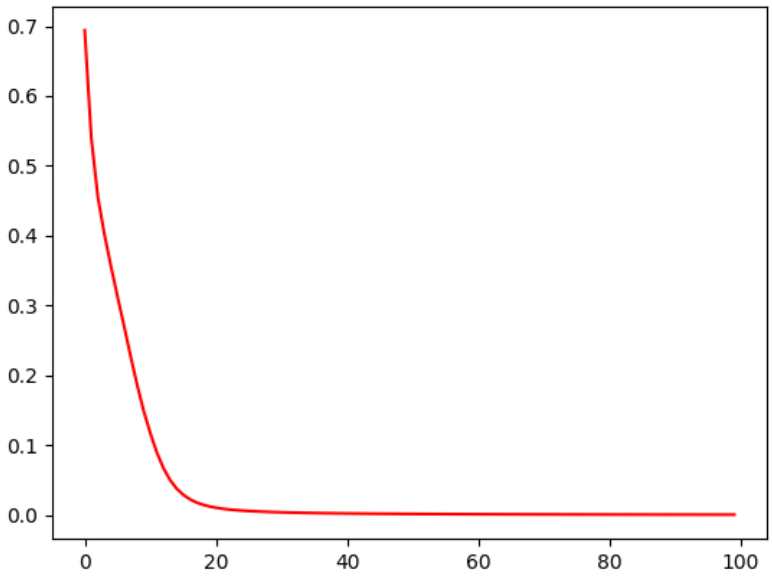

Figure 6.7 Stock Neural Network Convergence

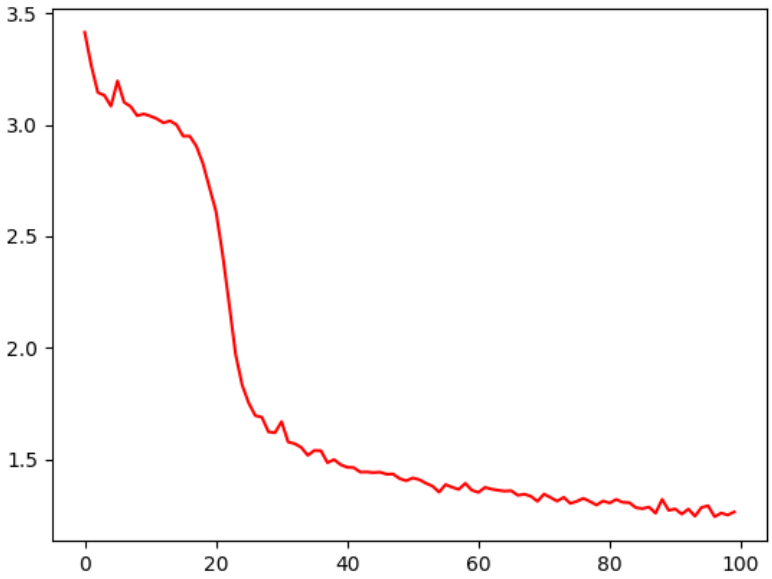

Figure 6.8 Example CNN LP Convergence

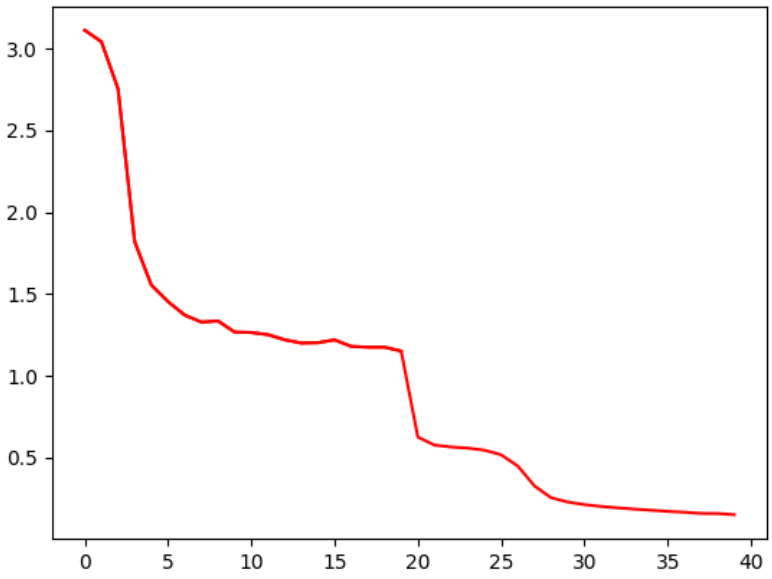

Figure 6.9 Stock CNN Convergence

6.4 Discussion

The results of the SVM show large increases in all three metrics. Most notably, the AUC increases dramatically in all real world and synthetic benchmark scenarios. Fraud detection has better baseline performance than the neural network in Fscore and Accuracy therefore the gains are smaller. For the neural network the increase across the board in Fscore is .04 to .09. These changes significatly increased the performance of the model and allowed it to better distinguish between the class. Accuracy does not change much however, with imbalanced data Fscore is more important because the majority class is skweing the results of the accuracy metric. In this scenario, being able to bettter distinguish between the minority and majority class will give better real world results and allow for better overall performance.

Synthetic Benchmarks in the SVM show for high increase in performance than the neural network implementation. Again, for the SVM we see a high increase in AUC. In some cases, such as class imbalance, the increase is 0.35. Fscore shows similar results across the board but with a less drastic increase. For SVM the most notably difference was in class separability test. Fscore sees and increases by .20. The only drop in performance was the High Class Separability test. With this, we see a small decrease in accuracy for the SVM. However, there is still a gain in Fscore and AUC. The Nerual Network did not see as many gains as the SVM in the synthetic tests. For the easier to classify datastets, such as the High Class Separability test, the performance is the same. All other cases see an increase in metrics. Most notably, High Class Imbalance test saw an increase of Fscore by .11.

Chapter 7

Conclusion

7.1 Conclusion

In this thesis, we implemented a drag and drop linear programming layer for optimizing non-decomposable objectives using a the formulations proposed in [9]. The drag and drop nature of the lp was made possible by Physarum a powerful LP solver proposed in [10]. The implementation was done in pytorch and allows for fast GPU compute. Because of this time to solve the model, it does not significantly increase.

Our SVM results show massive increases across test in AUC. This allows the model to show higher confidence in classification when ranking classes. Each dataset shows increases in Fscore as well as accuracy. Similar can be said for the Neural network implementation. While AUC remains similar throughout due to high AUC from the vanilla implementation. We see solid increases in Fscore throughout. In the imbalanced fraud data-set, we saw a large increase in Fscore raising from .84 to .91. The CNN implementation shows similar results to the Neural Network while the AUC saw a minor increase due to higher performance in the vanilla implementation the increase in Fscore showed similar results.

7.2 Future Works

For future works, the LP can be implemented into other neural network architectures such as Recurrent Networks. Further machine learning methods outside of recurrent neural networks, such as logistic regression can also be implemented. Outside of various learning models

and other non-decomposable objectives such as Jaccard Dice, P@R, R@P can also be implemented. Our pytorch implementation is modular and allows need for only a small change to the buildLP method to be changed for different objectives. Physarum can act as a general solver. With the output of the LP, no changes to the training loop need to be made allowing for easy implementation.

LIST OF REFERENCES

[1] Bishop, Christopher M. (2006). Pattern recognition and machine learning. New York :Springer, page 240

[2] Kar, P., Narasimhan, H., Jain, P. (2014). Online and Stochastic Gradient Methods for Non-decomposable Loss Functions. NIPS.

Krawczyk, B. Learning from imbalanced data: open challenges and future directions. Prog Artif Intell 5, 221–232 (2016). https://doi.org/10.1007/s13748-016-0094-0

[3] Galar, M., Fernández, A., Barrenechea, E., Bustince, H., Herrera, F.: A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. IEEE Trans. Syst. Man Cybern. Part C 42(4), 463–484 (2012)

[4] Fernández, A., López, V., Galar, M., del Jesús, M.J., Herrera, F.: Analysing the classification of imbalanced data-sets with multiple classes: binarization techniques and ad-hoc approaches. Knowl. Based Syst. 42, 97–110 (2013)

[5] Bamber, D., "The area above the ordinal dominance graph and the area below the receiver operating graph". Journal of Mathematical Psychology, 12, 387-415, 1975.

[6] Zhu, Mu (2004). "Recall, Precision and Average Precision" (PDF). Archived from the original (PDF) on 2011-05-04.

[7] Tristan Fletcher, Support Vector Machines Explained, unpublished

[8] Lopamudra Mukherjee, Back to LP: Optimizing Non-decomposable objectives in Deep Networks, unpublished

[9] Elad Eban, Mariano Schain, Alan Mackey, Ariel Gordon, Ryan Rifkin, and Gal Elidan. Scalable learning of non-decomposable objectives. In Artificial Intelligence and Statistics, pages 832–840, 2017

[10] Zihang Meng, Sathya N Ravi, and Vikas Singh. Physarum powered differentiable linear programming layers and applications. arXiv preprint arXiv:2004.14539, 2020.

[11] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. In Advances in neural information processing systems, pages 9562–9574, 2019.

[12] Krzysztof J. Dembczynski, Willem Waegeman, Weiwei Cheng, and Eyke H¨ullermeier. An exact algorithm for f-measure maximization. In J. ShaweTaylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 24, pages 1404– 1412. Curran Associates, Inc., 2011

[13] Culver, Matt Deng, Kun Scott, Stephen. (2006). Active Learning to Maximize Area Under the ROC Curve. 149-158. 10.1109/ICDM.2006.12.

[14] Kaan Ataman, W Nick Street, and Yi Zhang. Learning to rank by maximizing auc with linear programming. In The 2006 IEEE International Joint Conference on Neural Network Proceedings, pages 123–129. IEEE, 2006

[15] Yang Song, Alexander G Schwing, Richard S Zemel, and Raquel Urtasun. Training deep neural networks via direct loss minimization. In Proceedings of The 33rd International Conference on Machine Learning, pages 2169–2177, 2016

[16] Yue, Y., Finley, T., Radlinski, F., Joachims, T. (2007). A support vector method for optimizing average precision. In ACM SIGIR conference on research and development in information retrieval (SIGIR)

[17] Thorsten Joachims. 2005. A support vector method for multivariate performance measures. In proceedings of the 22nd international conference on Machine learning (ICML '05). Association for Computing Machinery, New York, NY, USA, 377–384.

[18] Pritish Mohapatra, C. V. Jawahar, M. Pawan Kumar. Efficient Optimization for Average Precision SVM. NIPS - Advances in Neural Information Processing Systems, 2014, Montreal, Canada. ffhal01069917f

[19] Marzban, Caren. (2004). The ROC Curve and the Area under It as Performance Measures. Weather and Forecasting. 19. 10.1175/825.1.

[20] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. 2006. Adapting ranking SVM to document retrieval. In Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '06). Association for Computing Machinery, New York, NY, USA, 186–193.

[21] M. Ranjbar, T. Lan, Y. Wang, S. N. Robinovitch, Z. Li and G. Mori, "Optimizing Nondecomposable Loss Functions in Structured Prediction," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 4, pp. 911-924, April 2013, doi: 10.1109/TPAMI.2012.168.

[22] Sanyal, A., Kumar, P., Kar, P. et al. Optimizing non-decomposable measures with deep networks. Mach Learn 107, 1597–1620 (2018). https://doi.org/10.1007/s10994-018-5736-y

[23] Amos, B. Kolter, J.Z.. (2017). OptNet: Differentiable Optimization as a Layer in Neural Networks. Proceedings of the 34th International Conference on Machine Learning, in Proceedings of Machine Learning Research 70:136-145 Available from http://proceedings.mlr.press/v70/amos17a.html

[24] Johnson, J.M., Khoshgoftaar, T.M. Survey on deep learning with class imbalance. J Big Data 6, 27 (2019). https://doi.org/10.1186/s40537-019-0192-5