Kennesaw State University

DigitalCommons@Kennesaw State University

Analytics and Data Science Dissertations

Ph.D. in Analytics and Data Science Research Collections

Fall 12-16-2019

Ordinal HyperPlane Loss

Bob Vanderheyden

Follow this and additional works at: https://digitalcommons.kennesaw.edu/dataphd_etd

Part of the Artificial Intelligence and Robotics Commons, Numerical Analysis and Scientific Computing Commons, Other Computer Sciences Commons, Other Statistics and Probability Commons, and the Statistical Models Commons

Recommended Citation

Vanderheyden, Bob, "Ordinal HyperPlane Loss" (2019). *Analytics and Data Science Dissertations*. 4. https://digitalcommons.kennesaw.edu/dataphd_etd/4

This Dissertation is brought to you for free and open access by the Ph.D. in Analytics and Data Science Research Collections at DigitalCommons@Kennesaw State University. It has been accepted for inclusion in Analytics and Data Science Dissertations by an authorized administrator of DigitalCommons@Kennesaw State University. For more information, please contact digitalcommons@kennesaw.edu.

KENNESAW STATE UNIVERSITY

DOCTORAL DISSERTATION

Ordinal Hyperplane Loss

Author: *Supervisor:*

Bob Vanderheyden Dr. Ying Xie

A thesis submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

in the

Institute of Analytics and Data Science

The Graduate College

October 1, 2019

DocuSign Envelope ID: 69431A1F-C2D2-4121-A114-3735908C5998

Kennesaw State University
Graduate College

Thesis/Dissertation Defense Outcome

Name Robert Vanderheyden
KSU ID 000716407
Email rvanderh@students.kennesaw.edu
Phone Number (770) 329-5377
Program PhD in Analytics and Data Science

Title
Ordinal Hyperplane Loss

Thesis/Dissertation Defense: Date 10/1/2019

☑ Passed ☐ Failed ☐ Passed With Revisions (attach revisions)

Signatures

DocuSigned by:
Ying Xie
C9BC0E4...
Thesis/Dissertation Chair
10/3/2019
Date

DocuSigned by:
Herman Ray
Committee Member
10/16/2019
Date

DocuSigned by:
Dr Sherry M
CCCA5314079D476...
Committee Member
10/21/2019
Date

DocuSigned by:
Dr Babak Moazzez
C8CC5E7A623D459...
Committee Member
10/21/2019
Date

DocuSigned by:
Dr Stefanos Manganaris
E5FCF90B5DF146B...
Committee Member
10/21/2019
Date

DocuSigned by:
Dr Sherrill Hayes
4E8AA36590E34AE...
Program Director
10/22/2019
Date

DocuSigned by:
Dr Jennifer Priestley
3274806ABE35425...
Department Chair
10/22/2019
Date

Graduate Dean
10/24/19
Date

Last Modified 05/26/16

Declaration of Authorship

I, Robert Vanderheyden, declare that this thesis titled, “Ordinal Hyperplane Loss” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: 10/24/2019

Abstract

This research presents the development of a new framework for analyzing ordered class data, commonly called "ordinal class" data. The focus of the work is the development of classifiers (predictive models) that predict classes from available data. Ratings scales, medical classification scales, socio-economic scales, meaningful groupings of continuous data, facial emotional intensity and facial age estimation are examples of ordinal data for which data scientists may be asked to develop predictive classifiers. It is possible to treat ordinal classification like any other classification problem that has more than two classes. Specifying a model with this strategy does not fully utilize the ordering information of classes. Alternatively, the researcher may choose to treat the ordered classes as though they are continuous values. This strategy imposes a strong assumption that the real "distance" between two adjacent classes is equal to the distance between two other adjacent classes (e.g., a rating of '0' versus '1,' on an 11-point scale is the same distance as a '9' versus a '10'). For Deep Neural Networks (DNNs), the problem of predicting *k* ordinal classes is typically addressed by performing *k-1* binary classifications. These models may be estimated within a single DNN and require an evaluation strategy to determine the class prediction. Another common option is to treat ordinal classes as continuous values for regression and then adjust the cutoff points that represent class boundaries that differentiate one class from another. This research reviews a novel loss function called Ordinal Hyperplane Loss (OHPL) that is particularly designed for data with ordinal classes. OHPLnet has been demonstrated to be a significant advancement in predicting ordinal classes for industry standard structured datasets. The loss function also enables deep learning techniques to be applied to the ordinal classification problem of unstructured data. By minimizing OHPL, a deep neural network learns to map data

to an optimal space in which the distance between points and their class centroids are minimized while a nontrivial ordering relationship among classes are maintained. The research reported in this document advances OHPL loss, from a minimally viable loss function, to a more complete deep learning methodology. New analysis strategies were developed and tested that improve model performance as well as algorithm consistency in developing classification models. In the applications chapters, a new algorithm variant is introduced that enables OHPLall to be used when large data records cause a severe limitation on batch size when developing a related Deep Neural Network.

Acknowledgements

First and foremost, I want to thank my wonderful spouse, Susan. Through 30+ years of marriage, she’s stood by me through three different attempts to complete my PhD and has always been my biggest supporter. Even when dealing with her own life struggles, she has been a rock upon which I always find support. Thank you honey.

My children, Alex, Andy and Donna have been huge support of my efforts to complete this life goal, along with Alex and Andy’s wonderful spouses, Leslie and Taylor. They have been incredible emotional support. All five will never know how much their encouragement emboldened me to return to schools after 25 years, to attend classes with students who are their ages. In addition, four of them have been terrific editors of my papers as well as people with whom I could vet ideas. Our discussions where we reviewed my research ideas and you shared personal perspective on my work from your understanding of topics like logic gates, recursion, signal processing and how data relates to behavior enhanced my perspective on complex processes that allow me to simplify the concepts, to be able to apply them as needed.

My mother and father, Jutta and Lynn Vanderheyden were the original encouragers in my life. They always believed in my abilities, even when I deeply questioned them. They were my original emotional support, both in pursuing a college and completing advanced degrees. I am forever indebted to them. I love you mom and dad.

I thank my much older sister, Susan, for always being my confidant and encourager. While we had very different academic areas in which we excelled, you set a standard of academic excellence that I had to work hard to attempt to achieve.

Thank you to my brother, Wesley. We spent a lot of years embroiled in typical sibling rivalries, but in the long run, we may have been each-others most important supporters over the course of our lives. You showed me that a with little talent, hard work towards a life goal, would inevitably result in successful attainment of that goal. I love you both dearly.

To my niece Ashley, thank you for the emotional support, but more importantly thank you for the motivation. Returning to school while working full time wasn’t easy, but I simply couldn’t allow myself to fail and permit you to hold your PhD over my head for the rest of my life. I love you always.

I’m not sure that “thank you” is a sufficient sentiment for how Dr. Ying Xie has influenced my life and studies. With so many young, talented students who want to work with you, I find myself constantly wondering why you agreed to let me work with you. From the very first course that you taught, you have been a valuable mentor and teacher. You challenged me to develop ideas and solutions that I would have never thought possible. Thank you for all of the time and energy that you put into my research efforts.

Dr. Jennifer Priestley, you were the person who planted the idea of me returning to academic studies at age 55. With so many young, talented people applying for the program, it would have been easy for you to ignore the crazy old guy. Instead, you saw, in me, the value of having an experienced data scientist, as part of the first cohort. You have been a terrific role model for every student in the program, including me. You are also the reason that this program has attracted so many highly intelligent, motivated young women. It is a true inspiration to see the impact that you have on them.

Thank you to my committee members. Dr. Ray, I can never fully explain to you the impact that you've had on my studies. I never had the pleasure of taking a course that you taught, but the one on one conversations that we've had, in particular our discussions about my specific life circumstances, have been invaluable. Dr. Ni, from the courses that you taught, to the two SAS Shootouts that you lead, to our discussions of draft versions of my dissertation, it is always felt like you treated me more like a colleague than a student, which sent a not so subtle message that I'd chosen well when I decided to enroll in the program. Thank you for your support and the time that you dedicate to your students. Thank you Dr. Moazzez for opening my eyes to an area of mathematical study that I knew existed but did not realize could provide the powerful applications that are possible. You've inspired me to think deeper into the mathematical implications of the work that I'm doing. You have also been a terrific encourager and friend. I think that I may have appreciated our personal conversations even more than our academic conversations. Dr. Stefanos Manganaris, thank you for taking time from your busy work schedule, to participate in this endeavor. Our early conversations related to the application of advanced math concepts to data analysis challenged me and encouraged me to think deeper into the problem that I was attempting to solve.

Thank you, Dr. Sherrill Hayes, for all of the support and encouragement. While your area of expertise may not be data science, the discussions that we've had regarding philosophical questions like "what is research" helped me understand how the technical work that I was doing could be transformed into research.

Thank you to members of Cohort 1: Dr. Jie Hao, Dr. Linh Le, Sergiu Buciumas, Edwin Baidoo and Dr. Bogdan Gadidov. I was very anxious about returning to school after a 25-year absence.

From the first day, all of you treated me like any other student and were a terrific help in getting through several of our challenging courses. The opportunity to work and study with you helped me survive the program. I also want to extend a special thank you to Bogdan. You seemed to always draw the short straw, sticking you with me as a project partner. Thank you for helping me survive those projects.

Speaking of project partners, thank you Jessica Rudd, for being my project partner when Bogdan was not in the class that I was taking. Our conversations outside of course work were special for me. You will always be a dear friend. I also want to thank Yiyun Zhou and Dr. Linh Le for all of the help that you provided. From helping me understand some nuances of Tensorflow and Theano, to helping me work through challenges that I encountered in trying to develop deep learning models. You both have always been willing and capable collaborators and sounding boards.

Thank you to the rest of the students, in other cohorts. This program will quickly be recognized as world class, when you complete your work. Our cohort is the first to finish, but yours may be the best to finish.

Thank you to all of the professors who “only” taught the courses that I took. Dr. Phillipe Laval, Dr. Eric Westlund, Dr. Bo Yang and Dr. Bradley Barney all provided interesting, challenging courses that were very important in laying a solid theoretical foundation for my research.

Possibly most important of all the people associated with this PhD program, thank you to Cara Reeve for working through all of the administrative challenges that we had to deal with. The support team is always important, but when you’re the support team for a brand-new program, the challenges are magnified at least 10-fold. Your help over the years may not have earned awards, but you are the critical life blood for the program.

In closing, I want to thank IBM, for providing an avenue for me to complete this program, while also working full time to support my family. I want to say a special thank you to Kathy McGettrick, Cynthia Wang, Melissa Gray and Mahendran Nagarajan. Without your support, I could not have even considered enrolling.

Table of Contents

Chapter 1. Introduction 1

Chapter 2. Literature Studies 5

Chapter 3. Deep Learning 16

3.1. The Multi-Layer Perceptron 17

3.2. The Exclusive OR problem (XOR) 20

3.3. Deep Neural Networks 24

3.4. Mini-Batch Processing 29

3.5. Hinge Loss 30

Chapter 4. Ordinal Classification Problem Description 32

4.1. Fundamental Ordinal Classification Problem 32

4.2. Geometric Motivation 35

Chapter 5. OHPL – Ordinal Hyperplane Loss 39

5.1. Linear Hyperplanes 41

5.2. Hyperplane Centroid Loss 43

5.3. Hyperplane Point Loss 44

5.4. Deep Learning Strategy Based on OHPL 49

5.5. Scaling to Large Datasets 53

Chapter 6. Experimental Results for OHPLnet 55

6.1. Experimentation: Standard Test Datasets 55

6.2. Algorithm Assessment 56

6.3. Benchmark Algorithms 58

6.4. Benchmark Results 61

6.5. Application To Large Datasets 65

Chapter 7. Evolution of OHPLnet 67

7.1. Mini-Batch OHPLnet 67

7.2. Two-Stage OHPLnet 74

7.3. OHPLall 77

7.4. Experimental Results for New Variants of OHPLnet 79

Chapter 8. OHPLnet Analysis Strategies 82

8.1. Double-Batch Sampling Strategy 82

8.2. Single Stratefied Sampling Strategy 83

8.3. Epoch Stratified Sampling Strategy 85

8.4. Experimental Results for OHPLnet Variants 87

Chapter 9. Application: Classification of Medical Images 97

Chapter 10. Application: Multi-Class Sentiment Analyizer 114

Chapter 11. Application: OHPLnet for Interprettive Assessment 127

Chapter 12. Conclusions 136

Bibliography...138

LIST OF Tables

Table 1 Ordinal Regression Three Class Label Encoding 8
Table 2 XOR Data 21
Table 3 Test Dataset Key Characteristics 56
Table 4 MZE Results for OHPLnet versus Benchmark Algorithms 62
Table 5 MAE Results for OHPLnet versus Benchmark Algorithms 63
Table 6 MAE/MAE Results for OHPLnet versus Benchmark Algorithms 64
Table 7 MZE Results for New OHPLnet versus other OHPL Base Algorithms 80
Table 8 MAE Results for New OHPLall versus other OHPLnet Base Algorithms 81
Table 9 ERA Dataset Double Batch Results for 5 Algorithm Executions 88
Table 10 MZE Results for OHPL/OHPLall versus Analysis Strategies Using OHPL 90
Table 11 MAE Results for OHPLall versus other OHPLnet Base Algorithms 91
Table 12 Standard Deviations of MZE 92
Table 13 Standard Deviations of MAE 92
Table 14 Sample Strategy for Double Batch and Stratified Batches 94
Table 15 BI-RADS Category Scale [53] 97
Table 16 Zero Assessment Patient Key Statistics 104
Table 17 Image Counts by BI-RADS Rating 105
Table 18 OHPLall vs Ordinal Regression MAE and MZE Results 110
Table 19 Rating Level Assessment for a High Performing OHPLall Model 111
Table 20 Detailed Results for a High Performing Ordinal Regression Model 112
Table 21 Results For ‘0’ Rated Cases 113
Table 22 Net Promoter Value to Semantic Label Recode 114
Table 23 NPS Sentiment Analysis Sample Counts by Response Class 119
Table 24 Confusion Matrix: Counts for Actual versus Predicted Classes 120
Table 25 NPS Three Class Counts by Class 121
Table 26 NPS Sentiment Analyzer Results For 20 Iterations of Each Algorithm 124
Table 27 NPS Responses That Are Inconsistent with Verbatim Comments 125
Table 28 Net Promoter Response Distribution 129
Table 29 IT Company NPS Response Counts and Tested Weighting Scales 131
Table 30 Random Assignment of Classes 132
Table 31 NPS Weighted Model Results with Binned Classes 134

LIST OF FIGURES

Figure 1 Local Neighborhood Ordered Classes vs Nominal Classes 15
Figure 2 Simple Multi-Layer Perceptron with weight updates..17
Figure 3 Plot of three common activation functions, found in Deep Neural Networks 18
Figure 4 XOR Plot...21
Figure 5: Fully Annotated XOR Neural Network Graph 22
Figure 6 DNN Representative Graph 24
Figure 7: Residual Neural Network Graph...25
Figure 8 Basic Convolutional Neural Network Graph 26
Figure 9: ResNeXt versus ResNet Architecture Fundamental Differences 27
Figure 10 Recurrent Neural Network Graph 28
Figure 11 Simple Autoencoder 29
Figure 12 Ordered Separation of Classes 34
Figure 13 Separable Mapping $\phi: \mathbb{R}2 \rightarrow \mathbb{R}2$. 36
Figure 14 Parallel Hyperplanes. 38
Figure 15 Hyperplane Point Loss - Increasing Direction. 45
Figure 16 Hyperplane Point Loss - Decreasing Direction. 46
Figure 17 HPL for Three Ordinal Class Case. 47
Figure 26 OHPL as a Deep Neural Network: 50
Figure 27 Convolutional Neural Network Graph with OHPLnet Neural Network Layers 51
Figure 28 Recurrent Neural Network Graph with OHPLnet Neural Network Layers.....................51
Figure 18 Color Coded Confusion Matrix. 58
Figure 19 Time to Complete 500 Epochs by Number of Records (K records)...............................66
Figure 20 SWD Training Dataset: MZE and MAE vs Total Training Error 71
Figure 21 SWD Validation Dataset MZE and MAE vs Total Training Error 72
Figure 22 SWD Validation Dataset MZE and MAE vs Training Dataset MAE 73
Figure 23 SWD Validation Dataset MZE and MAE vs Training Dataset MZE 73
Figure 24 Validation Set MZE and MAE versus Training Set MAE 75
Figure 25 SWD Dataset Validation MAE, MZE and MAE + MZE vs Training set MAE + MZE. 95
Figure 29 Distribution of Row Pixel Count for Cropped Calcification Images 101
Figure 30 Distribution of Column Pixel Count for Cropped Calcification Images 101
Figure 31 Distribution of Row Pixel Count for ROI Calcification Images 101
Figure 32 Distribution of Column Pixel Count for ROI Calcification Images 101
Figure 33 Distribution of row pixel count for ROI Calcification Images with row pixel count between 700 and 1,100. 102
Figure 34 Distribution of column pixel count for ROI Calcification Images with column pixel count between 700 and 1,100...102
Figure 35 Distribution of column pixel count for ROI Calcification Images 102
Figure 36 Sample Mammography Image..103
Figure 37 Sample Mammography Image..103
Figure 38 Sample Mammography Image..103
Figure 39 Sample Mammography Image..103

Figure 40 Convolutional Neural Network Architecture 109
Figure 41 Training Data Mean Batch Error 110
Figure 42 NPS GRNN Network Graph 123

Chapter 1. INTRODUCTION

The problem of ordinal class data occurs in a large and growing number of areas. Some of the most common sources and applications of ordinal data are:

- Ratings scales (e.g. Likert scales, star ratings), like customer satisfaction ratings, “promoter” ratings and quality ratings
- Sentiment scales (negative, neutral and positive)
- Medical classification scales of disease stage/severity/risk (mammogram image BI-RADS)
- Student performance (e.g., letter grades)
- Socio-Economic scale (e.g., high, medium and low)
- Meaningful groupings of continuous data (e.g., generational age groupings, grouping of noisy sensor data)
- Facial emotional intensity [1]
- Facial age estimation [2]
- Weather (e.g., storm severity classes)
- Performance ratings (e.g., high school prospects in football and basketball)

Historically, due to the high cost of data capture for sources like surveys, medical studies, etc., the vast majority of sources for ordinal data generated relatively small datasets (e.g., under 20K records of structured data or a hundred or less for unstructured data like medical images). In more recent years, there’s been a dramatic increase in the number of datasets and analysis problems, with ordinal classes as the primary output/focus, that have hundreds of thousands or even millions of records are being analyzed. In addition, relatively large image and datasets, with ordinal labels are becoming common place. Many of these large data sets have their genesis in the explosion of use of digital and text data. Ratings surveys found on sites like Amazon and Yelp, large corporation Customer Satisfaction/Net Promoter surveys and the aggregation of medical history and/or imaging records into large data systems are primary examples.

Ordinal classes differ from nominal (unordered) classes by providing additional information/requirements in the form of a precise ordering of the classes. As a direct consequence, strategies for predicting nominal classes, tend to under-perform when applied to ordinal data. The use of sequential integers to represent the ordered classes is natural and commonly used for labeling the ordered classes. This representation might suggest that the application of methodologies like regression, that attempt to predict a continuous value would be effective in developing ordinal classifiers. Strategies like regression assume that equal "distances" between values have a consistent numerical meaning (e.g., all one-unit differences having the same meaning), but this assumption is rarely true in ordinal data. Within prediction algorithms, these fundamental differences in the type of data being predicted may be addressed in the loss function or by employing potentially complex, multi-model strategies. An ideal loss metric for ordinal classification would assess the ordering of the data and form discrete homogeneous class groupings without imposing an equal "distance" assumption between predicted classes.

The fundamental difference in ordinal and nominal classes also leads to a difference in assessment for classifier performance. The best classification strategies must not only have a classification accuracy that is on par with or better than other strategies, but in the best strategies misclassified cases should to be ‘close’ to the correct class (e.g., misclassifying a ‘3’ as a value of ‘4’ is more desirable than misclassifying it as a ‘5’).

Existing strategies to address the unique requirements of classifying ordinal data utilize the power of methodologies like SVM (Support Vector Machines) [3, 4, 5, 6, 7] and Gaussian Processes [8]. Others that use Deep Neural Networks employ complex multi-model or repeated-sampling approaches. As such, any attempts to apply them to the large datasets would require major alterations to the algorithm or the use of complex sampling or ensemble strategies that are applied to nonlinear model results.

To address these conditions unique to the ordinal classification problem (also known as the ordinal “regression”), the Ordinal Hyperplane Loss (OHPL) was developed by addressing the following algorithm goals for a current ordinal labelled class of the training data:

1) develop a Neural Network to define a nonlinear mapping of the data into a vector valued output space
2) train the network to establish and maintain the ordering of the classes
3) “drawing” like labelled samples closer together

This formulation maintains the ordering without imposing assumptions regarding the distance between different classes (e.g., as would be imposed by using ordinary least-squares regression analysis). At the same time, the algorithms that are developed from this approach can be applied to very large classification problems.

The remainder of the thesis is organized as follows. Chapter 2 reviews relevant work and existing algorithms, that attempt to solve the ordinal classification problem. Chapter 3 provides a review of Deep Learning including variants of Artificial Neural Networks and related considerations in using Deep Learning algorithms to solve classification problems such as Ordinal Classification. Chapters 4 and 5 cover the geometric and mathematical framework for the development of Ordinal Hyperplane Loss (OHPL). Chapter 6 documents experimental results for the original OHPL work. OHPLall is the culmination of work that's focused on improving upon the original OHPL methodology for application to very large datasets. These advances are reported in Chapter 7. A successive series of algorithm strategies that were designed and tested to improve algorithm performance both in terms of speed and accuracy of predictions are reported in Chapter 8. Chapters , while Chapters 8 through 10 review three different applications of OHPL and OHPLall. Chapter 11 contains conclusions from this work.

Chapter 2. LITERATURE STUDIES

In January 2016, Gutierrez, et. al. published an extensive examination of solutions to the Ordinal Classification problem [9], including benchmark performance metrics versus a set of standard datasets that were included in the work of Chu and Ghahramani [8]. In their review Gutierrez et. al. grouped the existing top performing methodologies into three categories that address the Ordinal Classification problem: 1) Naïve Approaches, 2) Ordinal Binary Decompositions and 3) Threshold Models. While their work attempts to provide a framework for three distinct classes of models, the team acknowledges that many of the most common approaches could be classified into more than one category. Unless specifically attributed to a different researcher, the content of the remainder of this section is attributed to the work of Gutierrez, et. al. [9].

Naïve approaches use an appropriate simplifying assumption to re-cast the problem in such a manner that existing methodologies can be applied. If the researcher assumes that the difference in classes is “close” to uniform they may transform the classes into sequential integers and apply regression analysis like ordinary least squares, neural networks or SVR (Support Vector Regression). Cost sensitive methodologies which use different weights for different misclassification types also fall into this category. Another common naïve approach ignores the class ordering by applying nominal classification approaches like SoftMax regression or multi-class SVM, to predict class membership.

Cost sensitive classification is a more advanced naïve approach. In this approach, misclassification costs will differ between two or more classes, with a goal of maximizing accurate classification of the most desired class. Support Vector Machines with Ordered Partitions (SVMOP) is a high performing algorithm that falls in this category [10]. The algorithm uses class differences, as weights, in an effort to not only provide correct classification, but to encourage misclassifications that are close in class number to the actual class (e.g., for an actual class value of ‘2’, the algorithm encourages a miss of ‘3’, instead of a ‘5’).

The fundamental basis of binary decomposition is to recast the problem as a set of binary classification problems. The problem may be posed by comparing pairs of ordinal values with the higher value being assigned a value of 1 and then using either a single or multiple binary classification models. In the case of multiple classifiers, the analyst may produce as few as *k-1* classifiers for k ordered classes or as many as $\binom{k}{2} = \frac{1}{2}k * (k-1)$ classifiers (i.e., all ordered pairs). An appropriate decision rule is then applied to the set of classifiers. In the *k-1* case, each adjacent ordered pair is analyzed as a binary problem. One popular process examines the highest value in a sequence of values that meet a minimum model value threshold. For example, assume that we have five ordinal classes: ‘1’, ‘2’, ‘3’, ‘4’ and ‘5’. If the first three classifiers (‘1’ vs ‘2’, ‘2’ vs ‘3’ and ‘3’ vs ‘4’) estimate values of 0.5 or higher, but the fourth (‘4’ vs ‘5’) does not, it results in a classification of the highest of the first three classifiers (or ‘4’, in this example). If the first binary classifier value does not meet the threshold of 0.5, then the record is classified as the lowest ordinal value (’1’ in the example). Similarly, the analyst may choose to group classes based on classes (e.g., ‘1’ vs ‘2’-‘5’, ‘1’ & ‘2’ vs ‘3’–‘5’, ‘1’-‘3’ vs ‘4’ & ‘5’ and ‘1’-‘4’ vs ‘5’).

The earliest ordinal binary decomposition approaches used Ordinal Logistic Regression [11], employs logistic regression to estimate the binary probabilities for class ordering (e.g., probability that the label for a given record is ‘3’ or higher). More recent binary decomposition strategies use machine learning approaches like Support Vector Machine (SVM) algorithms to create individual binary classifiers, combined with classification strategy using the binary classifiers. Deep Neural Networks allow of the output of multiple estimates. These estimates may be used to create class probabilities for all classes in a single model. Some approaches endeavored to use non-parallel hyperplanes, in an SVM framework, but at a high cost of increased model complexity. Note that SVMOP would fit into the binary decomposition category but is more appropriately classified as a cost sensitive methodology, within naïve approaches.

A new variant of Ordinal Regression was proposed by Cheng et. al., in 2007. In this approach, a single Deep Neural Network is used to predict the classes. Their approach is very similar to a multilabel classification problem using a DNN, where multiple outputs are estimated with all elements of the output layer being the value from a sigmoid function [12]. To set up the analysis for *k* ordinal classes, the label value for each record is recoded into a *k-1* length vector. For a given class value, ‘a,’ all index values of the vector with position value (using the standard 0 index value for the 1st position in the vector) that are less than ‘a’ minus the minimum ordinal value are coded with a 1. All other values are coded with a zero [12].

The three ordinal class case, with ordinal values ‘1’, ‘2’ and ‘3’, is illustrated in Table 1. For the three-class problem, the neural network essentially estimates two binary models. The first output predicts the likelihood that the label is greater than ‘1’, and the second one predicts the likelihood that the label is greater than ‘2.’ Once the algorithm converges or reaches a predefined stopping point, a classification rule, typically whether or not the value is greater than 0.5, converts each output vector into a binary array that is similar to the one used for training. Ordinal classes are assigned based on which encoded vector matches the binary output. If the first position is zero, then the record is assigned the value of the minimum label [12].

Table 1 Ordinal Regression Three Class Label Encoding

Label	Vector
1	[0, 0]
2	[1, 0]
3	[1, 1]

It should be noted that, while the vast majority of class predictions will conform to one of the vector values of the encoded ordinal classes, it is possible for vector values that do not conform to exist. In the three-class problem, it is possible to have a prediction of '[0, 1]' from applying the resulting model to a data record (either in the training set, a test or validation set or to completely new data). It is left to the analyst to determine how to classify these nonconforming results.

Threshold models are comprised of a large number of methodologies including:

1. Cumulative Link Models: Traced to the Proportional Odds Models that were originally created in the 1980s. Cumulative Link Models map the input data into a one dimensional (i.e., a number line). This number line is appropriately partitioned, to provide class predictions.
2. Support Vector Machines: In 1999, Herbrich et. al. developed single model SVM approach that transformed the input data by calculating the difference between pairs and used the sign of the ordinal class differences. Other applications involve pointwise approaches that

produce *k-1* hyperplanes, to classify *k* ordinal classes. Given the simple ordering information that is available, with ordinal data, the problem lends itself well to algorithms that uses distance learning principles. In 2005, Chu & Keerthi developed two SVM algorithms that specifically address the ordinal classification problem through the estimated multiple hyperplanes that maintain the sequential ordering of the classes [5]. While successful in application to small datasets, their algorithm converts the original SVM proposed by Vapnik et. al., that has a unique individual constraint, for every record, in the dataset, into an optimization problem that has *(k-1)*n* constraints. Keerthi et. al.'s more effective algorithm, which they call 'IMC,' has a problem size of *(k-1)*n*, while the 'EXC' variant scales to a problem size of *2n+k* [5]. The most efficient SVM algorithms have a computational cost of $\mathcal{O}(n^2)$. This computational cost tends to make SVMs impractical with large datasets. Scaling the problem size by a factor of two would quadruple compute cost. For problems with 10 or more classes, the cost for IMC would increase by a factor of 100 or more.

3. Discriminant Learning: The models maximize between class differences and minimize within class differences using the variance-covariance matrix and the Rayleigh coefficient. To adapt discriminant analysis to the ordinal classification problem, an ordering constraint is applied over the contiguous classes. SVM falls under a broader context of kernel methods. Cardoso et. al., in 2012, developed a set of three Kernel Discriminant Analysis (KDA) base ordinal classifiers [13]. One of the classifiers extends the work Frank and Hall, in 2001, which employs a series of binary classifiers [14]. The second uses the data replication strategy of Pinto da Costa, et. al., in 2005 [15]. The third strategy involves the

development of a modified Kernel Discriminant Analysis which applies an ordering constraint on the projected means.

4. Augmented Binary Classification: The general framework includes the development of multiple samples from the original sample, including a weighting of the samples. A binary classifier is then developed using the full set of multiple samples (any binary classification algorithm can be used). Lastly, a ranking process is constructed using the output of the binary classifiers. Pinto da Costa et. al. developed a data replication strategy to design an ordinal classifier that utilizes Deep Neural Networks (DNNs) [15]. In their work they utilized an additional data dimension that represented the sample orderings (e.g., '0' vs '1' and higher has a value of 0, in the additional dimension while '0' & '1' vs '2' and higher had a value of 1). In work that was published in published in 2010, the researchers successfully extended their work data replication strategy, into SVM applications [16]. One of the most common of the distance learning methodologies is Support Vector Machines, which seek to identify hyperplanes that separates classes, in a higher dimensional space. As such they are a natural machine learning methodology to apply to ordinal classification problems.
5. Ensemble Models: The RankBoost algorithm attempts to improve a set of confidence functions, that maximize an ensemble of binary classifiers. Similarly, the ORBoost algorithm applies the same concepts to develop improved performance from ordinal regression models. The basic framework for the creation of ensemble models is the development of "weak" classifiers, that are combined to produce an algorithm that outperforms each of its components. Ensemble methods have a documented history of outperforming competing single model solutions. The weak classifiers may be generated

by using a subset of available features, a subset of records (usually bootstrap sampling) or some combination of the two. Instead of determining an optimal combination from a full set of weak classifiers, boosting algorithms begin with an initial classifier, then add additional weak classifiers until incremental classifier improvement (e.g., improvement in model accuracy, on the training set), becomes zero (or approaches zero).

6. Gaussian Process: GPOR uses a Bayesian framework to model a latent function via Gaussian Processes. Prior and posterior probabilities for class membership are estimated for a set of latent functions of the input features. Optimization with respect to the hyperparameters results in probability estimates of class membership, based on the input record. GPORs include an optimization algorithm that discovers the ideal thresholds for classifying data records based on the output metric, from the gaussian process. GPOR is an example of an analytic framework that could fit into multiple categories.

In late 2016, Hamsici and Martinez proposed a Support Vector Machine based algorithm that attempted to maximize the margins between adjacent classes [17]. The authors apply Sequential Minimal Optimization (SMO), to efficiently and simultaneously solve *k-1* problems, where *k* is the number of ordinal classes. Their algorithm is similar to that Keerthi and Chu, but with the notable and meaningful difference that their algorithm does not assume equal margins between adjacent classes. In addition, their algorithm includes weight parameters, that enable the prioritization of one or more of the individual algorithms, over others. This prioritization weighting allows a researcher to focus on a specific pair of ordered classes (e.g., a medical researcher may want the classifier to have the best possible classification of stage two cancer versus stage three, while still effectively classifying five different ordinal classes). Weighting can also be used to address unbalanced classes within the data (i.e., unbalance records counts for the classes).

In 2017, Wang, et. al. used a nonparallel hyperplane assumption for the development of a specialized Support Vector Machine (SVM) algorithm to address the Ordinal classification problem [7]. For *k* ordinal classes, their algorithm estimates *k-1* hyperplanes. For each, they include constraints which ensure that like-labelled samples are within a prescribed margin of the hyperplane, while unlike-labelled samples are one or more units away. They also include constraints to ensure the ordering of the hyperplanes reflect the ordering of the classes. The use of nonparallel hyperplanes may result in classification issues, if data points map into a region near the crossing of two hyperplanes.

These algorithms exhibit mixed performance across the standard test data sets that are used to benchmark performance of ordinal classifiers. Many are benchmarked using 20 or more small datasets, with performance that represents modest improvements, when the algorithm actually outperforms other classifiers. While these incremental improvements are notable, they are being benchmarked against current “best in breed” classifiers, so as a rule, it is rare to find one that outperforms best benchmark classifier by 10% or more in terms of decline in classification error. It is worthy of note, because the solution that is reported in Chapter 5 has an accuracy improvement of fourteen percent or more on two out of seven benchmark datasets, when compared to four of the highest performing algorithms.

In February 2018, Nguyen et. al. incorporated “Triplet Loss” based constraints to an algorithm that is similar to SVM optimization [3]. Their algorithm employs triplet loss-based constraints, on local clusters of data points. The researchers produced a linear version of their algorithm, as well as a version that employs the kernel trick to produce a nonlinear mapping of the data into a higher dimensional space. Within their work, the algorithm produced solid results with mixed performance where the linear version outperformed the nonlinear version roughly half of the time. Given the researcher's stated algorithm compute cost of $\mathcal{O}(n^3)$, while their solution is successful with relatively small datasets (e.g., under 25,000 records), it may not be viable for larger datasets. In the future, they could conceivably develop a new version that uses SMO to solve the problem, once the constraints are developed. Doing so should broaden the applicability to larger datasets, but still may not be viable if the number of records exceeds 100,000 by a significant amount.

Triplet Loss is a term that was first used in the ground-breaking FaceNet solution to the ReId (reidentification) problem [18]. In developing FaceNet, Schroff et. al. leveraged the foundational work in Large Margin Nearest Neighbor (LMNN) Classification published by Weinberger and Saul [19]. The essence of the FaceNet process is to train a Convolutional Neural Network (CNN) to produce an *N*-dimension embedding, that is optimized based on relative distances of similar and dissimilar pairs of data points. A margin, that is analogous to the margin found in a Support Vector Machine, is used to ensure that similar pairs (those with the same label) being "closer" than dissimilar pairs (those with different labels) is based on a difference in distances that is not trivial (i.e., not arbitrarily close to zero). This process produces what is commonly called a "triplet loss" function (discussed further in Chapter 4) that is based on linear distance comparisons. As the following general triplet loss function demonstrates, the loss function uses a fixed margin that is strictly greater than zero, to ensure that a point, $\boldsymbol{x}_i$, is closer to the positive anchor, $\boldsymbol{x}_p$ (same class), than it is to the negative anchor, $\boldsymbol{x}_n$ (different class from $\boldsymbol{x}_i$) and the difference, δ, is fixed and not trivially close to zero [18].

$$triplet\ loss = \max\left(d\left(x_p, x_i\right) - d(x_n, x_i) + \delta, 0\right) \quad (1)$$

Triplet loss puts a significant burden on the analyst to devise a reasonable strategy for identifying triplets for use in estimating function error, since the number of possible triplets grows as a cubic function of dataset size [20]. The framework of triplet loss provides a mechanism for applying a distance comparison between points without requiring the underlying distance assumptions of regression analysis. This framework of triplet loss makes it well suited to the ordinal classification problem, but triplet loss cannot be used because it does not guarantee that the ordering information is utilized nor that the ordering of classes is guaranteed (it only guarantees that different classes are separated). While it cannot be directly applied, triplet loss provides some of the intuitive motivation for methodology reported in Chapter 4.

Triplet loss effectively addresses the ordering, but only as it relates to an identified triplet of data points. In developing LODML, Nguyen et. al. developed a useful geometric representation of the goal of their use of triplet-based constraints. In a two-dimensional representation of a neighborhood, they illustrate the goal of classes falling in 'distance band' radiating out from the center of the neighborhood with the center being a chosen data point (Figure 1, below) [3]. In comparing a nominal problem, to an ordinal problem their graphic illustrates that the 'distance' frame of reference must be rotated, to ensure that the ordering of classes to be properly maintained. Without loss of generality, these distances could easily be mapped to a continuous scalar scale. In doing so, the ordered classes would occur in clusters along the number line.

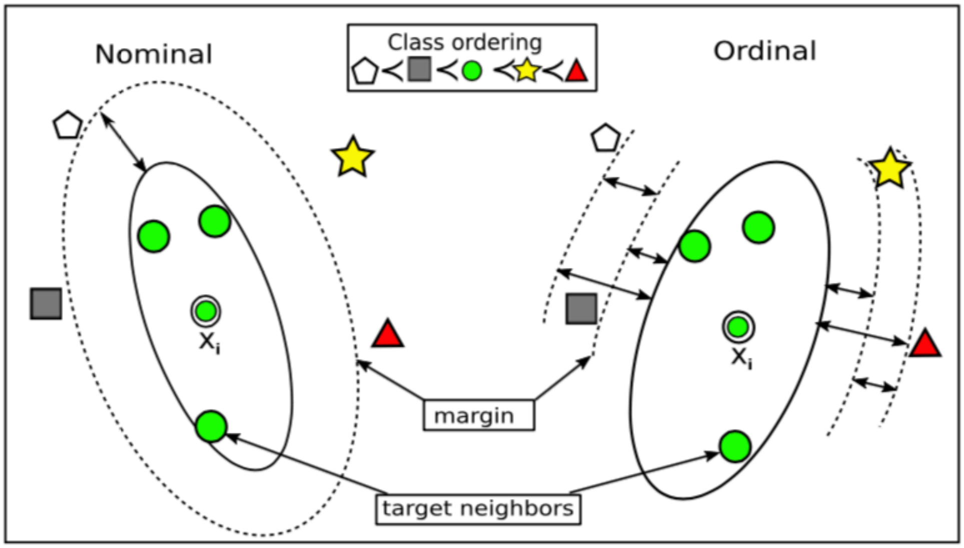


Figure 1 Local Neighborhood Ordered Classes vs Nominal Classes

The image on the left illustrates distance metric learning for the nominal classification problem. The image on the right illustrates the ordinal classification problem *[3]*

Chapter 3. DEEP LEARNING

Deep Learning falls under the broad class of Artificial Neural Networks (ANNs), which have origins that date back to the 1800s [21]. With its origins from simple Multi-Layer Perceptrons, Deep Learning is one of the primary Machine Learning strategies that are in wide use throughout the world with a history of solving a broad variety of data analysis and classification problems. Deep Learning is made up of a number of specialized classification strategies that have been derived from Deep Neural Networks (DNNs) which may also be called Artificial Neural Networks (ANN). DNNs originate from the Multi-Layer Perceptron, with the DNNs primary distinction as having a greater degree of complexity due to having more hidden layers and more nodes.

Like other machine learning methodologies, the application of Deep Learning algorithms falls into two general categories based on the “goal” of the application. Supervised applications have a targeted outcome that the algorithm attempts to predict based on other existing data. This targeted outcome is separate from the data that is being used for prediction. Examples of supervised problems are the prediction of category (class) membership (e.g., predict whether or not a picture has a dog in it) or predicting a volumetric outcome (e.g., how much money will a customer spend in the future). Unsupervised applications focus on the development of insight or understanding of the data without having a specific target with which the outcome may be compared to determine the accuracy of the mathematical model that is developed. Examples of unsupervised applications are data reduction techniques (e.g., autoencoders) which attempt to capture as much “information” from the existing data in a significantly fewer number of data elements. Unsupervised applications also include various forms of cluster analysis, which attempt to group data records into homogeneous sets while providing maximum separation between the groupings [21].

3.1. THE MULTI-LAYER PERCEPTRON

Figure 2 is a basic visual representation of a Multi-Layer Perceptron (MLP). The columns of circles are called a "layer" and each circle is called a "node." The arrows that connect the nodes represent numerical weights, that are calculated during the model estimation process. The dashed arrows represent a feedback process, that incrementally updates the weight values, through a process called "Back Propagation."

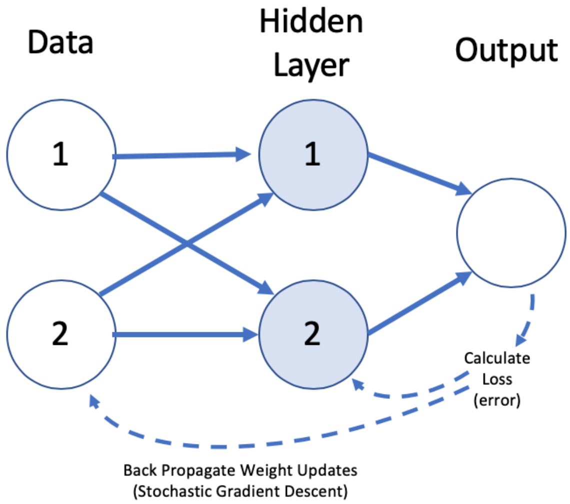


Figure 2 Simple Multi-Layer Perceptron with weight updates

The nodes also represent a nonlinear transformation of the input data, called the "activation function", after they are multiplied by their respective weights and summed. The activation functions provide the nonlinearity to the algorithm's learning process. Ideal activation functions are sufficiently simple and well behaved to allow the numerical estimation and update processes that are required for learning. Figure 3 includes the graphical representation of three of most common activation functions [22]:

$$\text{Sigmoid Function (AKA Logistic Function)}: \quad \frac{1}{1+e^{-x}} \qquad (2a)$$

$$\text{Rectified Linear Unit (ReLU)}: \quad \max(x,0) \qquad (2b)$$

$$\text{Hyperbolic Tangent (tanh)}: \quad \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad (2c)$$

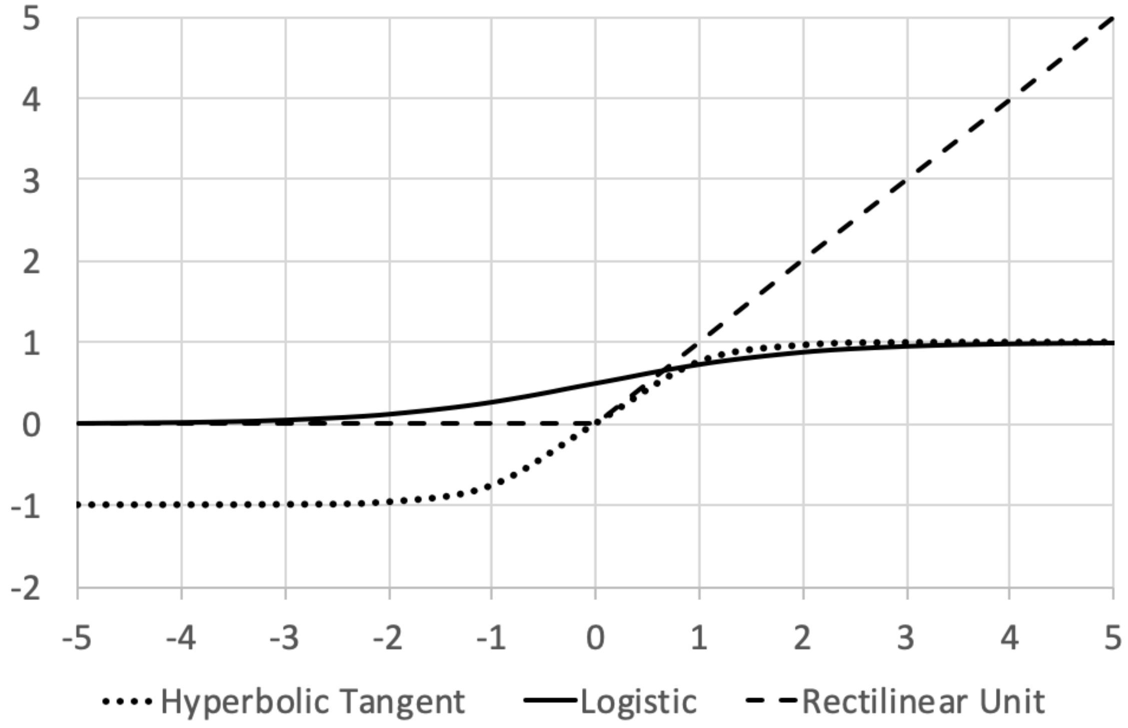


Figure 3 Plot of three common activation functions, found in Deep Neural Networks

As a general rule, Neural Networks, including MLPs, are initialized with small random weights. Each record of data is then fed through each node, by applying the corresponding weights, summing and then applying the activation function for the node. This process occurs in each node, layer by layer until the final output (output layer) is reached. At this point, the loss (i.e., classification error) value is calculated by comparing the output value to the “ground truth” that is represented by the label or target value for the individual data record. One of the most common loss functions is the summed squared error. If we denote the algorithm output as $\hat{y}$ and the ground truth value as y then the summed squared error for a DNN that is applied to dataset D, would be:

$$Summed\ Squared\ Error = \sum_{i \in D} (y_i - \hat{y}_i)^2 \qquad (3)$$

The weight values within each of the nodes are then updated via backwards propagation, represented by the curved dashed arrows, in Figure 2. The numerical basis for the weight updates is based on Stochastic Gradient Descent (SGD), which calculates the optimal update value via the application of partial derivatives, with respect to the respective weight values for the composition of activation functions that lead from the node to the output layer. Unlike Total Gradient Descent, which applies updates after all training records are fed through the neural network, Stochastic Gradient Descent, updates with each record [23]. However, in most cases today, SGD is applied to mini-batches of records to promote stability in estimating the gradient [21]. An explicit example of this weight update process is given in the Exclusive OR section (see Section 3.2).

When applied to data, SGD is calculated by using the sum of values across a sampling of the data. In real applications, SGD is not what is used in most Deep Learning algorithms [22]. Newer methodologies like Adam have a foundation in SGD, but address some of the issues of applying SGD to small batches and combining the result with prior weight update values, which leads to faster convergence of algorithms [24]. After the weights are updated, the process repeats.

One of the outputs of successive iterations of a DNN is a sequence of scalar value, that represents the total cost (error in estimation) for the iteration.

Definition1: *For* $a_n \in$ *sequence* A, *if there exists* a^* *where for given* $\varepsilon > 0$ *and all* $i \geq n$ *above a threshold*, $|a_i - a^*| < \varepsilon$, *then the sequence is said to converge to* a^* [25].

From a practical application, individual machine learning algorithms are not tested for this formal version of convergence, but the basic principle is applied. When the algorithm reaches the point that improvement in the cost function value ceases to occur or improvements are trivially small, the algorithm is said to have converged.

3.2. THE EXCLUSIVE OR PROBLEM (XOR)

The Exclusive OR (XOR) problem represents one of the simplest classification examples, where the labeled outcomes are not linearly separable in the space of available predictive attributes. The problem has four records with two attributes, x_1 and x_2 [26]. The labels of 'AND' represent combinations of x_1 and x_2 that are equal, while the desired labels of 'OR' will have a value of '1' for x_1 or x_2, but not both and '0' for the non-one (see Table 2). Geometrically speaking, the four records represent the corners of a box in two-dimensional space. In Figure 4, the desired 'OR' cases are represented by solid dots, while the 'And' case are circles. As illustrated in the figure, a circular shaped threshold provides the separation of the cases that is desired for the problem [26].

Table 2 XOR Data

Label	x_1	x_2	y
AND	0	0	0
OR	1	0	1
OR	0	1	1
AND	1	1	0

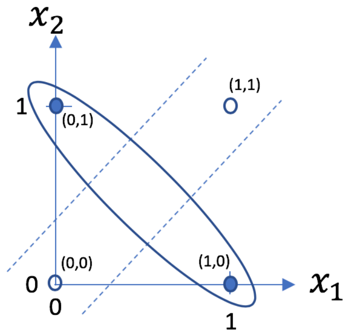


Figure 4 XOR Plot

To solve the problem, the labels are converted to binary 0 and 1 values (i.e., the '*y*' values in the table), since algorithms cannot use text directly. It can also be noted that

$$y = f(x_1,\, x_2) = 1 - (x_1 + x_2 - 1)^2 \qquad (3)$$

provides a perfect solution to the problem, but the vast majority of classification problems cannot be solved via simple visual inspection and educated guessing of a solution. If traditional statistical methodologies were used to attempt to provide a numerical formula to solve the problem, the analyst may attempt to fit a function of the form

$$y = f(x_1,\, x_2) = ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + \mathrm{g} \qquad (4)$$

where 'a'-'e', 'g' in (4) represent the unknown coefficients that the methodology would attempt to estimate to improve model fit. This framing of the problem results in four data points, with six unknowns, so no *unique* solution is possible. As such, classical statistical methodologies that are commonly applied would not work for solving the problem. As a direct consequence, the XOR problem may be the simplest problem that requires the nonlinear estimation power that is presented by ANNs.

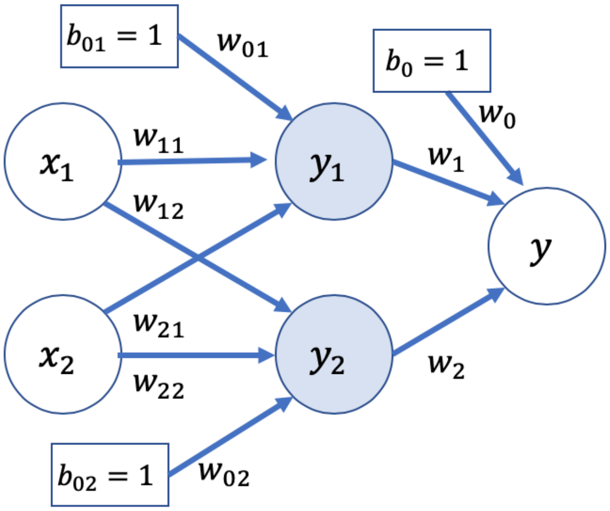


Figure 5: Fully Annotated XOR Neural Network Graph

Figure 5 represents the fully annotated network graph for solving the XOR problem. The boxes with 1's represent the constant or "bias" terms (w_0's) that need to be estimated along with the weights for the data elements. They were omitted from Figure 2, to provide a simplified visual introduction to neural networks. To solve the XOR problem, the process starts by using random values for the weight and bias. After each submission of the data points through the neural network is completed, the "loss" value, L is calculated, by summing the squared difference between the predicted value $\hat{y}$ and the correct label y as the error value [22]

$$L = \sum (y - \hat{y})^2. \qquad (5)$$

The gradients that are used for the weight updates are the partial derivatives with respect to the given weight and bias value. The sigmoid function is used as the node activation function (nodes y_1 and y_2) and is represented as σ_1 and σ_2 in equations (6) and (7). For each weight or bias value and for each data record n (represented by row number) the ANN update process uses the partial derivatives with respect to the weight (or bias value). The update for the output layer is [22]

For $i \in \{0, 1, 2\}$ representing the hidden layer node:

$$\frac{\partial L}{\partial w_i} = \sum_{n=1}^{4} \frac{\partial (y_n - \hat{y}_n)^2}{\partial w_i} = \sum_{n=1}^{4} 2(y_n - \hat{y}_n)\, y_{0,n} \tag{6}$$

The updates for the hidden layer (nodes *y1* and *y2*) are a little more complicated (see equation (7)).

For $j \in \{0, 1,\ 2\}$ and $i \in \{1, 2\}$, representing the data source and hidden layer nodes, respectively:

$$\frac{\partial L}{\partial w_{j,i}} = \sum_{n=1}^{4} \frac{\partial (y_n - \hat{y}_n)^2}{\partial w_{j,i}} = \sum_{n=1}^{4} 2(y_n - \hat{y}_n) \frac{\partial \sigma_{i,n}}{\partial w_{j,i}} = \sum_{n=1}^{4} 2(y_n - \hat{y}_n)\sigma_{i,n}\left(1 - \sigma_{i,n}\right)x_{j,n} \tag{7}$$

Note that equation (7) is effectively equation (6) with an added term, to represent the gradient from the output of the hidden layer to the input layer of the data. This chaining of gradient components has potentially serious implications if a large number of hidden layers are used in the neural network.

The gradients represent the direction and magnitude for increasing value at the current state of the system. To reduce the error terms the gradients are subtracted from the weights. As a general rule, a step size or "learning rate" is applied to the gradient before it is subtracted from the weight. Adjusting the step size can lead to a more efficient convergence to an optimal solution. Note that excessively large step sizes may even prevent the algorithm from achieving an optimal solution.

3.3. DEEP NEURAL NETWORKS

The most basic form of Deep Learning is a form of supervised learning called Deep Neural Networks. They are distinguished from simple MLP's in the number of hidden layers that are utilized. This deeper architecture comes with its own challenges. The calculated gradients may explode in size or vanish, if the multiplicative chain in the calculation has sufficiently large or small values, respectively, at each point in the chain [21].

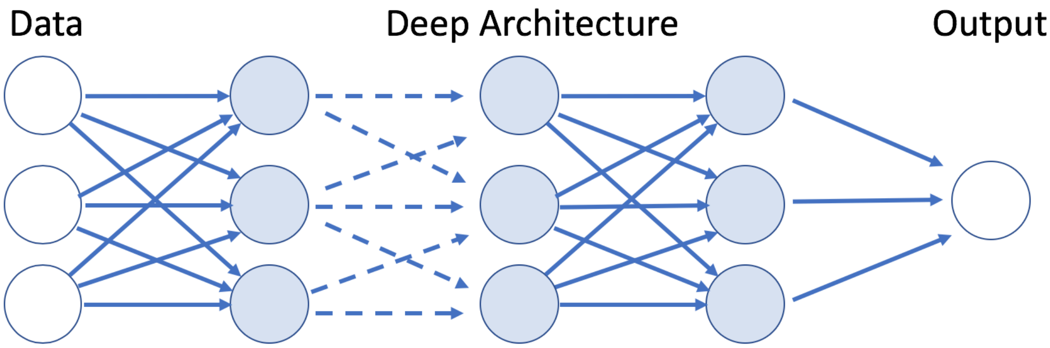


Figure 6 DNN Representative Graph

There are a number of strategies that may be employed to address this issue. For a period of time, the pretraining of network layers, using unsupervised learning techniques to establish initial weights then using back propagation to refine the weights for the full network was a useful strategy. More current architecture designs use Rectified Linear Units (ReLU) as the activation function to address the problem. In addition to minimizing the likelihood of vanishing/exploding gradients, the use of ReLU as the activation function in DNN nodes has also been demonstrated to improve algorithm performance [21].

For extremely deep neural networks, the use of ReLU activation functions does not always solve the vanishing/exploding gradient problem. Residual Neural Networks include additional connections in the graph that skip layers. These networks were used to handle very deep image classification problems with exceptional performance [27].

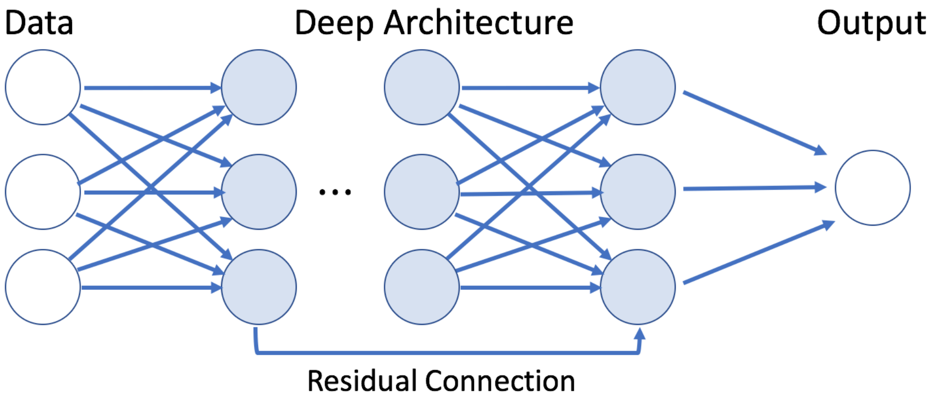


Figure 7: Residual Neural Network Graph

Deep neural networks are also applied to more challenging problems like image classification. The most obvious challenge in attempting to classify images, is the structure of the data itself. Images are two-dimensional if grey scale or three-dimensional if they are in color (e.g., a color image with red, green and blue layers). Reformatting an image to a one-dimensional array removes a significant amount of information from the data.

The standard approach to address image classification is the use of a Convolution Neural Network (CNN). In this approach, images are analyzed by systematically assessing small overlapping "patches" of the image (two-dimensional subsets of the image; if color then three-dimensional). Much like the input to output process for a DNN node, a set of weights is applied to the data, in the patch, which are summed, and a nonlinear activation function is applied. Each set of weights and activation function that is applied, in a pass over an image is called a “filter” to create a two-dimensional output. The application of multiple filters results in multiple two-dimensional outputs (called “channels”). A single “convolution” layer applies multiple filters producing a three-dimensional data object that is many times deeper than the original image (or prior layer output; see Figure 8). The data are then “pooled”, typically by taking the maximum value of a patch of the output channels (which may differ in size from the convolutional layer patches), to reduce the volume of data. Each patch is applied independently in a convolutional layer. Weights for the filters are updated across the entire layer (and mini-batch). Multiple iterations of convolutions and pooling may occur within an algorithm. At the end of these iterations, the data are reformed into a one-dimensional vector (the “Embedding” in Figure 8), that is then fed into a standard DNN layers [28].

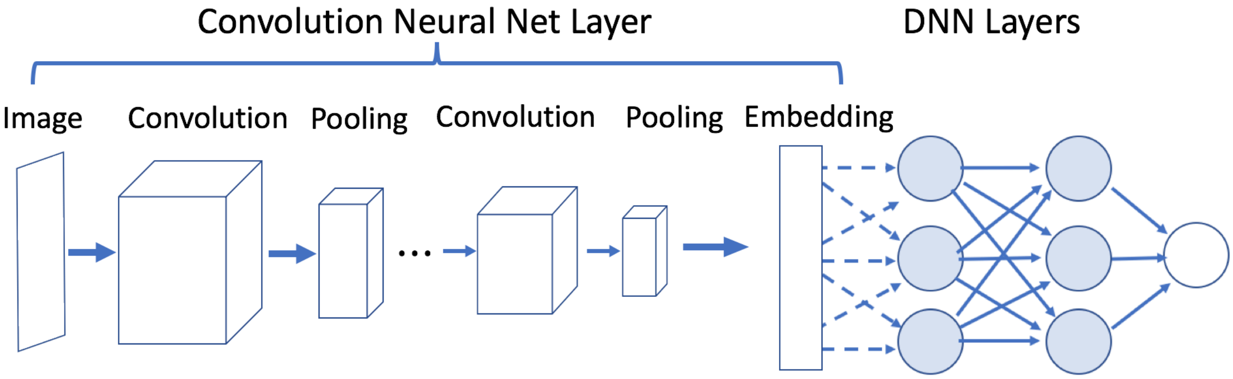


Figure 8 Basic Convolutional Neural Network Graph

A number of highly successful, general purpose CNNs are available for image classification. These CNNs are pretrained on very large image datasets and can be used to simply preprocess image data into the final one-dimensional layer or may be used as a pretrained CNN that refines the network weights through a training session. Examples of these pretrained CNNs are VGG16, ResNet50, AlexNet, GoogleNet and InceptionV3 [27, 29, 30, 31, 32].

The next generation of ResNet image classifier called “ResNeXt” [33]. Figure 9 comes from the publish paper by Xie, et. al. A key differentiator of ResNeXt and ResNet is the use of multiple parallel paths, which contain their own convolution and pooling layers as well as the inclusion of a residual path (essentially a second independent path of hidden layers from some output layer that is rejoined later in the neural network) [33]. The residual path may have the same architecture as the other path(s), but it also may differ. Before the two paths are joined their data structures must match, so great care must be taken in creating the residual paths [33].

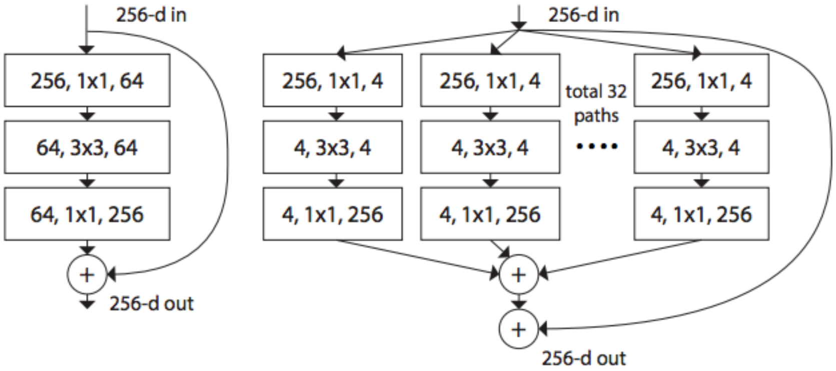


Figure 9: ResNeXt versus ResNet Architecture Fundamental Differences
Left: A block of ResNet. Right: A block of ResNeXt with cardinality=32. Layers shown as: # in channels, # out channels. Complexity is essentially equal [33]

One of the most complex Deep Learning algorithms is the Recurrent Neural Network or RNN. From a basic design point, the RNN graph is similar to a DNN. The difference in design comes from the recurrent connections which feed data backwards to an earlier layer within the network (Figure 10). This recurrence process makes RNNs well suited to handling time series or sequence data. The RNN may have a full DNN as the last layers of the network, or it may have a last layer that simply feeds the output layer [34].

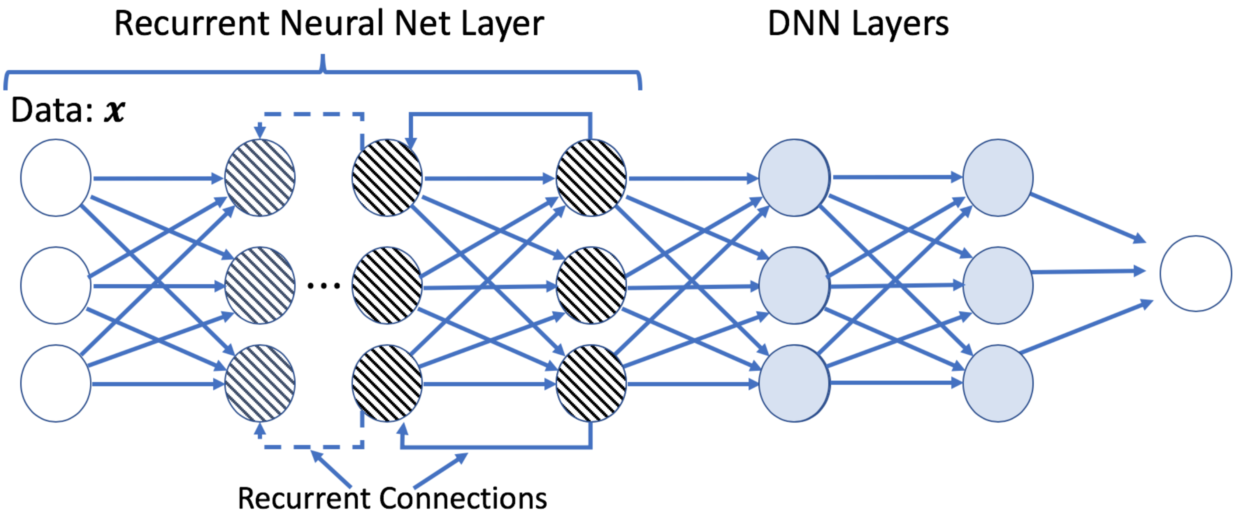


Figure 10 Recurrent Neural Network Graph

A useful type of unsupervised learning that comes from the Deep Learning is the Autoencoder. The goal of these neural networks is to reduce data dimensions (number features). They take an input dataset and process it through one or more hidden layers, that have significantly fewer nodes than the number of input features. The output layer has the same number of nodes as the number of input features. The input and output nodes are paired one to one. The loss function is the sum of the squared differences for the pairings [21]. Since the goal of the process is to reduce data dimensions, the number of nodes is usually significantly smaller than the number of input features. When training is completed, the layer with the smallest number of nodes represents the reduced data dimensions. Figure 11 is a simple representation, but auto encoders may have a deep architecture, particularly for extremely large data sources.

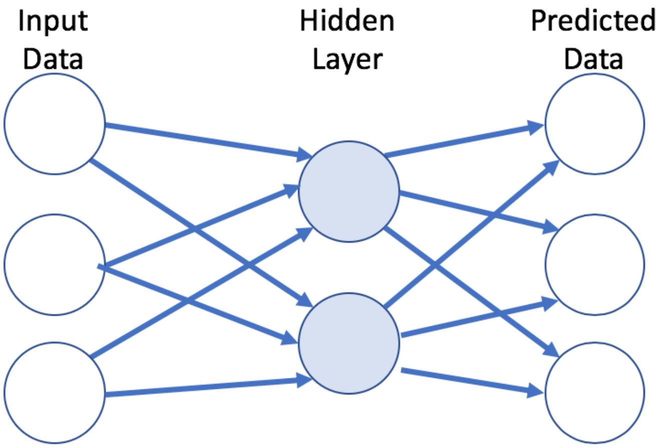


Figure 11 Simple Autoencoder

3.4. MINI-BATCH PROCESSING

Deep neural network algorithms have a computation complexity of $\mathcal{O}(n^5)$ [35]. As data set size, increases to today's "Big Data" levels of millions or billions of rows of data, the computation complexity, in submitting the entire dataset, in a single pass through the DNN is not possible. DNNs almost always use some form of small batch or mini-batch submission process. For a batch size *b*, that is significantly smaller than the full dataset size, the computation complexity of submitting a single batch is $\mathcal{O}(b^5)$. To submit all of the data in a data set with *n* records, $ceil(n/b)$ submissions must occur, so the computation complexity of submitting the full dataset, one time using mini-batchers is

$$\mathcal{O}(b^5) * ceil\left(\frac{n}{b}\right) \approx \mathcal{O}(nb^4) = b^4\mathcal{O}(n) \qquad (8)$$

$\mathcal{O}(b^4)$ is a constant, meaning the use of mini-batches takes an algorithm that has a computation complexity of $\mathcal{O}(n^5)$ and makes the problem linear in terms of number of records, $\mathcal{O}(n)$. In addition, the use of mini-batches has demonstrated improved generalizability of deep neural network models [36].

3.5. HINGE LOSS

Support Vector Machines were introduced by Vapnik et. al. in the mid 1990's [37]. While the name did not originate until sometime later, they created the concept of "hinge loss". For the vast majority of datasets that are not perfectly separable, the "soft margin" version was introduced that introduced a constraint of the form:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \zeta_i, \qquad \zeta_i \geq 0 \qquad (1)$$

Where 1, on the right-hand side, is the "margin" associate with the loss function (the margin can be set to a value of 1, without loss in generality). A more general version of this inequality, with nonzero margin, γ, could be expressed as:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq \gamma - \zeta_i, \qquad \zeta_i \geq 0, \qquad \gamma > 0 \quad (2)$$

It can be shown that this system of inequalities is equivalent to:

$$\zeta_i = \max(\gamma - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b), 0) \qquad (3)$$

This equation is the essence of the Hinge Loss function, where loss is zero, for function values below zero and loss contribution occurs when the function is above zero. For at least the past decade, Hinge Loss is one of the most common loss functions used in deep learning algorithms. This functional form is important in the creation and application of Ordinal Hyperplane Loss

(OHPL), where a simple linear difference of scalar values contributes to algorithm loss (error). If the value of the difference is positive the loss is set to that value. If not, it is set to zero. This function is continuous for all $\boldsymbol{x}$ and differentiable for all $\boldsymbol{x}$, except when $\boldsymbol{w}^T\boldsymbol{x}_i = -b$. Triplet Loss is a special application of Hinge Loss, that uses the difference in distance from a single point (called the positive anchor) to two other points, one of which has the same label as the positive anchor and the other has a different label. The value is zero unless the point with the unmatched label is not sufficiently further away from the anchor point, than the matched label point, by a preset margin. This function provides an easy way to focus deep net training, on "hard cases" that are a significant distance from the desired goal, while setting the distance, for cases that are close to the goal, to a value of zero. In developing OHPL, the underlying principles of Triplet Loss and Hinge Loss are combined, to develop a special loss function that directly addresses ordinal classification problem.

Chapter 4. ORDINAL CLASSIFICATION PROBLEM DESCRIPTION

This chapter covers the proposed solution to the Ordinal Classification problem that utilizes deep learning to directly develop a classification metric, a relatively intuitive mathematical and geometric motivation for the solution. The proposed strategy employs a commonly applied functional form that is used to develop large margin classifiers in machine learning. Conceptually, these frames of reference provide a foundation for the development of a unique loss function, that enables the application of virtually any deep learning architecture (DNN, CNN, RNN, etc.) to solve ordinal classification problems [38].

4.1. FUNDAMENTAL ORDINAL CLASSIFICATION PROBLEM

The proposed solution focuses on the identification or estimation of a nonlinear mapping, $\phi(x)$, that provides an optimal separation of classes, with three fundamental properties.

1. Different classes must be properly ordered. Numerically, they can be separated in either increasing order or decreasing but they must be properly ordered. In ensuring this property, the solution requires an assumption of monotonically increasing ordering without imposing an unnecessary and limiting restriction on distances between adjacent classes. Note that if ordering in the mapped space is naturally decreasing based on the optimal weights a simple multiplication of the output by -1 would ensure increasing ordering, so without loss in generality, the increasing ordering is set as the goal.

2. Borrowing generalizability benefits of large margin classifiers (per Vapnik et. al. [37]), the distances between classes must be non-trivial (i.e., not arbitrarily close to 0). Note that setting a minimum distance between classes does not impose regression like distance assumptions where the distance between two adjacent classes must be exactly one, since any non-zero distance may be rescaled to the minimum value while other distances would increase to a value greater than one. The simple multiplication of a constant would have no impact on classification. At the same time, some degree of regularization (upper bound or error penalty on the weights, a la Ridge Regression, Lasso Regression and SVM) must be employed to ensure that the minimum margin is not a simple rescaling of a trivial margin. Setting a minimum value avoids the challenge of implementing a rescaling component to the algorithm. No other distance assumptions that restrict relative class distances are applied.
3. Depending on the specific execution of the first two fundamental properties, it is possible that the group centers perfectly adhere to the minimum distance requirement, but the classifier behaves no better than random guessing. To avoid this scenario, the algorithm must learn a mapping that forms homogeneous sample classes clusters (i.e., provide for a clustering of the data, in the mapped space, that has homogeneous clusters, in terms of class). If the problem is not completely separable, then this property becomes a requirement of “near” homogeneous (or as close as possible to homogeneous). Under ideal circumstances, the results of these three strategy properties will provide a mapping, $\phi(\boldsymbol{x})$, as illustrated in Figure 12.

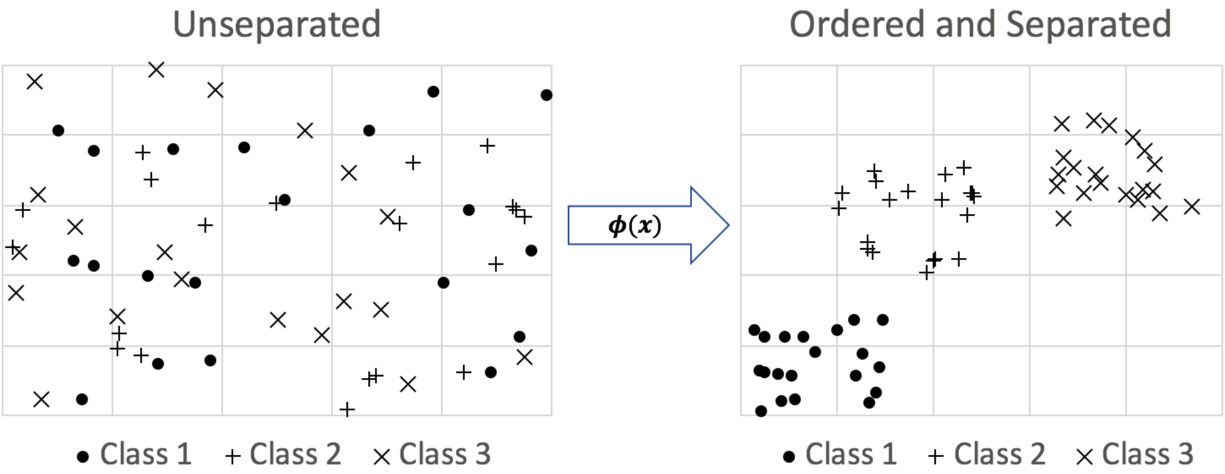


Figure 12 Ordered Separation of Classes

The goal of this research is to solve the ordinal classification problem by developing a deep learning strategy that can learn such an optimal mapping as described above from training data. Current best in breed algorithms that attempt to solve the ordinal classification problem use a predetermined function or set of functions and optimize a set of weights that minimize an associated cost function. Deep learning algorithms use the available data to learn highly complex nonlinear functions, without imposing a limitation of predefining the functional form. These nonlinear functions are most likely estimations of more complex functions, within the space that is represented by the available data.

Current ordinal classification applications using deep learning apply binary classification neural networks to develop a set of solutions to alternate problems. These strategies either classify one class versus another by analyzing multiple pairs of classes or one class versus all others (exhaustively using each class as the base class in at least one binary classifier). Additionally, they may employ a repeated sampling of available data with changes in the relative binary classes. In the case of the multiple classifiers, the models are then combined into a single ordinal classifier through an aggregation strategy (e.g., simple sums or weighted sums). Deep learning algorithms within a single model architecture (e.g., DNN, CNN) that solve different problems (e.g., regression, nominal classification) differ in their loss functions. This loss function may be applied across multiple model architectures. Developing a loss function that meets the problem requirements would not only enable the development of DNNs to develop ordinal classifiers but may also be broadly applied to other deep learning model architectures. This loss function forms the mathematical and algorithmic solution to the deep learning solution to the ordinal classification problem.

4.2. GEOMETRIC MOTIVATION

In developing Ordinal Hyperplane Loss (OHPL), a similar representation of data, that Ngyuen et. al. used in Figure 1, provides a geometric representation of the ordinal classification problem. In a fully separable problem, the perfect solution would to be a transformation $\phi(\boldsymbol{x})$ (usually nonlinear, but could be linear, if appropriate) that maps the unseparated classes into a new space. Figure 13, below represents a simple three class problem, represented and separated, in two-dimensional space. This illustration assumes a goal that is similar to that of the application of triplet loss, but with ordering of classes requirements. The goal of such an algorithm would be to “pull” all of the points in a single class as close as possible to the cluster center, while maintaining the ordering of the clusters. From this representation, the vector between points in adjacent classes, can be parsed into two components: 1) the component parallel to the vector between adjacent class cluster centers (solid lines) and 2) a component that is perpendicular, to the vector between the class cluster centers (dashed/dotted lines).

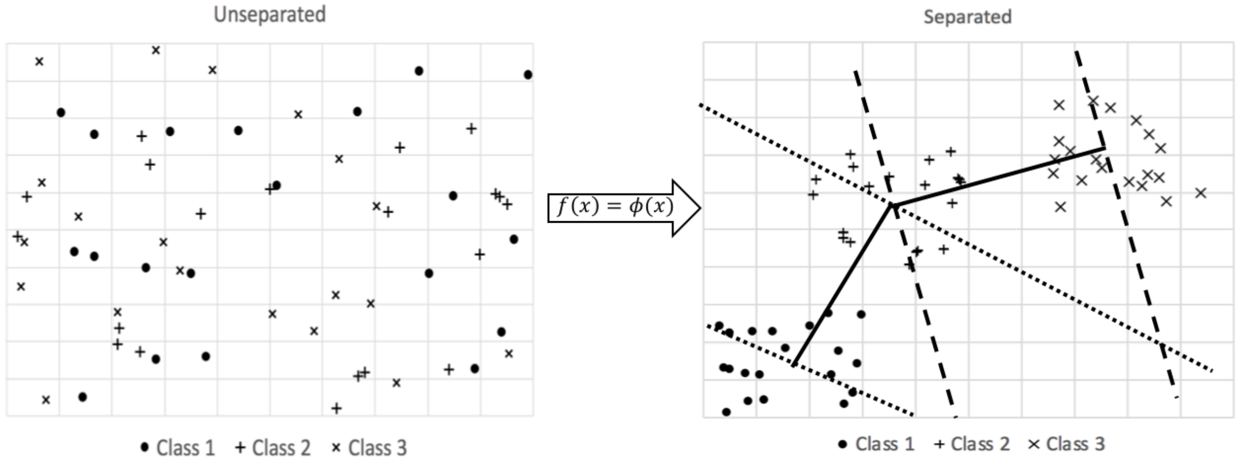


Figure 13 Separable Mapping $\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^2$.

Solid lines represent vectors between class cluster centers. Dotted lines are perpendicular to the vector between Class 1 and Class 2 cluster centers. Dashed lines are perpendicular to the vector between Class 2 and Class 3.

If the algorithm improves loss (error), in terms of pulling points closer to the cluster center, but does not improve error between classes (aggregate distance of all points in adjacent classes, from each other) or the movement is large for trivial improvement in error between classes, the movement is essentially perpendicular to the vector between the cluster centers (i.e., movement parallel to the dashed/dotted lines, in Figure 13). Not only does this “improvement” in terms of distance loss (error) not improve classification, but it may contribute to the over fitting of the model, in terms of its ability to generalize, to other datasets. This perpendicular direction introduces a hyperplane that may better represent the data, in terms of class membership as determined by point distances. If the parallel hyperplanes, going through the two cluster centroids, are used, to represent the cluster, then the distance from points, to these hyperplanes, defined as the length of a perpendicular line segment that connects a point to the hyperplane, would represent distances and direction where a potential change in the position of a point would guarantee improvement in the separation of the two classes.

Similarly, hyperplanes between other adjacent class pairs would provide significant benefit in truly separating other class pairs. In aggregate, this process introduces a new potential issue. If the hyperplanes are not parallel, then it is possible that when the algorithm is applied to a new data set, values near the intersection of two of the hyperplanes may be misclassified even though they have zero contribution to the loss function value that is used to estimate weights in the model. To address the possible issue of intersecting hyperplanes causing classification problems for points near the intersection, a requirement of parallel hyperplanes (see Figure 14) is applied. In doing so, reducing loss (error) that is in the direction of the solid line should provide an effective separation of classes. In addition, using position along a single vector reduces the loss (error) calculation to a simple difference of scalar values. Ensuring class ordering and reducing the distance from the class centroid may be efficiently addressed by an algorithm.

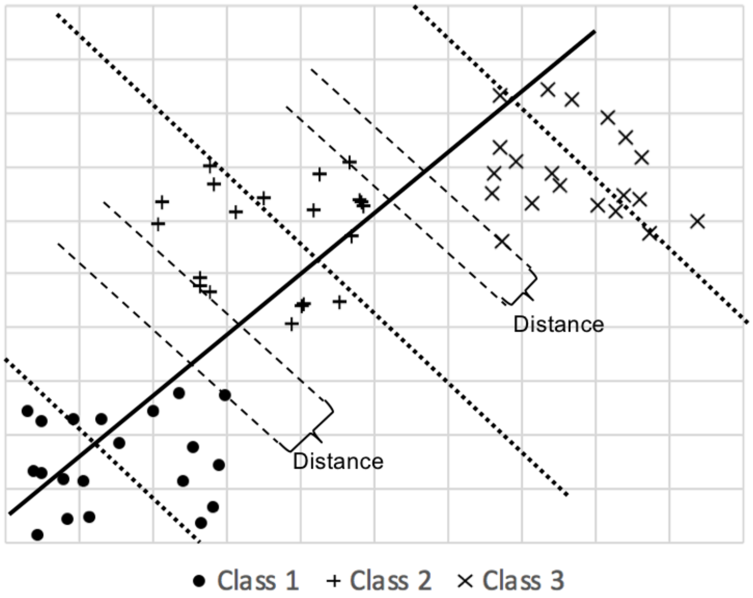


Figure 14 Parallel Hyperplanes.

For the same separable mapping, dotted lines represent parallel hyperplanes through the cluster centers, to which the solid line is perpendicular. Distance between hyperplanes, represent distance between points (e.g., dashed lines), that is aligned with the separation of all classes.

Chapter 5. OHPL – ORDINAL HYPERPLANE LOSS

This chapter reviews the novel loss function called Ordinal Hyperplane Loss (OHPL) which is specifically designed for predicting ordinal classes. OHPL enables deep learning techniques and strategies to be applied to the ordinal classification problem. The more complete application of OHPL within a deep neural net context is appropriately named *OHPLall*. More specifically, by minimizing OHPL, a deep neural network learns to map data to an optimal space where the distance between points and their class centroids are minimized while a nontrivial ordinal relationship among classes are maintained.

Class centroids, based on simple averaging of data values in the mapped space, provide a framework for imposing the fundamental property of class ordering and a mechanism for measuring distances between classes as well as a numerical framework for estimating loss/error contribution, due to inefficient class ordering. These class centroids can therefore be used to enable algorithm "learning" (i.e., improvements in performance/fit). Class distance may be defined distance between two class centroids. Setting a minimum distance threshold ensures that a non-trivial distance between centroids is created and maintained, while not imposing rigid regression like distance assumptions [38].

Once class ordering is established, data point distances from class centroids are used to ensure that points are closer to their class centroid than to unlike class centroids. Large margin methodology ensures "closer" does not mean trivially closer. Given the multi-dimensional nature of data, not all "distance" from a centroid that exceeds a threshold, is "error" in terms of class separation and classification. Decreasing the distance that is perpendicular to the line connecting the centroids of two classes, does not improve the separation of classes. Using parallel hyperplanes to define the class centroids ensures that the algorithm prioritizes individual point transformations that contribute to class cluster separation. In addition, this use of hyperplanes enables the use of scalar distances in a loss function [38].

The development of an efficient algorithm, based on this set of requirements, introduces the ability to apply deep learning to a broad set of core problems using structured data, in DNNs as well as Deep and Wide Networks. This new loss function provides a foundation for single image classification using CNNs including but not limited to primary problems of age estimation and medical classification (e.g., using MRI's to determine cancer stage). It would also enable more advanced applications like RNN which examine time series data, text (short statements or full documents) or possibly spatially sequential MRI images. In essence it provides a fundamentally new methodology for developing ordinal classifiers.

OHPL is an aggregation of two key components [38]:

1. Hyperplane Centroid Loss (HCL), which applies a large penalty within the algorithm, for violations of the ordering and minimal distance assumptions
2. Hyperplane Point Loss (HPL), which provides an error value, for points that violate the large margin boundary around the class Hyperplane Centroid, that is proportional to the distance from the boundary.

Based on the proposal of OHPL, a researcher may design a deep learning strategy that learns an optimal dimensional space where OHPL is minimized.

5.1. LINEAR HYPERPLANES

A given point $\boldsymbol{x}$ is a point on the hyperplane, defined by $\boldsymbol{w}$ and c, if $\boldsymbol{x}$ satisfies equation (2), where $\boldsymbol{w}$ and $\boldsymbol{x}$ are vector valued and c is a scalar constant.

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + c = 0 \quad (2)$$

A set of different parallel hyperplanes of this form will have the same coefficient vector, $\boldsymbol{w}$**,** and differ in their constant value c. The absolute value of c represents the 'distance' of the hyperplane, expressed as the minimum distance of the points on the hyperplane, from the origin. This concept of distance can be applied to two parallel hyperplanes (note that two nonparallel hyperplanes will intersect and therefore will always have a distance of zero). Given hyperplanes:

$$H_1 = \{\boldsymbol{x}: \boldsymbol{w}^T\boldsymbol{x_i} = b_1\} \quad (3a)$$

and

$$H_2 = \left\{\boldsymbol{x}: \boldsymbol{w}^T\boldsymbol{x_j} = b_2\right\} \ (3b)$$

$$\text{where } b_1 \neq b_2 \qquad (3c)$$

For a point, $\boldsymbol{x_1}$, on H_1 hyperplane, the 'distance' to a second, parallel hyperplane, is the minimal distance of the point, to any point on the hyperplane. By shifting the frame of reference for $\boldsymbol{x_1}$ and the hyperplane as follows, to create $\boldsymbol{x_1'}$ and H_1':

$$\boldsymbol{x_1'} = \boldsymbol{x_1} - \boldsymbol{x_1} \qquad (4)$$

$$H_1' = H_1 - b_1 = \{\boldsymbol{x}: \boldsymbol{w}^T(\boldsymbol{x_i} - \boldsymbol{x_1}) = b_1 - b_1 = 0\,\} \qquad (5)$$

$\boldsymbol{x_1'}$ is the origin and sits on a hyperplane through the origin that is parallel, to H_2. Define H_2' by applying the same transformation to H_2, to arrive at:

$$H_2' = \{\boldsymbol{x}\colon \boldsymbol{w}^T(\boldsymbol{x_j} - \boldsymbol{x_1}) = b_2 - b_1 = \ b_2{}'\} \qquad (6)$$

The absolute value of $b_2{}'$ is the distance of the hyperplane H_2' from the origin and

$$distance(\boldsymbol{x_1}, H_2) = b_2' = |b_2 - b_1| \qquad (7)$$

To generalize:

$$\forall\ \boldsymbol{x_i} \in H_1, \qquad distance(\boldsymbol{x_i}, H_2) = |b_2 - b_1| \qquad (8)$$

The distance between the two hyperplanes, $H_1, \text{and } H_2$ can be defined as follows:

$$distance(H_1, H_2) = |b_2 - b_1| \qquad (9)$$

In general, for a mapping, $\boldsymbol{\phi}(\boldsymbol{x})$, this concept of distance can be applied to two points, that respectively sit on hyperplanes, H_k and H_l, in the mapped space:

$$\boldsymbol{\phi}(\boldsymbol{x_i}) \in H_k = \{\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) = b_k\} \qquad (10)$$

and

$$\boldsymbol{\phi}(\boldsymbol{x_j}) \in H_l = \{\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) = b_l\} \qquad (11)$$

Then the distance, d, between $\boldsymbol{\phi}(\boldsymbol{x_i})$ and $\boldsymbol{\phi}(\boldsymbol{x_j})$:

$$d\left(\ \boldsymbol{\phi}(\boldsymbol{x_i}), \boldsymbol{\phi}(\boldsymbol{x_j})\right) = |\ H_k - H_l| = |b_k - b_l| \qquad (12)$$

For the purposes of writing a loss function, (12), for a computer algorithm application of the Hinge Loss function allows for the use of an algorithmically simple function as follows:

$$d\left(\ \boldsymbol{\phi}(\boldsymbol{x_i}), \boldsymbol{\phi}(\boldsymbol{x_j})\right) = \max(b_k - b_l, 0) + \max(b_k - b_l, 0) \qquad (13)$$

Similarly, OHPL is actually the combination of two loss functions, that utilize the distance between hyperplanes, as well as the specific distance function in (13). The algorithms use the concept of the “hyperplane centroid” as the fundamental definition of a class centroid, from which data points may be assessed, in terms of proximity to the centroid. In (14), the hyperplane centroid for class k and a given $\boldsymbol{w}$ is defined to be the mean value of all b_i for all *i,* in class *k,* where $\boldsymbol{w}^T \boldsymbol{x_{ki}} = b_{ki}$. For n_k samples in class *k*, the hyperplane centroid for class *k,* HC_k, is:

$$HC_k = \frac{1}{n_k} \sum_{y_i = k} \boldsymbol{w}^T \boldsymbol{x_{ki}} = \overline{b_k} \qquad (14)$$

5.2. HYPERPLANE CENTROID LOSS

First the component of the OHPL function, ensures that the hyperplane centroids are properly ordered, per the ordering of the classes. This ordering can be expressed as a difference in adjacent hyperplane centroids. If adjacent hyperplane centroids are properly ordered, then the transitive property ensures that all hyperplane centroids are properly ordered. For the purposes of developing a useful algorithm, not only do we need to achieve the ordering, of the hyperplane centroids for adjacent classes, but it is more desirable for the spacing be non-trivial. For fixed $\delta > 0$ and two adjacent hyperplane centroids, HC_{k+1} and HC_k, where the higher subscript denotes the higher class: [38]

$$HC_{k+1} - HC_k > \delta \qquad \text{for fixed } \delta > 0 \;\; (15)$$

Within the OHPL algorithm, adjacent classes *k* and *k+1*, and $\delta = 1$, the Hyperplane Centroid Loss contribution of HC_k relative to HC_{k+1} is:

$$\text{HC Loss}_{\text{k,k+1}} = \max(HC_k - HC_{k+1} + 1, 0) \qquad (16)$$

If HC_{k+1} is at least δ $d$istance from HC_k, then the ordering is correct with sufficient distance between the adjacent classes. For the k ordinal class problem, the Hyperplane Centroid Loss (HCL) is:

$$\mathrm{HCL} = \sum_{i=1}^{k-1} \max(HC_i - HC_{i+1} + \delta, 0) \qquad (17)$$

In the actual OHPL loss algorithm, HCL is coded as the multiplication of three matrices with hyperparameter δ (usually set to 1) added to each element of the resulting *k x 1* vector. Negative values in the resulting vector are set to zero, then the elements of the vector are summed to arrive at a total HCL loss value. This formulation of HCL is important when assessing the viability of applying OHPL to large data sets (e.g., 250,000 or more records) since it may be one of the limiting factors in the algorithm [38].

In the initial formulation, of OHPL, the hyperplane centroid ordering is applied to the full data training set, in batch within each iteration, through a data set. In the initial work, weighting is used to prioritize these relationships over the point loss effort to move points close to their corresponding hyperplane centroid. Experimental tests indicate very minor violation of the hyperplane centroid minimum distance requirement (less than 1%), but in all cases the distance between hyperplane centroids, can be demonstrated to be a nontrivial distance from zero [38].

5.3. HYPERPLANE POINT LOSS

The second component of OHPL is “Hyperplane-Point Loss” (HPL). In calculating this loss component, individual data points are compared to their corresponding Hyperplane Centroids. The primary goal is to “draw” points closer to their corresponding Hyperplane Centroid. This component of the algorithm provides a natural “regularization” of the model which limits the size of the weights. HPL is actually the sum of two analogous loss functions that work in different “directions” a la the formulation of (6) (see Figure 15 and Figure 16). This process is effectively an application of L1 distance (absolute value of the differences) which works effectively within the algorithm [38].

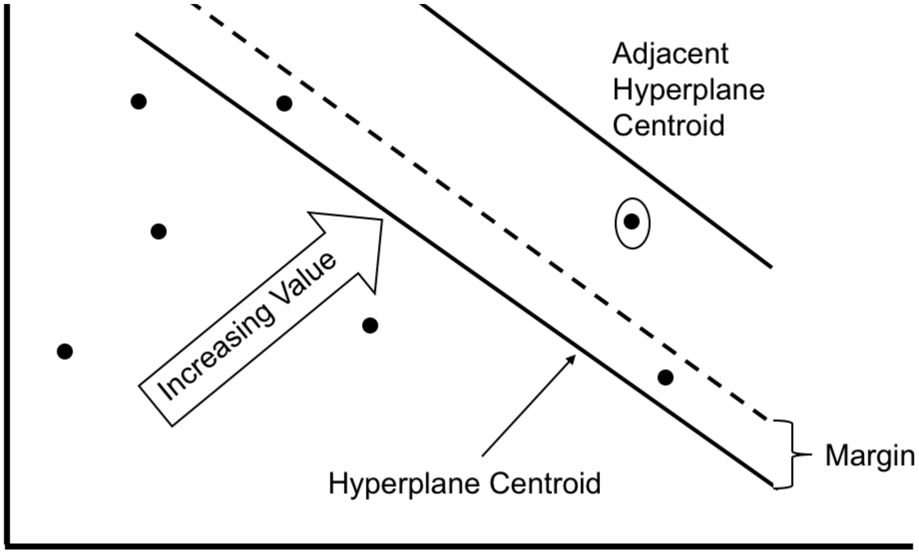


Figure 15 Hyperplane Point Loss - Increasing Direction.

Solid parallel lines represent adjacent Hyperplane Centroids. Dashed line represents the upper margin for the lower value ordinal class. In the increasing direction, points above the upper margin have nonzero contribution to the total loss [38].

For the points in a given class, “looking” in the “increasing” direction (corresponding to an increase in ordinal class value), the points that are higher than their respective hyperplane centroid may potentially contribute to the loss (those below will be examined later). For points that are above their hyperplane centroid, but are already sufficiently close to their hyperplane centroid, to result in a proper classification, drawing them closer to the hyperplane centroid won’t improve classification, so their loss contribution is set to zero. As a minimum, these points with zero contribution to the HPL must be closer than their distance to the next highest HC. Based on the success of other large margin classifiers, the HPL algorithm uses a margin that ensures that points are closer to their hyperplane centroid than the midpoint between the hyperplane centroid and the adjacent hyperplane centroid [38].

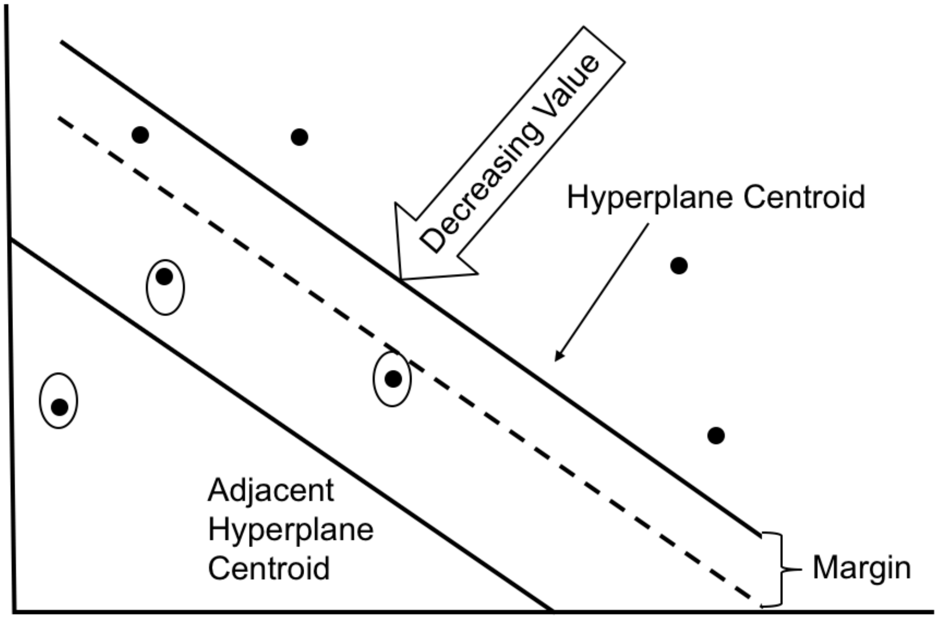


Figure 16 Hyperplane Point Loss - Decreasing Direction.
Solid parallel lines represent adjacent Hyperplane Centroids. Dashed line represents the lower margin for the upper value ordinal class. In the decreasing direction, points below the lower margin have nonzero contribution to the total loss [38].

In Figure 16, the circled points are lower than the margin below their hyperplane centroid, so it contributes to the total HPL value. Note that the dotted margin line/threshold is closer to the hyperplane centroid, than to the adjacent hyperplane centroid. Similarly, when we look in the decreasing direction, points that are further from their hyperplane centroid, than the margin, will contribute to the HPL total. In Figure 17, below, the seven circled points contribute nonzero values to HPL [38].

These two components of point related loss combine to produce a simple loss calculation, based on the subtraction of scalar values, combined with the application of the maximum function on two scalar values, creating a piecewise linear function, with two pieces. Figure 17 demonstrates the application of HPL to a simple three ordinal class example, in two dimensions [38].

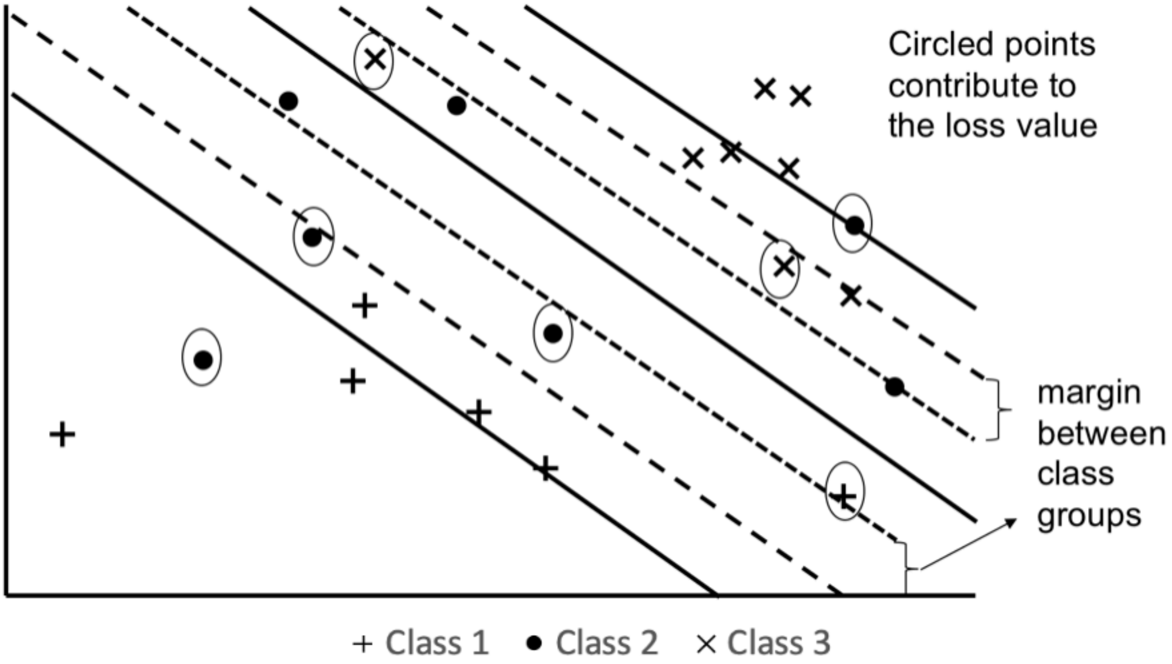

Circled points contribute to the loss value
margin between class groups
+ Class 1
● Class 2
× Class 3

Figure 17 HPL for Three Ordinal Class Case.

Solid parallel lines represent adjacent Hyperplane Centroids. Long dashed lines represent the margins for Class 1 and Class 3. Short dashed lines are the margins for Class 2. Circled points contribute to HPL and total OHPL [38].

As discussed in the Hyperplane Centroid Loss section, the distances between adjacent hyperplane centroids is not fixed. In fact, within the algorithm, they are not guaranteed to be greater than the set margin value. If not, the value contributes to the overall loss within the algorithm and are heavily weighted, so losses from this violation tend to be minor. On the other hand, there is no upper bound on the distance between adjacent hyperplane centroids. In these cases, a larger absolute margin may be used, leading to better algorithm efficiency. To account for the desired nonequal nature of the distances between adjacent hyperplane centroids, the HPL algorithm uses a fixed proportion of the distance between adjacent HCs [38].

The two components of the HPL algorithm (an increasing and a decreasing) are summed to arrive at the total loss contribution. To illustrate the "increasing" case set γ to be desired proportion of distance between adjacent hyperplane centroids then then let HPL^{+}represent the HPL for the direction of increasing class value. For point $\boldsymbol{x_i}$, in dataset S, and its corresponding hyperplane centroid, $HC$and the adjacent hyperplane centroid HC_{+} which is above HC [38]:

$$\text{for } 0.5 < \gamma < 1.0 \qquad (18)$$

$$point\ margin = \ \gamma(HC_{+} - HC) \qquad (19)$$

$$HPL_i^{+} = \max\left((f(\boldsymbol{x_i}) - HC) - (HC_{+} - HC) + \gamma(HC_{+} - HC), 0\right) \qquad (20)$$

$$= \max(f(\boldsymbol{x_i}) - \gamma HC - (1-\gamma)HC_{+}, 0) \qquad (21)$$

Similarly, for the decreasing case,

$$HPL_i^{-} = \max(\gamma HC - f(\boldsymbol{x_i})) + (1-\gamma)HC_{-1}, 0) \qquad (22)$$

$$HPPL = \sum_{\boldsymbol{x}_i \in S} HPL_i^{+} + HPL_i^{-} \qquad (23)$$

In the initial work, the two components of Ordinal Hyperplane Loss are combined, to arrive at the total loss. A weight η is applied to the HCL component of the loss calculation, to ensure a prioritization of class ordering over the reduction in point distance from the hyperplane centroid, to arrive at:

$$OHPL = \eta HCL + HPPL \qquad (24)$$

OHPL is applied within Deep Neural Network to create OHPLnet. Subsequent research, using OHPLnet, a found evidence that dynamically maintaining the hyperplane centroid distances was in direct conflict with the goal of drawing points closer to their hyperplane centroid. The evidence was mostly anecdotal in that some datasets experienced good results early on that were on par with other executions of the algorithm on the dataset. Then progress would stop well before achieving the minimum loss value and training set classification accuracy of other submission of the algorithm. The breakthrough that lead to the initial OHPLall variant (covered later) is the most compelling evidence that there was internal conflict between maintaining the hyperplane centroids and reducing point distances from the respective hyperplane centroids.

5.4. DEEP LEARNING STRATEGY BASED ON OHPL

Deep Neural Networks require structured data with a single 1 X m vector, per data record. Since OHPLnet addresses the specific error calculation requirements for ordinal classification tasks, it can be easily added to DNNs. The design is identical to any other DNN with the notable difference that the DNN maps the data into a new multi-dimensional space ($\boldsymbol{\phi}(\boldsymbol{x})$). From that point optimal weights, $\boldsymbol{w}$ and constant term $\boldsymbol{b}$, that define the hyperplanes is estimated to produce with a scalar output value. The scalar output is used to calculate prediction error (Figure 18). At that point, model estimation error, based on the OHPL design principals are used, instead of using another existing methodology to calculate error.

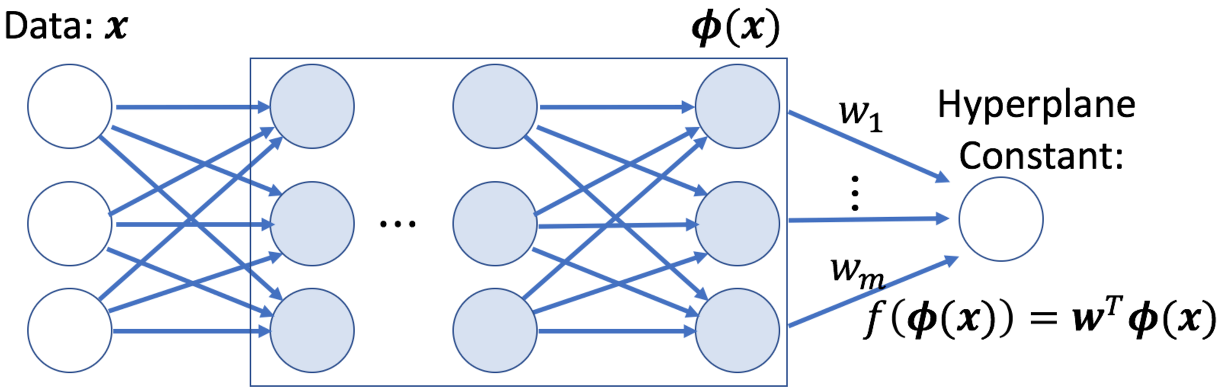

Data: $\boldsymbol{x}$
$\boldsymbol{\phi}(\boldsymbol{x})$
w_1
Hyperplane Constant:
w_m
$f(\boldsymbol{\phi}(\boldsymbol{x})) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})$

Figure 18 OHPL as a Deep Neural Network:

Design is identical to other DNNs with the exception of the linear output $f(\boldsymbol{\phi}(\boldsymbol{x}))$, which is used to calculate HCL and HPL loss components.

Similarly, an OHPL can be included as the DNN component of CNNs, to facilitate the analysis of images. Example problems would include:

- medical images that have ordered labeling (e.g., cancer stage)
- age recognition for facial images
- Facial emotional intensity
- Weather (e.g., satellite images to predict storm severity classes)

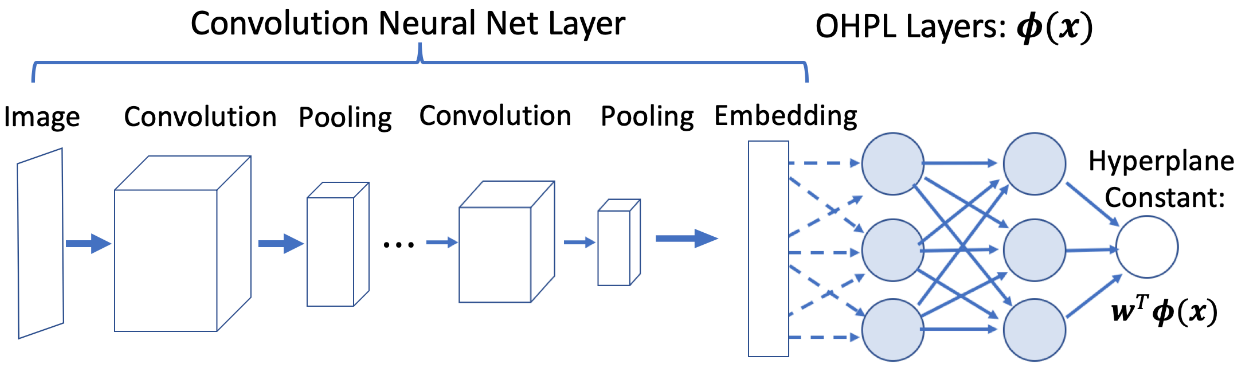

Convolution Neural Net Layer
OHPL Layers: $\boldsymbol{\phi}(\boldsymbol{x})$
Image
Convolution
Pooling
Convolution
Pooling
Embedding
…
Hyperplane Constant:
$\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})$

Figure 19 Convolutional Neural Network Graph with OHPLnet Neural Network Layers

The response value of Net Promoter survey responses has been linked to the recency of interaction of the respondent with the company's offering.) [39]. RNNs, with an OHPLnet layer could be employed to not only better predict customer response, gain insight into the 'drivers' of the response value, by removing the time component of response and then conducting a driver analysis, to identify key product attributes, that relate to rating.

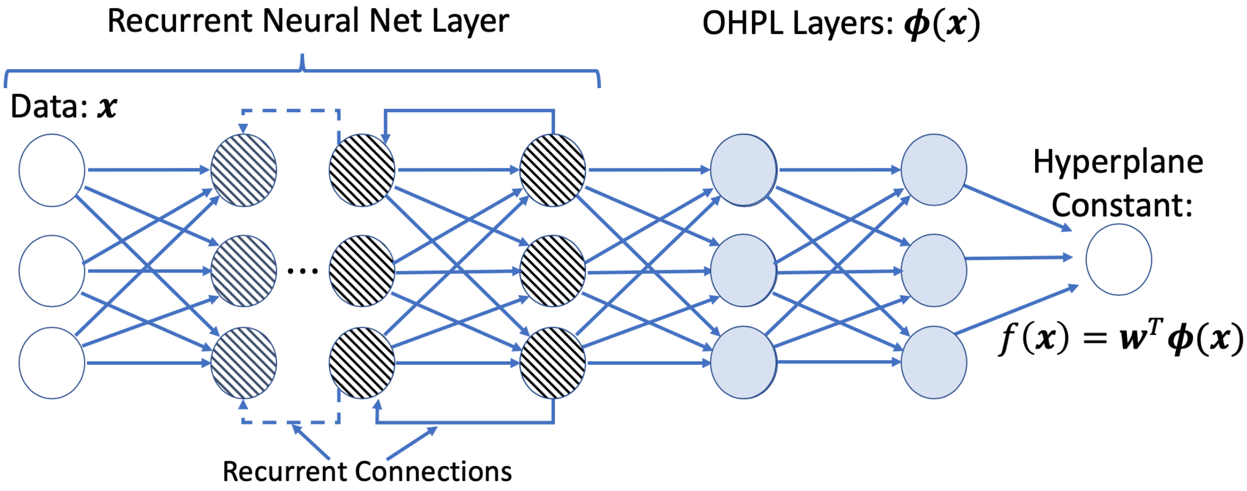

Recurrent Neural Net Layer
OHPL Layers: $\boldsymbol{\phi}(\boldsymbol{x})$
Data: $\boldsymbol{x}$
…
Hyperplane Constant:
$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})$
Recurrent Connections

Figure 20 Recurrent Neural Network Graph with OHPLnet Neural Network Layers

In all cases, it is conceivable that, in these more complex neural networks (e.g., RNN, CNN, ResNet), the layer that “feeds” the OHPLnet is a scalar value, that has a nonlinear activation function, prior to output. In this case, the hyperplane centroids would simply be a point on the number line.

As the use of Deep Learning expands, more complex and varied neural network designs are being created. Since OHPLnet does not depend on the architecture that precedes the output layer, it is well suited to being applied to new network designs as they are developed.

OHPLnet ALGORITHM: Iterative Algorithm

```
Hyper-Parameters:
h – number of hidden layers
$l_h$ – number of nodes per layer
$\alpha$ – prioritization weight for HCL
lr – learning rate
m – HC margin
γ– point margin proportion
bs – batch size
Input:  Rescaled training data $\{(\boldsymbol{x}_i, y_i) | i=1,…,n\}$
        Parameters h, $l_k$, $\alpha$, lr, $\{l_k = 1,…h\}$
Begin:
        Randomize weight (W) and bias (b) in each DNN node
        While not converged do
                OHPL = 0, HPL = 0, HCL = 0
                Feed full dataset through selected ANN
                From ANN Output, Calculate HCL:
                        Calculate difference in adjacent centroids
                        Subtract m
                        Sum positive values, as HCL
                Select mini-batch (bs)
                        Calculate mini-batch output from network
                        Calculate distances from respective hyperplane centroids
                        Sum positive values, as HPL
                OHPL = HPL + $\alpha$*HCL
                Calculate Stochastic Gradient Descent (SGD)
                Update W and b via SGD and lr
```

```
    Repeat until training sample exhausted
    Check convergence
End: Output W and b
```

5.5. SCALING TO LARGE DATASETS

Truly large ordinal datasets, while real, have not been tested with algorithms for which benchmark performance can be found. For this reason, experimental results for the performance of OHPLnet on large datasets (100K+ records) is not available. An effort was made to include the largest datasets that have been tested with other algorithms. The largest of these has fewer than 25000 records [38].

Heuristically speaking, OHPLnet should be scalable to any dataset that can be analyzed using a DNN. The HPL component is applied to mini batches, a process that allows DNNs, to be applied to very large datasets. This leaves HCL is the potential limiting factor [38].

Per the HCL algorithm discussion, the loss is calculated using simple matrix multiplication, to calculate means by class. If a data set is so large that it cannot be computed within a single computer the algorithm allows for breaking up the processes into as many pieces (submatrices) as necessary (e.g., parse the matrix, by row into sizes that can be computed). Summaries by class are calculated for each of the submatrices, then those resulting vectors are summed (position by position). The elements of the resulting vector are divided by the sample counts for the corresponding class. While breaking up the problem would be less than ideal, since there are computational costs for doing so, it is a viable process that could be run in parallel on multiple processors [38].

OHPLnet was developed to allow the application to large datasets. In this case, we define “large” as 200000 records or more. The benchmark algorithms reported by Gutiérrez, et. al. and the very recently reported algorithm that was developed by Nguyen, et. al., use estimation processes like SVM or Gaussian Processes (or algorithms that are very similar to the point that their estimated computational complexity is $\mathcal{O}(n^2)$ for the most efficient algorithms), making them potentially unsuitable to apply to datasets that are appreciably larger than 100K records (or require complex processing strategies, to do so). An example application is reported in the OHPLnet results section.

Chapter 6. EXPERIMENTAL RESULTS FOR OHPLNET

Experimental tests used Python version 3.6.3, in Jupyter Notebook 5.0.0. with Google's Tensorflow 1.5.0 [40] and several packages from Sci-Kit Learn (e.g., StratifiedKFold and shuffle) [41], Numpy [42] [43] and Pandas [44]. Development and analysis work were split between a MacBook Pro (Retina, 15-inch, Mid 2015) and a desktop with an AMD FX-8350 processor, 12GB of DDR3 RAM and a Nvidia GEFORCE GTX 1080ti GPU. Classification datasets were chosen from datasets that are found in a number of related studies. For benchmark purposes, with the exception of the LODML linear classifiers produced by Nguyen et. al. [3], the results that are reported by Gutiérrez, et. al. are used [9].

6.1. EXPERIMENTATION: STANDARD TEST DATASETS

OHPLnet was tested against seven ordinal classification datasets that are found in a number of related studies. For benchmark purposes, the results that were reported by Gutiérrez, et. al. are used [9]. The Cars and Red Wine datasets contain typical ordinal classes, for ~1,600 records. They come from the UCI (University of California Irvine) dataset repository [45].

From the from the Chu and Ghahramani research, [8] the CPU Small and Census 10 datasets are used. These datasets are among the largest for which benchmark results are available. They represent a very difficult problems where the ordinal classes are created by producing equal size binning of records using a sorted continuous variable (or as close as possible, given 10 bins and a number of records that are not a multiple of 10). For both datasets, continuous values were split into 10 bins, representing ordinal response classes. Chu and Ghahramani provided a MatLab script that allows prospective researchers to create an identical binning [46].

The ERA (Employee Rejection/Acceptance), LEV (Lecturers Evaluation) and SWD (Social Worker Decisions) datasets were introduced by David [47] and can be found at http://mldata.org.

Table 3 Test Dataset Key Characteristics

	# Records	# Features	# Classes	Average # Records per Class	Class Distribution
CPU Small	8,192	12	10	819.2	~820 per class
Census 10	22,784	16	10	2,278.4	~2,278 per class
Cars	1,728	6	4	432	(1,210, 384, 69, 65)
Wine-Red	1,599	11	6	266.5	(10, 53, 681, 638, 199, 18)
ERA	1,000	4	9	111.1	(92, 142, 181, 172, 158, 118, 88, 3, 18)
LEV	1,000	4	5	200	(93, 280, 403, 197, 270)
SWD	1,000	10	4	250	(32, 352, 399, 217)

6.2. ALGORITHM ASSESSMENT

There are two standard assessment tests that are used to assess performance in attempting to predict ordinal classification data. The MZE test is also used to test classification of nominal data. For ground truth values, y_i and prediction values $\hat{y}_i$, the test reports the proportion of misclassifications when scoring the validation samples. MZE is explicitly computed as:

$$MZE = \frac{1}{N}\sum(y_i \neq \hat{y}_i) = \frac{1}{N}\sum(y_i \neq \hat{y}_i) + \frac{1}{N}\sum(y_i = \hat{y}_i) - \frac{1}{N}\sum(y_i = \hat{y}_i) \qquad (26a)$$

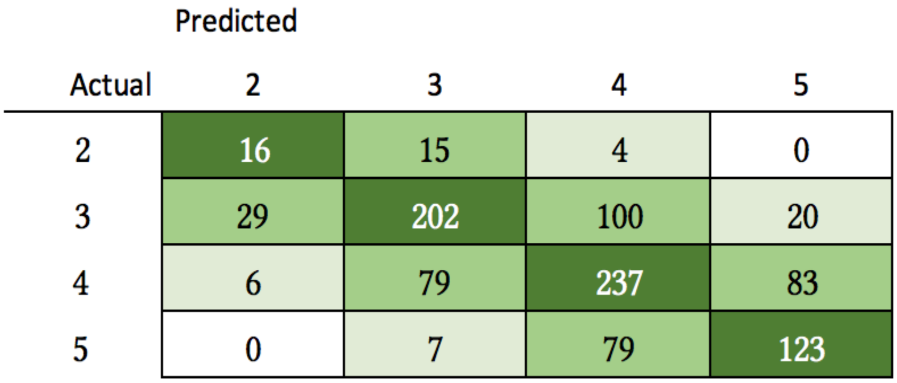
$$= 1 - \frac{1}{N}\sum(y_i = \hat{y}_i) \ = 1 - accuracy \qquad (26b)$$

Mean Absolute Error (MAE) is the standard measure of closeness that is used to assess the performance of ordinal classifiers. Not only may MAE be a more meaningful metric for special cases, it also may be a more meaningful way to access model performance in general. In calculating MAE, for each record the absolute difference between actual class and predicted class is calculated. The mean of these values becomes the MAE score for the algorithm, when applied to the given dataset, as follows:

$$MAE = \frac{1}{N}\sum|y_i - \hat{y}_i| \qquad (27)$$

As a minimum, MAE is a powerful way to distinguish among models that have comparable MZE performance. It should be noted that the MAE metric was a primary motivating factor in deciding to use a variant of the L1 norm for OHPL, instead of the L2 norm, that is more commonly used in data analysis and machine learning methodologies.

Figure 12 provides a visual illustration of the fundamental difference between MAE and MZE. A standard methodology to assess classifier performance is the use of a “confusion” matrix. The basic principle is to use the classifier to score a dataset that has known labels, giving each record an actual and a predicted class value. The actual values correspond to the rows of the matrix and the predicted classes are represented in the columns. Every record is an ordered pair that occurs within the matrix. Cells of the matrix are filled with counts of the corresponding ordered pairs. Assuming that the row and column sequence is the same, then the diagonal (darkest colored cells in the matrix below) represents the correctly classified counts, which sum to the MZE value, before dividing by the total number of records.



	Predicted			
Actual	2	3	4	5
2	16	15	4	0
3	29	202	100	20
4	6	79	237	83
5	0	7	79	123

Figure 21 Color Coded Confusion Matrix.

Entries are counts for ordered pairs of actual and predicted classes. Darker colored cells represent “closer” agreement of actual and predicted values.

As you move further from the diagonal of the matrix, the values get lighter in color (further in color from the diagonal). This color change represents increasing error, in the classification and the lighter the color, the higher the error for points that are represented in the cells. An ideal classifier, that is not a perfect classifier will have zeros in the three lightest colors in the Color-Coded Confusion Matrix (Figure 21).

6.3. BENCHMARK ALGORITHMS

The POM algorithm had slightly better results on the CPU Small data set (0.580 vs 0.588 for the GPOR algorithm), but the algorithm performed so poorly on the other datasets, that it was removed in favor of the GPOR. In addition, the results for the POM algorithm were excluded from the mean* of the algorithms reported by Gutiérrez, et. al. (a total of 15 algorithms were included in the mean calculation, reported in Figures 5 & 6 of the paper) [9].

An additional motivation for including the GPOR is its overall status as the best performing algorithm across the 41 benchmark datasets that Gutiérrez, et. al. examined. The ORBoost and SVMOP are selected due to their excellent performance across the seven included datasets.

Four Benchmark Algorithms :

1. Support Vector Machines with Ordered Partitions (SVMOP): The algorithm converts a problem with *k* classes into *k-1* binary classification problems. SVM classifiers are estimated for each problem, using an error weighting that is proportional to the absolute difference in classes. In specifying the classifier that discerns whether the label for record $\boldsymbol{x}_i$ is larger than *p*, the error weight for the record is abs$(y_i - p + 1)$ [14]. From that point, a standard strategy for using k-1 classifiers is employed (e.g., assess the classes in order, then choose, one class prior to the first instance of a positive objective score)
2. GPOR is Wie and Ghahramani's Gaussian Process for Ordinal Regression [8]. Their algorithm uses a Bayesian framework to estimate latent functions $\{f(\boldsymbol{x}_i)\}$. Mercer kernel functions are used to explicitly define the covariance of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, in the Gaussian kernel as:

$$Cov[f(x_i), f(x_j)] = \mathcal{K}(x_i, x_j) = \exp\left(-\frac{\kappa}{2}\sum_{\zeta=1}^{d}\left(x_i^{\zeta} - x_j^{\zeta}\right)^2\right) \qquad (28)$$

making the prior probabilities, of $\{f(\boldsymbol{x}_i)\}$, a multivariate Gaussian:

$$\mathcal{P}(f) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}f^T\Sigma^{-1}f\right), \qquad \Sigma \text{ is nxn with elements defined in (20)}$$

The joint probability is written:

$$\mathcal{P}(\mathcal{D}|f) = \prod_{i=1}^{n} \mathcal{P}(y_i|f(\boldsymbol{x}_i)) \qquad (29)$$

By Bayes' theorem, the posterior probability is:

$$\mathcal{P}(f|\mathcal{D}) = \frac{1}{\mathcal{P}(\mathcal{D})} \prod_{i=1}^{n} \mathcal{P}(y_i|f(\boldsymbol{x}_i))\mathcal{P}(f) \qquad (30)$$

The kernel parameters κ, in (28), the threshold parameters ($\{b_1, \Delta_2 \dots \Delta_{r-1}\}$) and Gaussian noise (assumed to have 0 mean and unknown variance) can be collected into the hyperparameter vector, θ, meaning the normalization factor $\mathcal{P}(\mathcal{D})$ is more precisely stated as $\mathcal{P}(\mathcal{D}|\theta)$ (called the "evidence for θ". Monte Carlo methods could be used to integrate over the θ-space but are considered to be too compute expensive for applications to most real datasets. Instead, Wie and Ghahramani use two approaches to determine the optimal values for θ: 1) Laplace Approximation and 2) Expectation Propagation. For optimal hyperparameters θ^*, prediction of ordinal class is explicitly stated as:

$$\hat{y}_j = argmax\, \mathcal{P}\left(y_j = i|\boldsymbol{x}_j, \mathcal{D}, \theta^*\right) \qquad (31)$$

3. Ordinal Regression Boosting (ORBoost) with "All Margins" (ORBALL) was developed by Lin and Li [48]. Per the authors, ORBoost is essentially an extension of the RankBoost algorithm developed by Freund, et. al [49]. Their algorithm estimates a set of "ordered" weak binary classifiers (binary variable is set to mimic, label ordering) based on subsets of the feature set. The classifiers are iteratively accumulated using weighting to maximize classification accuracy within the training set. Training is stopped when addition of new

weak classifiers fails to make meaningful improvements in training set classification accuracy.

4. For LODML, training samples are segmented into “target neighborhoods” with as many as q samples in each target neighborhood, triplet loss constraints are developed to preserve the class ordering of labels, within the target neighborhoods [3]. For the constraints, Nguyen et. al. use a Mahalanobis distance metric of the form:

$$d_{\boldsymbol{M}}^{2}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^{\boldsymbol{T}} \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_j) = \langle \boldsymbol{M}, (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^{\boldsymbol{T}} \rangle \qquad (32)$$

where $\boldsymbol{M}$ is symmetric, positive semidefinite. For $\mathcal{R}$, the set of constraints, their algorithm solves the problem:

$$\min \alpha \operatorname{tr}(\boldsymbol{M}) + \frac{1}{m} \sum_{(i,j,l) \in \mathcal{R}} \xi_{i,j,l} \qquad (33)$$

$$s.t.\, d_{\boldsymbol{M}}^{2}(\boldsymbol{x}_i, \boldsymbol{x}_l) - d_{\boldsymbol{M}}^{2}(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 1 - \xi_{i,j,l} \qquad (34)$$

$$\xi_{i,j,l} \geq 0, (i,j,l) \in \mathcal{R} \qquad (35)$$

$$\boldsymbol{M} \succcurlyeq 0 \qquad (36)$$

Nguyen et. al. also produced nonlinear versions of these algorithms which employ the kernel trick, much like SVM. LODML performs on par with the nonlinear version and has test results that span a large number of test datasets, so LODML is used in this assessment.

6.4. BENCHMARK RESULTS

Fig 5 compares MZE for OHPLnet versus the top performers for the individual data sets. The OHPLnet results are based on executing the algorithm using a five-fold cross validation strategy. Because neural network solutions are dependent on their starting weights, a researcher will typically develop multiple models, then use a predetermined selection criterion to choose the best model. The initial variant of OHPLnet tended to have inconsistent results. For every good model result, there tended to be a result that was disappointing. For this exercise, the algorithm was executed 50 times, on each fold and the 20 best results, based on training set MAE, was selected.

For the small "traditional" ordinal data sets (i.e., not "CPU Small" or "Census 10"), OHPLnet performs on par with the more complex algorithms. It is the larger data sets with larger numbers of ordinal classes where OHPLnet achieves demonstrably better results. When applied to the CPU Small OHPLnet improves MZE by 27% over GPOR (the second-best performing algorithm; 26% over POM).

Table 4 MZE Results for OHPLnet versus Benchmark Algorithms

	SVMOP	GPOR	ORBALL	Mean*	LODML	**OHPL**
CPU Small	0.631	0.588	0.654	0.634	0.569	**0.428**
Census 10	0.771	0.749	0.774	0.774	0.7373	**0.635**
Cars	**0.003**	0.037	0.012	0.038	0.028	0.024
Wine-Red	0.358	0.394	**0.334**	0.374	0.432	0.431
ERA	0.745	**0.712**	0.760	0.752	0.828	0.722
LEV	**0.367**	0.388	0.391	0.381	0.490	0.399
SWD	0.424	**0.422**	0.439	0.437	0.529	**0.422**

Table 2 compares MZE for OHPLnet versus the top performers. When applied to solve the Census 10 problem, OHPLnet improves MAE by 16% over GPOR (and 18.3% over the mean score). OHPLnet performs even better when using MZE, where the score improves by almost 50% versus the best benchmark algorithms (both are either best of the 16 reported or differ from best by less than 1%). The

When MZE scores differ as much as reported for the CPU Small and Census 10 data sets, it should be expected to see improved MAE values, since there are significantly fewer values, that have a nonzero MAE contribution.

Table 5 MAE Results for OHPLnet versus Benchmark Algorithms

	SVMOP	GPOR	ORBALL	Mean*	LODML	**OHPL**
CPU Small	1.06	0.92	1.04	1.05	0.895	**0.735**
Census 10	1.64	1.64	1.51	1.683	1.597	**1.247**
Cars	**0.003**	0.04	0.01	0.04	0.034	0.024
Wine-Red	0.41	0.42	**0.37**	0.42	0.516	0.488
ERA	1.61	**1.24**	1.25	1.366	1.836	1.610
LEV	**0.40**	0.42	0.43	0.418	0.622	0.438
SWD	0.45	**0.44**	0.46	0.464	0.616	0.459

It can be useful to combine MZE and MAE, into a single metric by taking the ratio of the two (MAE/MZE; see Table 6). This new metric is essentially equivalent to a conditional error (error for incorrect classifications). For the CPU Small, this ratio is 1.16 (so MAE is 16% above MZE). The same ratio for the mean of the 15 models is 1.66 and the best ratio of the four top algorithms is 1.56 (so 56% above the MZE). Equivalently, on the Census 10 dataset, the MAE/MZE ratio for OHPLnet is 1.15, while the best ratio for the four benchmark algorithms is 1.95. For these larger datasets, even when restricting the assessment to mean error of misclassified records, OHPLnet represents a significant improvement over the best existing algorithms. For some applications of ordinal classes, ranges of values are group into important classes. Net Promoter Score reviewed in Chapter 8 is a good example of this process. In those cases, the error in misclassification has a very high degree of importance, since it should lead to better results in the grouped classes.

Table 6 MAE/MAE Results for OHPLnet versus Benchmark Algorithms

MAE/MZE	SVMOP	GPOR	ORBALL	Mean*	LODML	OHPL
CPU Small	1.680	1.565	1.590	1.656	1.573	**1.099**
Census 10	2.127	2.190	1.951	2.174	2.166	**1.121**
Cars	1.000	1.081	1.000	1.053	1.183	**1.000**
Wine-Red	1.145	**1.066**	1.108	1.123	1.194	1.159
ERA	2.164	1.742	1.645	1.816	2.217	**1.034**
LEV	1.090	**1.082**	1.100	1.097	1.269	1.098
SWD	1.061	**1.043**	1.048	1.062	1.158	1.088

During follow up analysis, it was discovered that while the OHPLnet algorithm did achieve the 0.745 MZE and 0.769 MAE scores on the ERA dataset that were reported in *Ordinal Hyperplane Loss* [38], these scores were invalid due to the algorithm failing to establish and maintain the proper ordering of classes (the highest class had a hyperplane centroid value that was lower than the prior hyperplane centroid). The correct values are reported in Table 4 and Table 5.

6.5. APPLICATION TO LARGE DATASETS

To illustrate that OHPLnet is capable of analyzing large datasets, an 80% training sample from the Census 10 dataset was replicated 11 times to create a training dataset of 200,464 records. The first replication was used as is. For each of the other 10 replications, a small amount of gaussian "noise" was added to ensure that the algorithm did not achieve an artificially fast convergence to an optimal solution due to the fact that the dataset had 11 identical records for each record from the original set (doing so would effectively mean the algorithm was executing 11 iterations through the dataset, with each iteration over the derived dataset). The algorithm was set to complete a comparable number of iterations that resulted from the development of the regular sample classifier for the Census 10 dataset. When processing on the same computer, the algorithm scaled linearly in terms of time to converge, taking roughly 11 times the average, for each fold of the 5-fold cross validation, of the Census 10 dataset. In terms of classifier MZE and MAE, the results were virtually identical to the results for the 5-fold cross validation. This test effectively confirms the ability to scale DNNs to address very large ordinal classification problems [38].

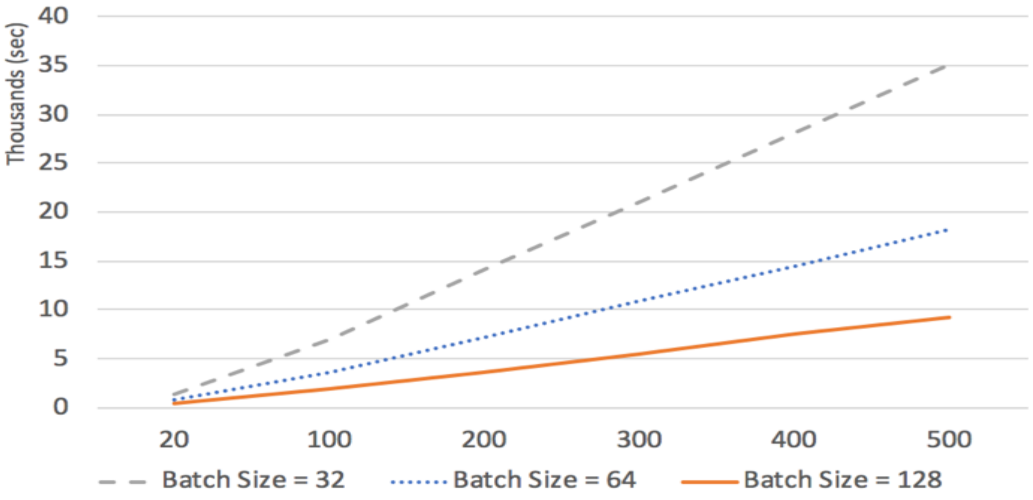


Figure 22 Time to Complete 500 Epochs by Number of Records (K records)

As can be seen in Figure 22, OHPLnet scales linearly with batch size, as would be expected from a DNN.

Chapter 7. EVOLUTION OF OHPLNET

7.1. MINI-BATCH OHPLNET

The initial work on OHPLnet provided a meaningful improvement over the best ordinal classifiers that are available today, but the methodology had some concerns that needed to be addressed. All of the benchmark data sets were small in size, so the initial algorithm design was able to use the entire dataset, for calculating the hyperplane centroids for each batch submission. Since the design for that part of the algorithm used straightforward matrix operations on structured data, the conceptual investigation could be conducted without concern for that the standard benchmark datasets that were too large to run in a single pass. To apply the original version to a very large dataset (e.g., one million records), algorithmic changes were going to be required (e.g., incorporate efficient matrix multiplication, that may be distributed to multiple computing nodes).

A primary example of a dataset that could not be analyzed using OHPLnet as it was originally constructed would be medical image files. To test the practical threshold of the OHPLnet design, a simple classification of medical images was examined. A fully dedicated computer, with an NVIDIA 1080 ti GPU that has 10 GB of GPU memory, could not process 2,000 medical images, through a Convolutional Neural Network, in a single batch. The practical limit for grey scale images that were 176 by 176-pixels was 500 images in a single batch.

In addition, the planning phase of the additional research included three different strategies for addressing the requirement to establish hyperplane centroids and allow them to update as the algorithm progressed, without the requirement of processing the entire dataset in a single execution and update of the DNN. Each strategy provides marginal success in the goal, but overall the algorithms did not perform as well as expected on one of the benchmark datasets (Red Wine Rating), indicating that there may be a unique challenge within that dataset, that needed to be overcome.

The first approach was the development of a simple mini-batch variant of the original OHPLnet. This approach was chosen as the first attempt due to the pervasive use of mini-batch processing within the Deep Learning community, but more importantly, that mini-batch based deep neural nets have a solid history of providing improved generalizability. As such, the approach may overcome the limitation of OHPLnet on the one dataset where it under performed, making it an algorithm that provide better generalization than OHPLnet. Developing the new variant required the restructuring of the HCL estimation to enable it to use small batches instead of the full dataset. These small batches may not include data from all class labels.

Many datasets are unbalanced in terms of the counts of records by class and frequently researchers must deal with highly unbalanced data. All of the benchmark datasets have some degree of imbalance in the class labels so they made appropriate test cases to ensure the new algorithm would properly address imbalances. To address this issue, the data labels were used to calculate an integer “distance” between adjacent hyperplane centroids that were present in the mini-batch, where the difference in label value is more than 1. For example, if the full dataset contained six distinct class labels, ‘0’-‘6’, but the mini-batch only contained records with values ‘2’ and ‘4’, then instead of requiring a minimum one-unit distance between the respective hyperplane centroids, the threshold was set to $(4-2)*1$.

OHPLnet Mini-Batch ALGORITHM: Iterative Algorithm

```
Hyper-Parameters: h – number of hidden layers
$l_h$ – number of nodes per layer
$\alpha$ – prioritization weight for HCL
lr – learning rate
m – HC margin
γ– point margin proportion
bs – batch size
Input:  Rescaled training data $\{(\boldsymbol{x}_i, y_i) | i=1,\ldots,n\}$
        Parameters h, $l_k$, $\alpha$, lr, $\{l_k = 1,\ldots h\}$
Begin:
        Randomize weight (W) and bias (b) in each DNN node
        While not converged do
                OHPLnet = 0, HPL = 0, HCL = 0
                Select mini-batch and one hot encode mini-batch labels
                Feed mini-batch through selected ANN
                        From ANN Output, Calculate HCL:
                                Adjacent Distance: Calculate difference in adjacent centroids
                                Adjacent Margin: Calculate adjacent label and multiply by m
                                Calculate HC error: Adjacent Distance – Adjacent Margin
                                Sum positive values, as HCL
                        From ANN Output, Calculate HPL:
                                Calculate distances from respective hyperplane centroids
                                Sum positive values, as HPL
                OHPLnet = HPL + $\alpha$*HCL
                Calculate Stochastic Gradient Descent (SGD)
                Update W and b via SGD and lr
        Repeat until training sample exhausted
        Check convergence
End: Output W and b
```

The resulting algorithm had highly mixed results when applied to the benchmark datasets. It performed extremely well on the Cars dataset that has a very high accuracy rate for virtually every ordinal classification methodology. On the more challenging data sets, the results were very sporadic. Not only was classification accuracy not as high as for OHPLnet, there was a very high degree of variability in results across repeated executions of the algorithm at the point that the algorithm reached its stopping criteria. Part of the issue is that the algorithm struggled to establish the proper ordering of the hyperplane centroids.

In addition, the total training error (loss) at the end of an iteration through the data, does not have the desired correlation with classification accuracy (both in terms of MZE or MAE values for the training set). Figure 23 is a simple plot of Training MZE and MAE versus total training error, for 20 consecutive model developments. The dataset is split into 80% training and 20% validation, with the same partitioning used for all 20 models. Note that this analysis did not include the model selection criterion that were used to benchmark the initial variant of OHPLnet. For that effort, a 100-sample test set was randomly selected from the training sample and used to select the best candidates for inclusions. The point of this effort is to demonstrate lack of relationship between the available model metrics and the scoring of a completely independent set of data. It does not require sophisticated statistical analysis to determine the lack of relationship between total error and classification error (see Figure 23).

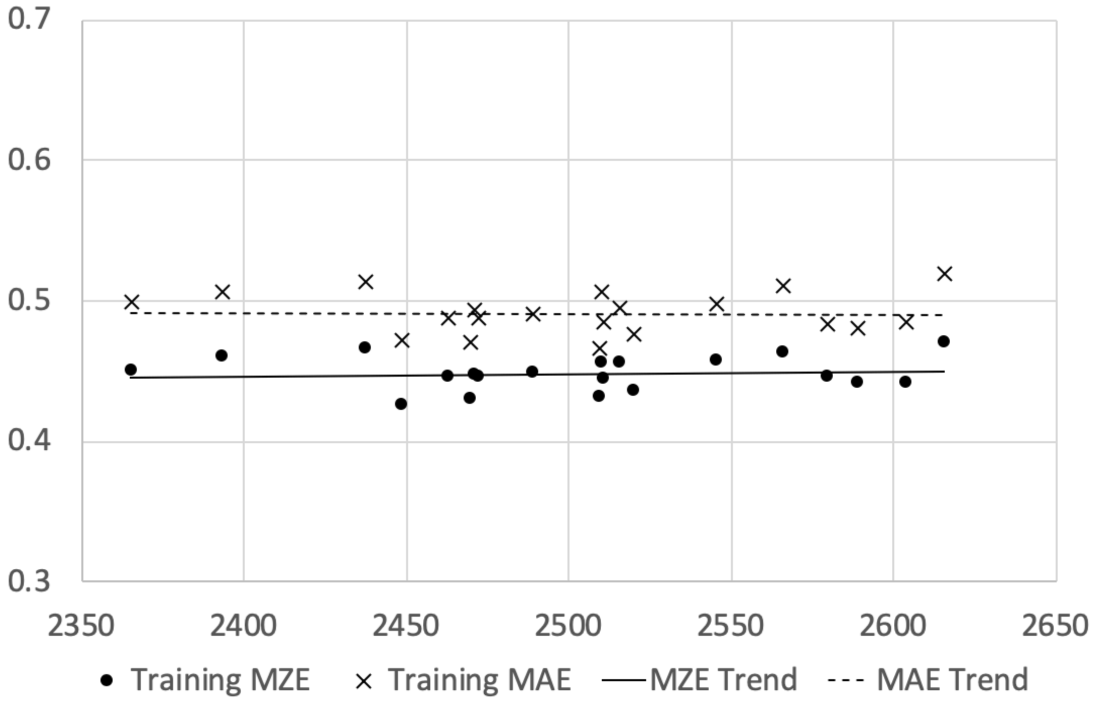


Figure 23 SWD Training Dataset: MZE and MAE vs Total Training Error

As would be expected, the same story holds true for the relationship between training error and validation set classification error. The slight increasing slope in the Validation set trend line in Figure 24, is not sufficient information to make the determination to use training error to choose the best model.

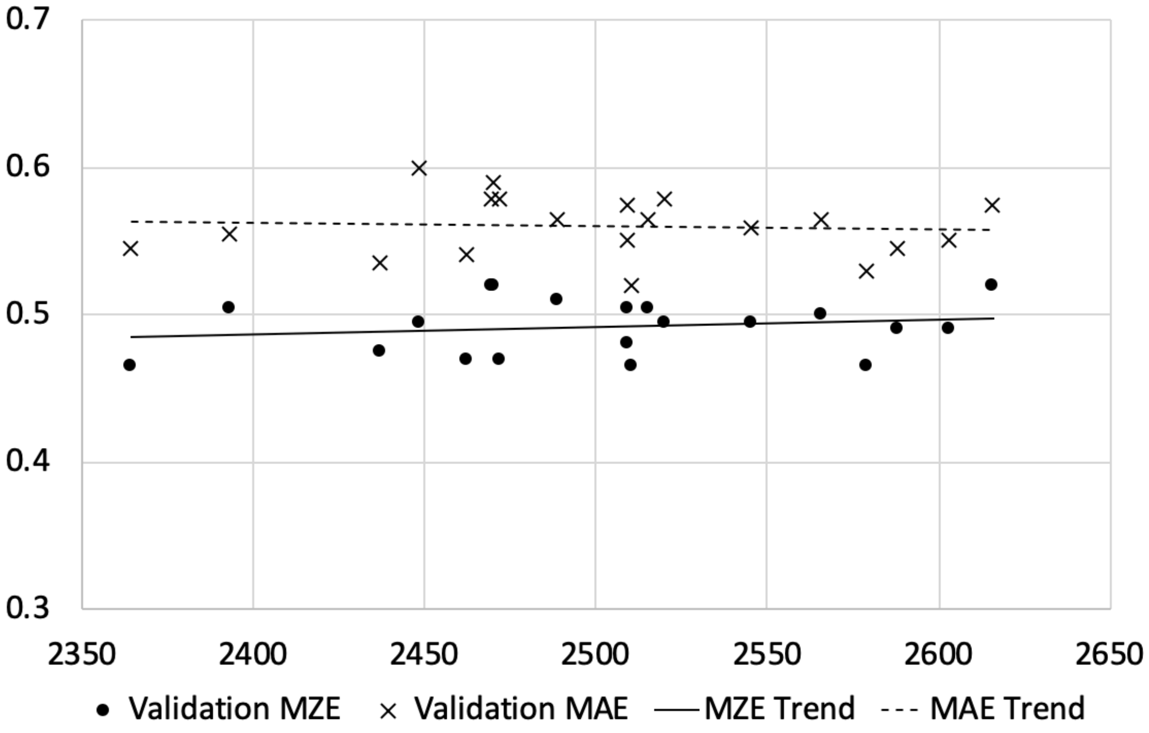


Figure 24 SWD Validation Dataset MZE and MAE vs Total Training Error

The other option is to use Training MAE or MZE to select the best classifier from a set of models that are generated from successive executions of the OHPLnet Mini-Batch algorithm. As can be seen in Figure 25 and Figure 26, the same lack of relationship holds true.

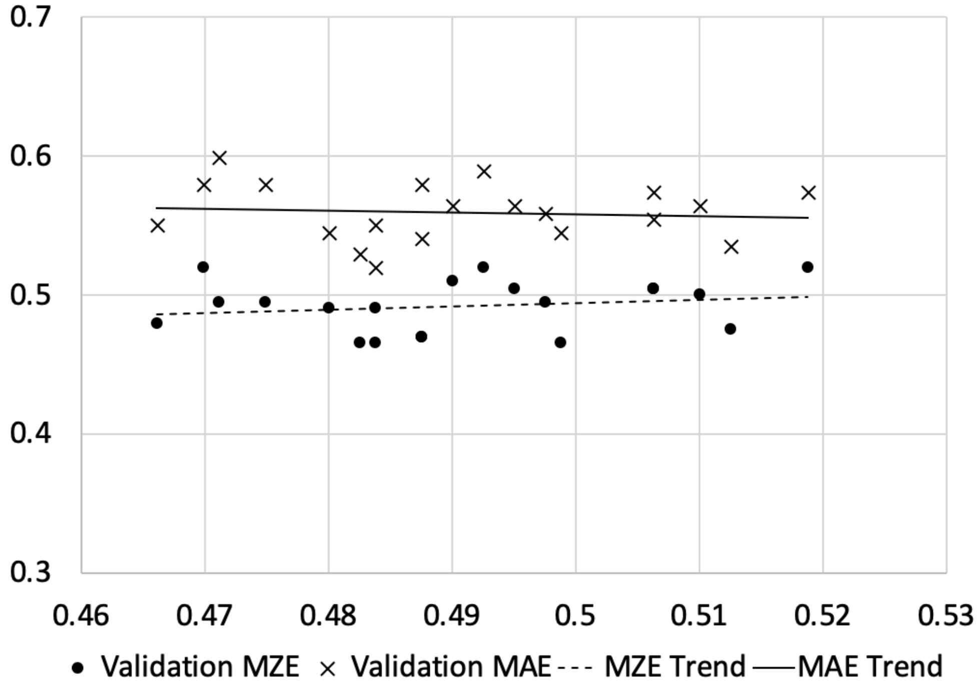


Figure 25 SWD Validation Dataset MZE and MAE vs Training Dataset MAE

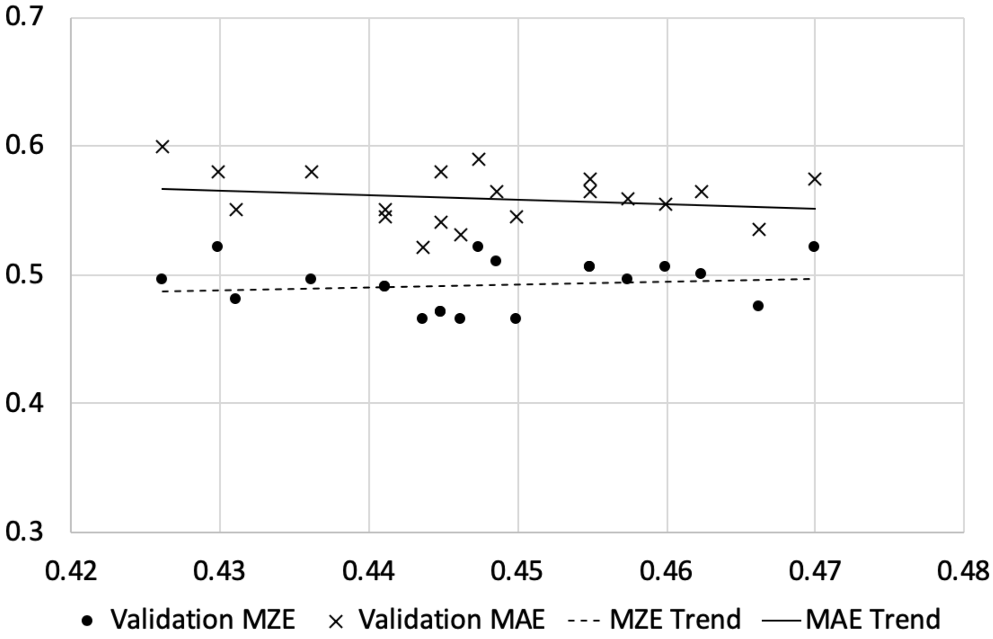


Figure 26 SWD Validation Dataset MZE and MAE vs Training Dataset MZE

It is highly plausible that the mini-batch variant was successful in uncovering local ordered relationships in the mapped space, but some of those local relationships negated each other, resulting in a model that fit the ordering of the most “localities” with the same or similar directional ordering. A key anecdotal point of evidence to supports this conclusion is the dramatic increase in total loss after a sort of the data.

7.2. TWO-STAGE OHPLNET

The initial version of OHPLnet was designed to create the proper ordering and spacing of the hyperplane centroids in stage one. In stage two, the algorithm fixes the hyperplane centroid values and passes the model weights and centroid values to a separate algorithm. In addition, the new algorithm provides results that have a strong relationship between the training set MAE values and the values for the validation set. Figure 27 plots MZE and MAE for the Census validation dataset. As can be seen there exists a strong relationship where a low training set MAE indicates a low validation set MAE. The MZE trend isn’t as steep, but the extremely low variation from the trend line suggests that the trend, while smaller is still reasonably reliable.

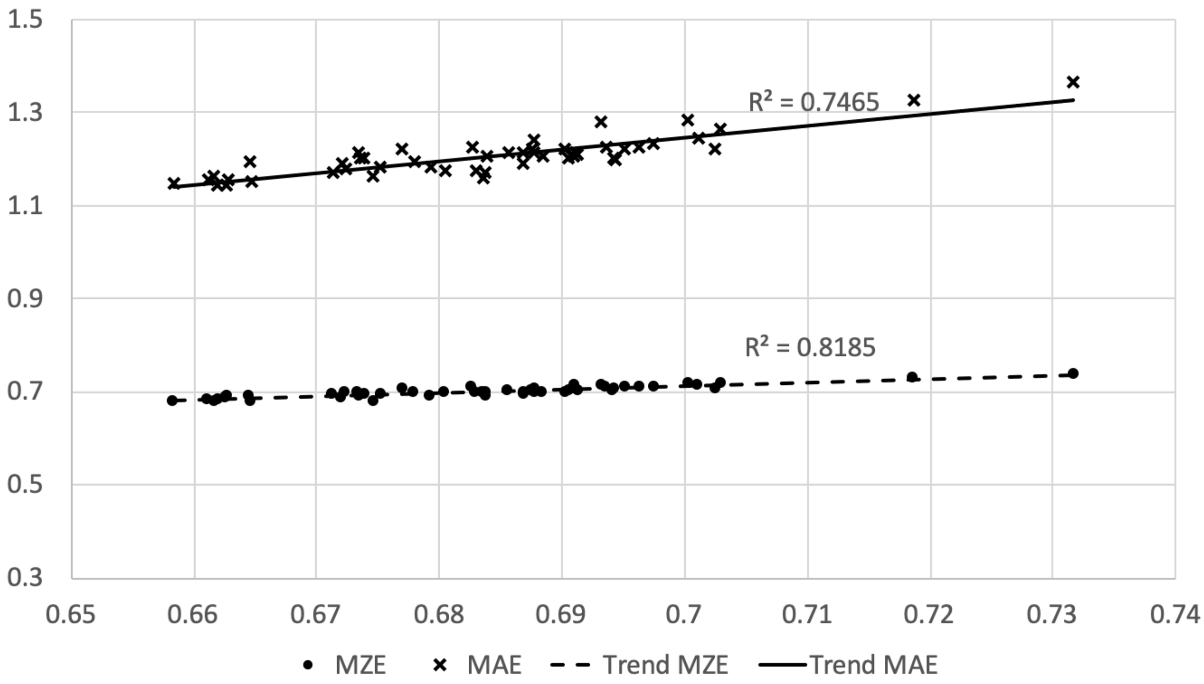


Figure 27 Validation Set MZE and MAE versus Training Set MAE

```
OHPLnet Two Stage ALGORITHM: Iterative Algorithm
Stage 1 – Hyperplane Centroid Ordering
Hyper-Parameters:
h – number of hidden layers
$l_h$ – number of nodes per layer
$\alpha$ – prioritization weight for HCL
lr – learning rate
m – HC margin
bs – batch size
Input:  Rescaled training data $\{(\boldsymbol{x}_i, y_i) | i=1,\ldots,n\}$
        Weights (to handle imbalanced labels)
        One hot encoded labels
        Parameters h, $l_k$, $\alpha$, lr, $\{l_k = 1,\ldots h\}$
Begin:
        Randomize weight (W) and bias (b) in each DNN node
        While ordered spacing less than 1
                OHPL = 0, HPL = 0, HCL = 0
                Select large batch
                        Calculate large batch output from network
                        Calculate HCL:
                                Adjacent Distance: Calculate difference in adjacent centroids
```

```
                    Adjacent Margin: Calculate adjacent label and multiply by m
                    Calculate HC error: Adjacent Distance – Adjacent Margin
                    Sum positive values, as HCL
        Repeat until training sample exhausted
        Check convergence
End: Output W and b

Stage 2 – Minimize Point Distance
Hyper-Parameters:
h – number of hidden layers
$l_h$ – number of nodes per layer
lr – learning rate
$\gamma$– point margin proportion
bs – batch size
Input:  Rescaled training data {($\boldsymbol{x}_i$,$y_i$ )|i=1,…,n}
        Weights (to handle imbalanced labels)
        One hot encoded labels
        Stage 1 model weights
        Hyperplane Centroids
        Parameters h, $l_k$, $\alpha$, lr, {$l_k$ = 1,…h}


            Select mini-batch (bs)
                Calculate mini-batch output from network
                Calculate distances from respective hyperplane centroids + $\gamma$
                Sum positive values, as HPL
            Calculate Stochastic Gradient Descent (SGD)
            Update W and b via SGD and lr
        Repeat until training sample exhausted
        Check convergence
End: Output W and b
```

Since the initial hyperplane centroid solution may not be optimal after a number of iterations of reducing point distance error, the algorithm needs to be able to update the hyperplane centroids, finding a potentially new solution that uses the current state as a starting point. The final two-stage version of OHPLnet was developed performs the same two-stage process, but also tracks model development performance. If performance does not improve within a prescribed number of iterations, the algorithm attempts to re-estimate the hyperplane centroids.

In current design of OHPLnet, the hyperplane centroids are estimated on the entire dataset, in a full batch processing, when possible. For very large datasets or large image files that cannot be processed in a single batch, a “maximum” batch strategy is employed. In experiments, the ordinal class ordering is established very quickly, but subsequent updates, when needed, will progress more slowly, taking a significantly larger number of epochs to reach a solution. For some datasets, this re-estimation of the hyperplane centroids required so may iterations that a processing limit was added. If the re-estimation does not occur in the allowed number of epochs, then the hyperplane centroids, that were in use prior to the attempted re-estimation are used. During experimentation, results suggest that reaching a point where hyperplane centroids cannot be re-estimated are indications that the algorithm has reached a local minimum and the condition may need to become one of the stopping criteria, for the algorithm.

7.3. OHPLALL

The simple mini-batch OHPLnet algorithm provides mixed result when applied to the same benchmark dataset found in Chapter 6 experimental results. In Chapter 10, OHPLall is used to predict the classification of medical images. The image sizes were very large (1 MB each) and could not be compressed without risking the loss of important detail, that is required for classification. In order to complete the analysis, a new mini-batch version of OHPLnet was needed. In addition, the original Mini-Batch OHPLnet required a sort of each mini-batch (labels and training data), before submitting the data for processing by the neural network. While this process could be repeated for the new algorithm, one goal for this research was to improve

OHPLnet by simplifying the algorithm and remove unnecessary processing steps. Sorting small batches does not require high compute time but running over 1,000 sorts per iteration through the data does have an impact. This new variant of OHPLnet is called OHPLall.

To provide a natural prioritization on the hyperplane centroid ordering the new algorithm changed the HCL loss component to compare all classes that were represent in the mini-batch to each the other classes. The margin must be appropriately adjusted to account for ordinal label differences that are greater than 1 (i.e., cases were the labels differ by more than 1). For each mini-batch, the new mathematical formula for HCL is found in equation (37).

$$HCL = \sum_{i<j} \max\left(HC_i - HC_j + (j - i) * \delta, 0\right) \qquad (37)$$

OHPLall ALGORITHM: Iterative Algorithm

```
Hyper-Parameters: h – number of hidden layers
$l_h$ – number of nodes per layer
$\alpha$ – prioritization weight for HCL
lr – learning rate
m – HC margin
γ– point margin proportion
bs – batch size
Input:  Rescaled training data {($\boldsymbol{x}_i$,$y_i$ )|i=1,…,n}
        Parameters h, $l_k$, $\alpha$, lr, {$l_k$ = 1,…h}
Begin:
        Randomize weight (W) and bias (b) in each DNN node
        While not converged do
                OHPL = 0, HPL = 0, HCL = 0
                Select mini-batch and one hot encode mini-batch labels
                Feed mini-batch through selected ANN
                        From ANN Output, Calculate HCL:
                                All HCL Distances: Calculate distances for all pairs of centroids
                                All HCLs Margin: Calculate label differences and multiply by m
                                Calculate HC error: Adjacent Margin – Adjacent Distance
                                Sum positive values, as HCL
```

```
            From ANN Output, Calculate HPL:
                Calculate distances from respective hyperplane centroids
                Sum positive values, as HPL
        OHPL = HPL + $\alpha$*HCL
        Calculate Stochastic Gradient Descent (SGD)
        Update W and b via SGD and lr
    Repeat until training sample exhausted
    Check convergence
End: Output W and b
```

7.4. Experimental Results for New Variants of OHPLnet

In testing the centroid ordering was attained with minimal priority weighting of hyperplane centroid loss components. The experimental results demonstrate that the new variants of OHPLnet, particularly OHPLall, perform well on the benchmark data sets that we examined in Chapter 6. Two-Stage OHPLnet and OHPLall perform well across the seven benchmark datasets. While Two-Stage OHPLnet consistently provides the top performance, OHPLall comes in at an admirable second place for 6 of the seven datasets.

Table 7 MZE Results for New OHPLnet versus other OHPL Base Algorithms

	OHPL	OHPLnet Mini-Batch	Two-Stage OHPL	OHPLall
CPU Small	0.542	0.518	0.573	**0.516**
Census 10	**0.646**	0.723*	0.701	0.681
Cars	0.024	0.012	**0.003**	0.014
Wine-Red	0.444	0.542	**0.358**	0.418
ERA	0.772	0.790	**0.709**	0.755
LEV	0.412	0.544	**0.362**	0.412
SWD	0.427	0.492	**0.371**	0.407

* - Five of the 20 scores for Census 10 would have been rejected, if the goal were focused on best performing model. Their training set MZE and MAE scores were essentially equal to random assignment. If those values are removed, the resulting MZE and MAE values would be 0.665 and 1.170, respectively.

The development of new OHPLnet variants lead to new algorithm, based on the same design principles with improved results. In addition, the new variants are more capable of analyzing very large datasets.

Table 8 MAE Results for New OHPLall versus other OHPLnet Base Algorithms

	OHPLnet	OHPLnet Mini-Batch	Two-Stage OHPLnet	OHPLall
CPU Small	0.763	0.701	0.814	0.709
Census 10	1.267	2.002*	1.207	1.199
Cars	0.024	0.012	0.003	0.014
Wine-Red	0.520	0.539	0.384	0.457
ERA	1.790	1.447	1.272	1.543
LEV	0.460	0.677	0.382	0.442
SWD	0.473	0.560	0.386	0.425

Chapter 8. OHPLNET ANALYSIS STRATEGIES

In addition to designing two new very capable OHPLnet algorithms, this research includes the development and assessment of sampling strategies that could be valuable approaches to analyzing extremely large datasets. Each of the three strategies uses a sampling process designed to reduce the compute cost calculating and updating the hyperplane centroids, by reducing the size of the data sample that is used to calculate and update the hyperplane centroids. These strategies should also provide an opportunity for significant improvements in processing time on very large datasets.

8.1. DOUBLE-BATCH SAMPLING STRATEGY

With the inclusion of a two-tiered batch selection framework, the algorithm for OHPLnet Double-Batch is a modification of the original OHPLnet work. The most significant limiting factor in applying the original OHPLnet to a very large data set (e.g., one million records or more) is the ability of the computer to process all of the data to establish the hyperplane centroids, in a single pass. In this strategy, large batches of records (e.g., 10,000 records each) are chosen, without replacement. The large batch is used to calculate the hyperplane centroids each time a mini-batch is processed. Mini-batches sampled, without replacement, from the large batch and submitted for processing. In essence, the large batch is treated as though it were the full training set for the processing. Subsequent large batches are submitted and processed until all data within the training data set are processed. If the large batch size (number of records) is set to that of the training set size then the algorithm the same as using OHPLnet.

OHPLnet Double Batch ALGORITHM: Iterative Algorithm

```
Hyper-Parameters:
h – number of hidden layers
$l_h$ – number of nodes per layer
$\alpha$ – prioritization weight for HCL
lr – learning rate
m – HC margin
$\gamma$– point margin proportion
bs – batch size
Input:  Rescaled training data $\{(\boldsymbol{x}_i, y_i)|i=1,\dots,n\}$
        Parameters h, $l_k$, $\alpha$, lr, $\{l_k = 1,\dots h\}$
Begin:
        Randomize weight (W) and bias (b) in each DNN node
        While not converged do
                OHPL = 0, HPL = 0, HCL = 0
                Select large batch and one hot encode large batch labels
                        Feed large batch through selected ANN
                                From ANN Output, Calculate HCL:
                                        Adjacent Distance: Calculate adjacent centroids distance
                                        Adjacent Margin: Calculate adjacent label, multiply by m
                                        Calculate HC error: Adjacent Distance – Adjacent Margin
                                        Sum positive values, as HCL
                        Select mini-batch (bs) within the large batch
                        Feed mini-batch through selected ANN
                                From ANN Output, Calculate HPL:
                                        Calculate distances from respective hyperplane centroids
                                        Sum positive values, as HPL
                        OHPL = HPL + $\alpha$*HCL
                        Calculate Stochastic Gradient Descent (SGD)
                        Update W and b via SGD and lr
        Repeat until training sample exhausted
        Check convergence
End: Output W and b
```

8.2. SINGLE STRATEFIED SAMPLING STRATEGY

The single stratified sampling variant employs a single stratified sample of the training set, at initialization of the algorithm. The same sample is used for all epochs (a single full iteration through the training set) within the execution to termination. It is assumed that with a sufficient sample size (e.g., 10,000 records per class), that the hyperplane centroid solution would be a sufficient representation of the full dataset. In employing a single sampling for the entire execution of the algorithm, there is a risk that a rare event occurs, and the sample is not truly representative of the entire data, which could lead to reduced or even poor generalizability of the algorithm. Since there is a single sampling event and the hyperplane centroids are estimated on a small sampling of the full dataset, relative to the full training set size. This speed consideration makes the development of this variant a worthwhile endeavor.

OHPLnet Single Stratified ALGORITHM: Iterative Algorithm

```
Hyper-Parameters:
h – number of hidden layers
$l_h$ – number of nodes per layer
$\alpha$ – prioritization weight for HCL
lr – learning rate
m – HC margin
γ– point margin proportion
bs – batch size
Input:  Rescaled training data $\{(\boldsymbol{x}_i, y_i) | i=1,…,n\}$
        Parameters h, $l_k$, $\alpha$, lr, $\{l_k = 1,…h\}$
Begin:
        Randomize weight (W) and bias (b) in each DNN node
        Select stratified batch and one hot encode batch labels
        OHPL = 0, HPL = 0, HCL = 0
        While not converged do
                Iterate through dataset
                        Feed stratified batch through selected ANN
                                From ANN Output, Calculate HCL:
                                        Calculate difference in adjacent centroids
                                        Calculate adjacent label and multiply by m
                                        Calculate HC error: Adjacent Distance – Adjacent Margin
                                        Sum positive values, as HCL
                        Select mini-batch (bs) from full training set
                        Feed mini-batch through selected ANN
                From ANN Output, Calculate HPL:
                    Calculate distances from respective hyperplane centroids
                    Sum positive values, as HPL
            OHPL = HPL + $\alpha$*HCL
            Calculate Stochastic Gradient Descent (SGD)
            Update W and b via SGD and lr
    Check convergence
End: Output W and b
```

8.3. EPOCH STRATIFIED SAMPLING STRATEGY

Epoch Stratified Sampling variant creates a new relatively large stratified sample of the training set at initialization of the algorithm (though not as large as the Single Stratified Sampling Strategy). A new stratified sample is created at the start of each epoch within the execution of the algorithm. Since the sampling is repeated, smaller strata sizes are used (e.g., 1,000 records per class). In doing so, unless the number of epochs is set very low, a larger number of records are used at some point in the model creation. The use of a larger percentage of the full training dataset should mitigate risk an undue influence from a rare sampling events that may include an unusual number of outliers. Like the Single Stratified Sampling Strategy, the hyperplane centroids are estimated on a small sampling of the full dataset, but the sampling changes with each epoch so over the course of the full execution of the algorithm the hyperplane centroids are based on a larger proportion of the data. While the sampling at the start of each epoch this provides allows for very large datasets without overwhelming the computing system.

OHPLnet Epoch Stratified ALGORITHM: Iterative Algorithm

```
Hyper-Parameters:
h – number of hidden layers
$l_h$ – number of nodes per layer
$\alpha$ – prioritization weight for HCL
lr – learning rate
m – HC margin
$\gamma$– point margin proportion
bs – batch size
Input:  Rescaled training data $\{(\boldsymbol{x}_i, y_i) | i=1,\dots,n\}$
        Parameters h, $l_k$, $\alpha$, lr, $\{l_k = 1,\dots h\}$
Begin:
        Randomize weight (W) and bias (b) in each DNN node
        While not converged do
        Select stratified batch and one hot encode batch labels at the start of the epoch
                OHPLnet = 0, HPL = 0, HCL = 0
                Iterate through dataset
                        Feed stratified batch through selected ANN
                                From ANN Output, Calculate HCL:
                                        Calculate difference in adjacent centroids
                                        Calculate adjacent label and multiply by m
                                        Calculate HC error: Adjacent Distance – Adjacent Margin
                                        Sum positive values, as HCL
                        Select mini-batch (bs)
                        Feed mini-batch through selected ANN
                                From ANN Output, Calculate HPL:
                                        Calculate distances from respective hyperplane centroids
                                        Sum positive values, as HPL
                        $OHPL = HPL + \alpha * HCL$
                        Calculate Stochastic Gradient Descent (SGD)
                        Update W and b via SGD and lr
        Check convergence
End: Output W and b
```

8.4. EXPERIMENTAL RESULTS FOR OHPLNET VARIANTS

Artificial neural networks are prone to having high variation in results, leading to a strategy of developing multiple models, which are validated against a validation sample that is selected from the training data (i.e., not the testing or validation sample that is used to test the performance of the final model that is selected) [50]. This can lead to a bit of a serendipitous approach to model development. OHPLnet shares some similarities with triplet loss, which uses relative position of similar and dissimilar samples to determine the error contribution of a triplet of points, in the mapped space. The process requires the use of "hard triplets" to optimize model performance. The identification of these hard triplets can make triplet loss challenge effectively use [51].

Similarly, OHPLnet is reliant on the identification of hyperplane centroids. Unlike triplet loss, identifying them is not based on a strategy of searching for specific data records to use in the analysis. The algorithm finds a mapped space where the hyperplane centroids exist, with the proper ordering and proper minimal spacing. At the same time, the algorithm is attempting to minimize the distances of individual points. The relative "push" to gain appropriate ordering and spacing for the hyperplane centroids is in direct conflict of the "pull" on the points to minimize distance from the average value for the point's class. This likely contributes to the high variability in results when OHPLnet is applied to some datasets.

Even with high initial weighting on the establishment of the hyperplane centroids (in some cases on order of 10^6), the results could have more variance than desired. This led to the development of a variable weighting of the hyperplane centroid loss, for early iterations of the algorithm, followed by decreased weighting, but at a level to maintain hyperplane centroid ordering and minimum distance. The variant with variable weighting had some success and was being used when the discovery that lead to OHPLall was uncovered.

The data in Table 9 illustrate this phenomenon. The same algorithm that produced unit minimum distances for all other datasets, included in this report as well as at least a dozen others, consistently fails to do so with the ERA dataset. The minimum distance of under 0.3 is actually lower than the point margin that was initially used in the component of the algorithm that focuses on a point's distance from its corresponding hyperplane centroid. Setting the margin for point distance to 0.1 or less does not change the result. More importantly, on an 800-record training set, the multiplier weight for hyperplane centroid distance was initially set to 1,000.

Table 9 ERA Dataset Double Batch Results for 5 Algorithm Executions

Training Set MZE	Training Set MAE	Validation Set MZE	Validation Set MAE	Minimum Hyperplane Centroid Distance
0.753	1.434	0.746	1.541	0.212
0.747	1.415	0.771	1.595	0.210
0.743	1.429	0.727	1.493	0.259
0.741	1.438	0.761	1.576	0.219
0.756	1.418	0.746	1.517	0.202

This finding presents a possible additional interpretation/application of OPHL base results. The ERA data set provides a scenario where extreme weighting is used to bias the loss almost exclusively on the ordering a minimum distance requirement for the hyperplane centroids. Some degree of minimizing point distance from the point’s corresponding hyperplane centroid, can be included. Otherwise, the algorithm may reach a solution where no points occur between hyperplane centroids, for the maximum and minimum classes, which would effectively turn an ordered prediction of three or more classes into a binary prediction. In the ERA dataset, the predictive features don’t have sufficient “information” or “signal” to provide the desired separation of classes (hyperplane centroids). In a case like this, the results indicate that relative to the available set of predictive features, the classes may not be as distinct as the labels imply. In some scenarios, there may even be an argument for combining the two classes into one. An example of this last scenario is explored in Chapter 10.

Through the evolution of the OHPLnet algorithms to the eventual development of OHPLall, algorithm accuracy in terms of MZE and MAE values for scored holdout samples did not necessarily change, but algorithm consistency did. The for the publication of *Ordinal Hyperplane Loss*, the benchmark testing relied on a separate 100 record test sample to identify versions of models that were generated using the same training set. For some datasets, like CPU Small and Census 10, the resulting model scores were relatively stable. For others like the Red Wine dataset, mean MZE was 0.444 with a standard deviation for 20 models of almost .11 (25% of the MZE score). The benchmark process selected the 20 best models from batches of five executions of OHPLnet , to arrive at the score, reported in the paper.

Table 10 and Table 11 report mean MZE and MAE, respectively, for 20 iterations of each algorithm for the five variants of OHPLnet and OHPLall, across the seven benchmark datasets.. Unlike the process for the benchmarking, that was done for *Ordinal Hyperplane Loss*, these results are reported without employing a selection process to choose a best model for a set of models. OHPLnet has consistently high performance across all seven datasets.

Table 10 MZE Results for OHPL/OHPLall versus Analysis Strategies Using OHPL

	OHPL	OHPLnet Mini-Batch	Two-Stage OHPLnet	OHPLall	OHPLnet Double Batch	OHPLnet Epoch Stratified	OHPLnet Single Stratified
CPU Small	0.542	0.518	0.573	0.516	0.544	0.535	0.534
Census 10	0.646	0.723*	0.701	0681	0.775	0.668	0.678
Cars	0.024	0.012	0.003	0.014	0.002	0.007	0.011
Wine-Red	0.444	0.542	0.358	0.418	0.531	0.446	0.459
ERA	0.772	0.790	0.709	0.755	0.769	0.750	0.758
LEV	0.412	0.544	0.362	0.412	0.558	0.417	0.422
SWD	0.427	0.492	0.371	0.407	0.536	0.442	0.451

* - Five of the 20 scores for Census 10 would have been rejected, if the goal were focused on best performing model. Their training set MZE and MAE scores were essentially equal to random assignment. If those values are removed, the resulting MZE and MAE values would be 0.665 and 1.170, respectively.

Table 11 MAE Results for OHPLall versus other OHPLnet Base Algorithms

	OHPL	OHPL Mini-Batch	Two-Stage OHPL	OHPLall	OHPLnet Double Batch	OHPLnet Epoch Stratified	OHPLnet Single Stratified
CPU Small	0.763	0.701	0.814	0.709	0.768	0.744	0.757
Census 10	1.267	2.002*	1.207	1.199	1.779	1.199	1.162
Cars	0.024	0.012	0.003	0.014	0.002	0.007	0.011
Wine-Red	0.520	0.539	0.384	0.457	0.636	0.512	0.533
ERA	1.790	1.447	1.272	1.543	1.660	1.564	1.619
LEV	0.460	0.677	0.382	0.442	0.706	0.456	0.451
SWD	0.473	0.560	0.386	0.425	0.644	0.493	0.498

Like Epoch Stratified OHPLnet, Two-Stage OHPLnet has excellent consistency in performance, as demonstrated by the low standard deviation across the results of 20 iterations of the algorithms (see Table 12 and Table 13). The major difference in the performance of Epoch Stratified OHPLnet and Two-Stage OHPLnet comes in one of the experiments, discussed later. In that experiment, OHPLnet Stratified failed to consistently establish the hyperplane ordering for a very challenging dataset. It is this failure that lead to the change in design that differentiates Two-Stage OHPLnet from the other algorithms.

Table 12 Standard Deviations of MZE

	OHPL	OHPL Mini-Batch	Two-Stage OHPL	OHPLall	OHPLnet Double Batch	OHPLnet Epoch Stratified	OHPLnet Single Stratified
CPU Small	0.007	0.006	0.006	0.009	0.010	0.006	0.008
Census 10	0.009	0.102	0.012	0.007	0.018	0.007	0.007
Cars	0.024	0.008	0.005	0.016	0.002	0.005	0.007
Wine-Red	0.0967	0.102	0.011	0.021	0.035	0.003	0.032
ERA	0.036	0.021	0.016	0.009	0.017	0.010	0.014
LEV	0.032	0.113	0.016	0.031	0.065	0.014	0.015
SWD	0.018	0.184	0.011	0.018	0.069	0.016	0.016

Table 13 Standard Deviations of MAE

	OHPL	OHPL Mini-Batch	OHPLall	Two-Stage OHPL	OHPLnet Double Batch	OHPLnet Epoch Stratified	OHPLnet Single Stratified
CPU Small	0.017	0.010	0.018	0.006	0.015	0.008	0.015
Census 10	0.036	0.144	0.018	0.043	0.147	0.018	0.024
Cars	0.024	0.008	0.016	0.005	0.002	0.005	0.007
Wine-Red	0.071	0.156	0.024	0.012	0.056	0.003	0.041
ERA	0.211	0.069	0.041	0.011	0.099	0.023	0.035
LEV	0.047	0.113	0.034	0.018	0.122	0.014	0.016
SWD	0.019	0.203	0.022	0.011	0.128	0.025	0.022

As reported in Table 14 Sample Strategy for Double Batch and Stratified Batches, the ERA dataset has the lowest number of records per class, at just over 111 records/class. Four other datasets have under 500 records/class. For these five datasets, the maximum number of records per class in the stratified data is set to ½ of their average number of records per class. For the other two, the maximum strata size is set to 10% of the average number of records per class, rounded up to the nearest integer multiple of 100. In each iteration, a new stratified sampling is generated. This strategy essentially ensures that a single poor sampling does not impact algorithm performance in terms of accuracy of prediction.

To overcome this issue, multiple, independent sorted copies of the data were appended into a larger file and the algorithm fit a model without sorting. While this revised process provided stability, the solution was not much of an improvement over the regular mini-batch. It simply provided a more stable output, but rarely provided exceptional results. In addition, the replication data strategy is counter to the goal of analyzing large data sets in a minimal amount of time. While OHPLnet Mini-Batch is not an abject failure, it is not a desirable solution to address the inclusion of hyperplane centroids for analysis of large datasets.

Closely related to the OHPLnet Mini-Batch strategy is the OHPLnet Double Batch strategy. The basic premise of this approach is to first select a relatively large sample from the available data (5,000-10,000 records depending on the number of labels), then run mini-batches of data in the large sampling to reduce distance to the hyperplane centroids that are determined by the larger batch. As documented in Table 1, only two of the datasets exceed 2,000 records in size. For the seven analyzed datasets, the "large" batches were set to be ½ of the number of records in the training sets. Each of the two datasets that exceed 2,000 records, also exceed 8,000 records, meaning the training sets would exceed 6,400 records. For the two largest datasets, CPU Small and Census 10, the large batch size was set to 1,024 records (see Table 10).

Table 14 Sample Strategy for Double Batch and Stratified Batches

	Number of Records	Number of Classes	Average # Records per Class	Large Batch Size	Maximum Strata Size
CPU Small	8,192	10	819.2	1,024	100
Census 10	22,784	10	2,278.4	1,024	300
Cars	1,728	4	432	692	200
Wine-Red	1,599	6	266.5	640	100
ERA	1,000	9	111.1	400	50
LEV	1,000	5	200	400	100
SWD	1,000	4	250	400	125

OHPLnet Double Batch is variant of OHPLnet that produced the top results for the Cars dataset (reported later, in Table 10 and Table 11). Though the results for the Cars dataset is not the compelling motivation for OHPLnet or OHPLall, the vast majority of algorithms that were tested by Gutierrez, et. al., performed very well on this dataset [9], so it any methodology that would be considered to be a top performer would likely perform very well on it, too.

It also performed well on the other datasets with better consistency in results, but the occasional very poor result could still occur. The OHPLnet Double Batch algorithm provides a more stable diagnostic result for selecting the 'best' performing model from a set of trained models, using the same training set. Across all seven benchmark datasets, summing training set MZE and MAE provide a good metric for selecting the best performing model out of a set of models. To illustrate, from 20 models, that were created using the same training and validation samples of the SWD data set. Summed training MZE and MAE are rescaled to a minimum value of zero and a maximum value of one for data. Similarly, validation set MZE, validation set MAE and summed MZE and MAE, from the validation set were rescaled. The regression trend line for each of the validation metrics, regressed on the training set metric are virtually the same line and several of the data points are almost perfectly overlapped (see Figure 28). While OHPLnet Double Batch is not the winner across the board, in terms of model accuracy, the ability to confidently select a model that should generalize to other data, without the use of a separate test sample, is very compelling.

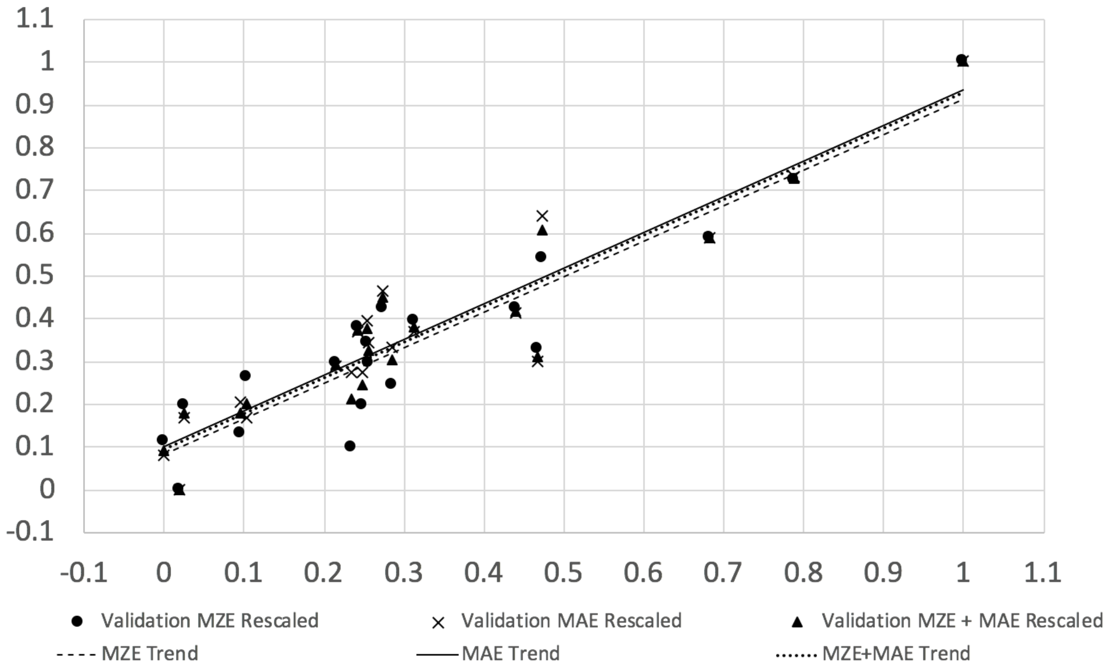


Figure 28 SWD Dataset Validation MAE, MZE and MAE + MZE vs Training set MAE + MZE.

All values rescaled to a [0,1] interval.

Chapter 9. APPLICATION: CLASSIFICATION OF MEDICAL IMAGES

The American Cancer Society reports that in 2017 over 300,000 people in the United States were diagnosed with breast cancer and over 40,000 people died from the disease [52]. Due to improvements in treatment and early detection, the death rates that are attributed to breast cancer have declined 39% from 1989 to 2015.

Radiologists use the first six categories of the seven-point BI-RADS (Breast Imaging Reporting and Data System) rating system to classify mammography images. The seventh category is used for images that are of breast images with a known malignancy, that was confirmed via a biopsy [53]. The zero category is used for images where classification is uncertain and additional information is required. Categories one through six are a sequence of ordinal classes.

Table 15 BI-RADS Category Scale [53]

Category	Definition
0	Additional imaging evaluation and/or comparison to prior mammograms is needed.
1	Negative
2	Benign (non-cancerous) finding
3	Probably benign finding – Follow-up in a short time frame is suggested
4	Suspicious abnormality – Biopsy should be considered
5	Highly suggestive of malignancy – Appropriate action should be taken
6	Known biopsy-proven malignancy – Appropriate action should be taken

The Cancer Imaging Archive (TCIA) is a public access database of curated medical images [54] [55], with accompanying annotations:

Table 15: CBIS-DDSM Annotations [54] [56]

Annotation	Relation to Scan Event	Definition/Values
Side	Prior to	Left or right breast
View	Prior to	CC - craniocaudal MLO - mediolateral oblique
Density Rating	Prior to	Breast density rating
Abnormality Type	After	Calcification (2 annotations) – Type and distribution Mass (2 annotations) – shape and margin
Assessment	After	BI-RADS rating (0, 2-5)
Pathology	After Image Assessment	Benign Without Callback Benign Malignant

The CBIS-DDSM (Curated Breast Imaging Subset of DDSM) is found within the TCIA and contains over 2,600 images that are selected by a trained mammographer [55, 56]. The data were released in 1997. Even though they are more than 20 years old, they remain a valid source of curated mammography data for researchers [57]. Several studies analyzing the CBIS-DDSM data, have been published in the past year or two [58, 59].

In a very recently published paper (Feb 10, 2019), Agarwal et. al. used pretrain VGG16, ResNet50 and InceptionV3 to produce classifier models to detect abnormal masses, in mammograph images. The work examined confirmed abnormal masses found in the CBIS-DDSM database. Their research reported a true positive rate classification of 0.98 when using the classifier that was built using the InceptionV3 algorithm [60]. After training the classifier, they scored 224 by 224-pixel image patches from the full mammography data in INbreast database [61] to determine the generalizability of their models [60].

Shen et. al. built a successful classifier that detected malignant abnormalities within full mammography scans. They trained their classifier on reduced size mammography images from the CBIS-DDSM database, with an average size reduction of approximately .29 (average reported image size of approximately 4,000 by 3,000 pixels reduced to 1,152 by 896 pixels). Patches of 224 by 224 pixels were generated from the reduced images. They used a sampling strategy to select malignant, benign, and background patches for training their classifier using Resnet50 and VGG16. The classifier was then used to [58] classify images from the INbreast database.

Li et. al. used Radiographic Texture Analysis combined with CNN based classifier, examining only the craniocaudal images, in an effort to predict the presence of unilateral breast cancer [62]. Their Radiographic Texture Analysis employed a stepwise feature selection using Support Vector Machines. Their research demonstrated meaningful improvement, in prediction AUC using the combination of the two classifiers, over-using either classifier on its own.

The goal of this research is to analyze mammography images from the CBIS-DDSM database that have been classified by radiologists, to build an image classification model predicts BI-RADS categories two through five. The CBIS-DDSM database contains images that in the DICOM format and classified to have suspicious masses or calcifications with provided labels.

There are three types of images, that differ by size:

1. Full mammography images
2. Images that are cropped for standardization for use in computer-aided diagnosis and detection (CADx and CADe, respectively). Regions of interest are at the centroid of the image [57].
3. Regions of Interest (ROI) images are smaller images that focus more directly on the abnormality [57].

The calcification image data provides almost 230 more images than the mass abnormality data. Due to the larger sample of data, the calcification data are examined. Based on examination of the image data, the cropped images are relatively large images for the purposes of this type of classification. Row pixel counts are in the 4,000 to 7,000 range and column pixel counts of 2,000 to 4,500. Attempting to use these images, with the available computer resources would require that the images be reduce by a factor of seven. Malignant abnormalities tend to differ from benign abnormalities in their “mathematical geometry” [57]. Compressing images, to 1/30th of their current image size would likely remove distinguishing characteristics that would be critical in differentiating BI-RADS class.

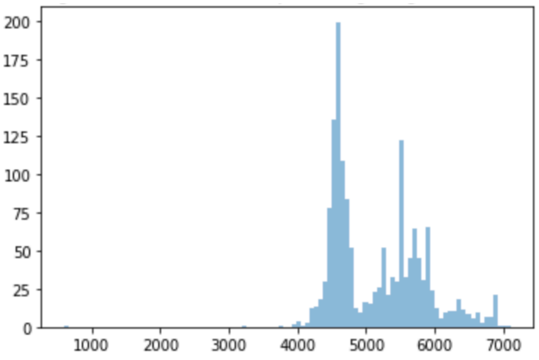


Figure 29 Distribution of Row Pixel Count for Cropped Calcification Images

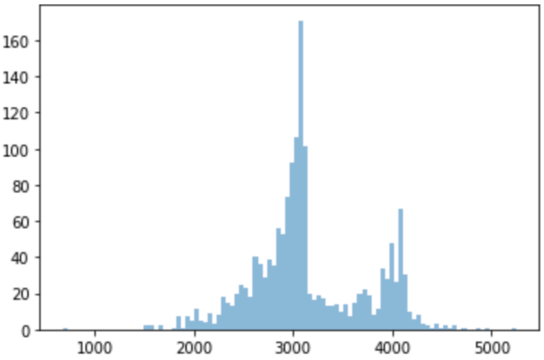


Figure 30 Distribution of Column Pixel Count for Cropped Calcification Images

The Region of Interest scans vary in size from 70 pixels to 3,000 pixels but are heavily skewed to under 1,000 pixels per side. Due to their size these images are better choices for analysis, on a desktop or laptop. Many of the images would still need to be resized to a smaller pixel count but on an order of ¼ (or smaller) the size of the original image.

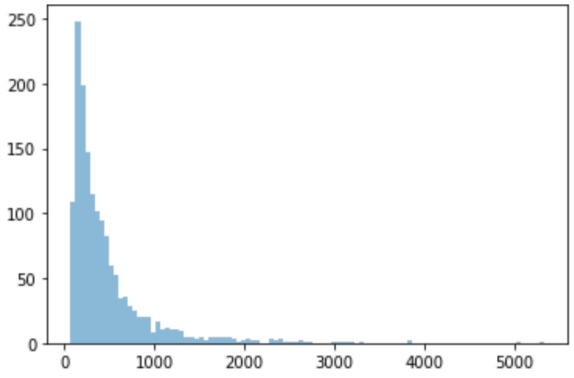


Figure 31 Distribution of Row Pixel Count for ROI Calcification Images

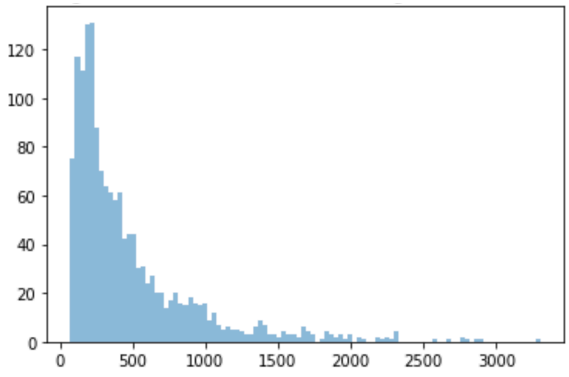


Figure 32 Distribution of Column Pixel Count for ROI Calcification Images

In examining the histograms in of pixel counts for rows and columns, there exists a clear break at 900 pixels for rows, but a meaningful number of additional images are in the 1,000-1,200-row pixel range. The column pixel counts for the images with row pixel counts below 900 is skewed, similar to the full set, but the “skinny” part of the tail begins at roughly 1,200 pixels. Setting the column limit to equal the row, would exclude a relatively small number of images.

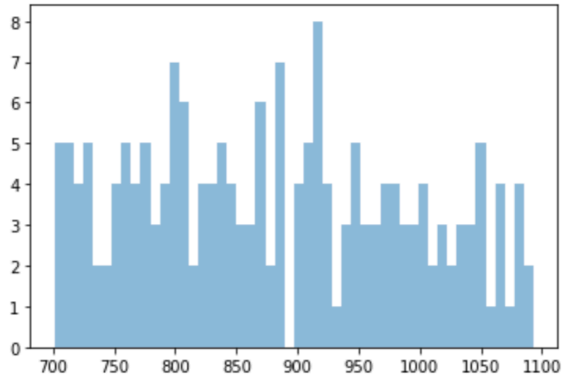


Figure 33 Distribution of row pixel count for ROI Calcification Images with row pixel count between 700 and 1,100.

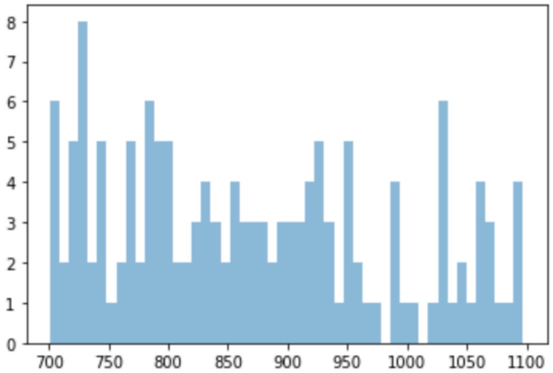


Figure 34 Distribution of column pixel count for ROI Calcification Images with column pixel count between 700 and 1,100.

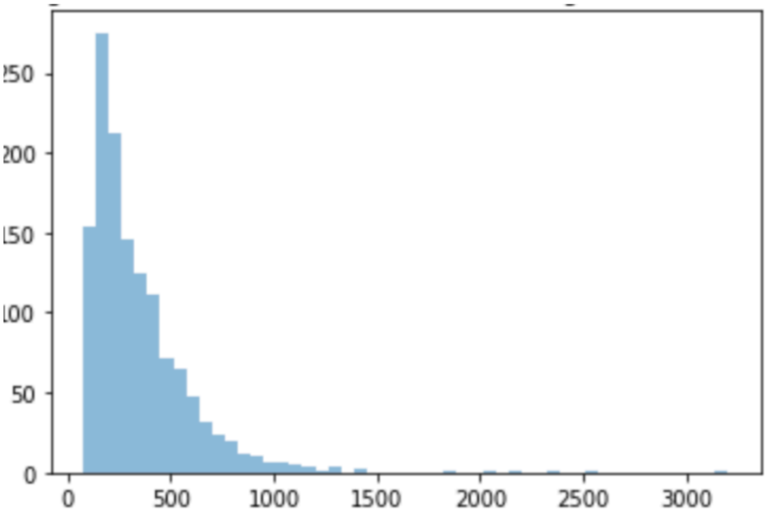


Figure 35 Distribution of column pixel count for ROI Calcification Images

with row pixel count less than 900

Figure 36-Figure 39 are a set of four sample images, select from the Calcification ROI Training set. In some cases, a single calcification is relatively large while in others there are multiple very small calcifications or may even be a cluster of calcifications. Across the images a variety of background texture/noise can be seen.

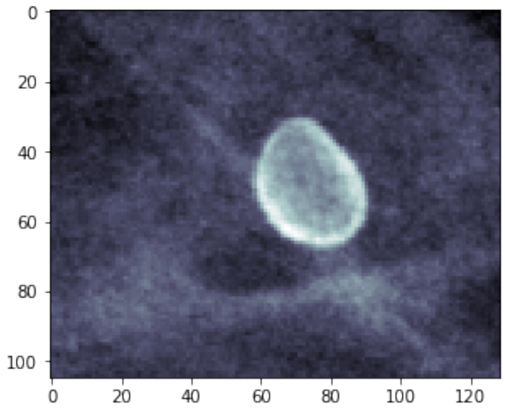


Figure 36 Sample Mammography Image

View: Craniocaudal
Distribution: ‘NA’
Pathology: Benign Without Callback

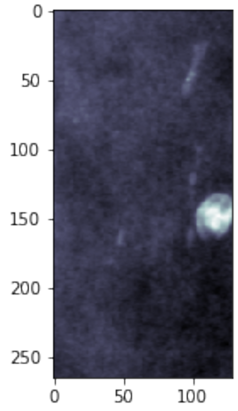


Figure 37 Sample Mammography Image

View: Craniocaudal
Distribution: Clustered
Pathology: MALIGNANT

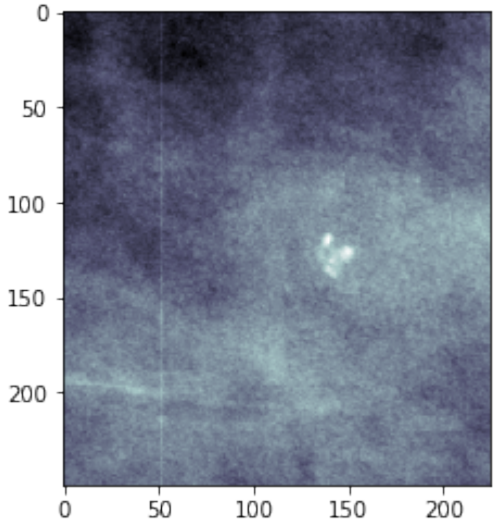


Figure 38 Sample Mammography Image

View: Mediolateral Oblique (MLO)
Distribution: Clustered
Pathology: Benign

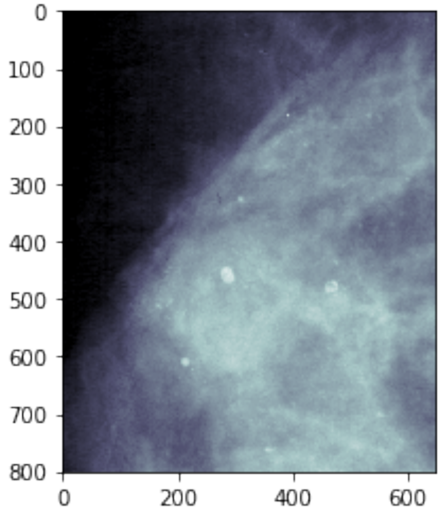


Figure 39 Sample Mammography Image

View: Mediolateral Oblique (MLO)
Distribution: Segmental
Pathology: Malignant

There are 71 images in the selected records of the combined in the Training and Test datasets, whose mammograms received a BI-RADS classification, indicating that additional tests and information was needed to be able to make an assessment. For the purposes of this analysis, these records are problematic, since the value ‘0’ is an inappropriate label for this ordered scale. Removing these records from the training set removes potential excessive error being introduced into training.

These records also provide an opportunity for additional assessment of a classification. While the exact BI-RADS class in the ordered scale is unknown, the pathology finding for the patient is available. Almost 50% of the images with classification of ‘0’ were determined to have a malignant tumor (Table 16). If the model provides an appropriate ordered classification based on the abnormal classification the records with the higher predicted BI-RADS class would have a larger number of patients who had a malignancy.

Table 16 Zero Assessment Patient Key Statistics

Pathology Finding	Number of Images
BENIGN	38
MALIGNANT	33
Total Records	71

In training set, the patients with a rating of ‘3’ have a higher incidence rate of a malignancy. While the difference is not statistically significant at the 90% level, given the fact that BI-RADS ‘4’ is supposed to be a higher risk group than BI-RADS ‘3’, this is a surprising finding. The high incidence of malignancy in these two groups also poses a bit of an issue when attempting to train an order classifier. Based on the literature, the patients with a malignancy could be appropriately reclassified as BI-RADS category ‘6’, though developing a classifier that predicted values of ‘6’, for use as a secondary assessment for a radiologist, would be inappropriate since that class is reserved for confirmed malignancies from biopsies.

Using the hypothesis that the patients with malignancy were appropriately labelled, based on a trained radiologist’s assessment, there may be an opportunity to classify the images of patients with a malignancy in separate classes, based on their initial BI-RADS rating. If successful, the finding would demonstrate that there are discernable differences in the images that can be detected by a classifier that is estimated using a convolutional neural network.

In addition to providing are preselected training set of images, The Cancer Image Archive provides an independent set of test images. For the purposes of this research, 306 of the images qualified for inclusion (Table 17). The ‘3’ and ‘4’ class malignancy rates do not adhere to the percentages that are expected of effective BI-RADS ratings of mammography images.

Table 17 Image Counts by BI-RADS Rating

BI-RADS Rating	Training Set Percentage Images with Malignancy	Number of Images	Test Set Percentage Images with Malignancy	Number of Images
2	0.2%	473	0.0%	71
3	35.5%	84	69.6%	23
4	39.9%	742	36.9%	176
5	98.5%	124	100.0%	36
Total	29.3%	1,423	38.2%	306

For benchmarking purposes, the algorithms that were used in Chapter 4 cannot be used for processing images. The Ordinal Regression algorithm developed by Cheng et. al will be used since it is able to CNNs to analyze images [12].

While CNNs provide a powerful methodology for analyzing images, image classification requires a very significant amount of data processing. Unless large, powerful computing systems are available, algorithms that solve image classification problems must use very small batches. For this research, the images were processed on two different computers. The first had a Nvidia GTX 1080 ti GPU, with 10 GB of memory. The second had a Nvidia RTX 2060 GPU with 6 GB of memory. Both GPUs provide significant processing improvement over using a CPU.

With the exception of the Mini-Batch variant, all OHPLnet variants, including the two stage variants that identify the hyperplane centroids prior to executing the minimum point distance, require 30 or more samples per class, within the batch, to provide reasonably stable hyperplanes. The computational system requirement for analyzing images using CNNs are impacted by more than simple image size. Each filter that is applied within a layer resulting in a “channel” that is essentially an altered version of the image. Large CNNs that analyze massive image databases (e.g., ImageNet) use hundreds of filters per layer, resulting in hundreds of channels. This process effectively multiplies a single “data point” (image) hundreds of times. This same process is done for large numbers of layers, again resulting in a multiplicative effect on the size of the data being processed. System resource capacity can quickly be exceeded even with relatively small network architectures.

For the purposes of the analysis of mammograms, the images were first compressed to expedite processing and to allow for appropriate identification of hyperplane centroids. The first attempt used 1,200x1,200 pixel or smaller images that were compressed to 256x256 pixels. Images for under-represented classes were over sampled to create a balanced training dataset with 742 images for each class. Attempts to create a classifier were modestly successful, with training set accuracy in the 30% to 32% range (versus 25% for random assignment). The lower memory in the Nvidia RTX 2060 GPU resulted in a system constraint in processing the images with a maximum of 64 per batch. The classifiers that were developed using this level of image compression performed poorly.

The degree of compression was then reduced resulting in 512x512 pixel images. While processing time increased and the maximum batch size limit decreased to 24 per batch, results for this compression level weren't much better than for the 256x256 pixels per image compression. More importantly, the results for this compression level had a high degree of variation in resulting model performance (MZE values between 0.40 and 0.70).

The calcification spots on the images are quite small. As mentioned earlier, radiologists examine the nature of the edges of the calcifications to assess BI-RADS level. While compression of images may have worked for other researchers who focused on predicting a binary outcome, that was based on an objective biopsy result, the experimental results from the compression of images indicate that classifying the images into four somewhat subjective classes based on human interpretation of the images, isn't appropriate. These less than desirable results may be an indication that compression of almost any level removes the fine details that could be important in determining BI-RADS rating class. The research proceeded using uncompressed images. The images that are more than 1,024 pixels on either side are cropped at the edges. Any image with a side that has fewer than 1,024 pixels are padded with zeros, which is consistent with the padding that CNN algorithms use. The larger size results in a 12 images per batch limitation on for the CNNs that show early promise in performance testing.

The 12 images per batch limitation required the use of the Mini-Batch OHPLnet variant. In the initial work, the variant that was outlined in Section 7.1 struggled to provide proper ordering of the hyperplane centroids for some datasets and results suffered if the data were sorted during processing. In the application of Mini-Batch OHPLnet when the ordering was not achieved, a single pair of adjacent hyperplane centroids were “inverted” (in the wrong order). If the algorithm compared these hyperplane centroids to all others, the error cost, in terms of the total loss would be significantly higher, so numerically they would have had higher “priority,” in the batch update. This batch size limitation lead to the development of the OHPLall.

Several dozen Convolutional Neural Network architectures were tested with the 1,024x1,024-pixel images. While several architectures performed comparably, to each other, an algorithm with ten Convolutional layers and four DNN hidden layers (see Figure 40) performed well for both OHPLall and Ordinal Regression.

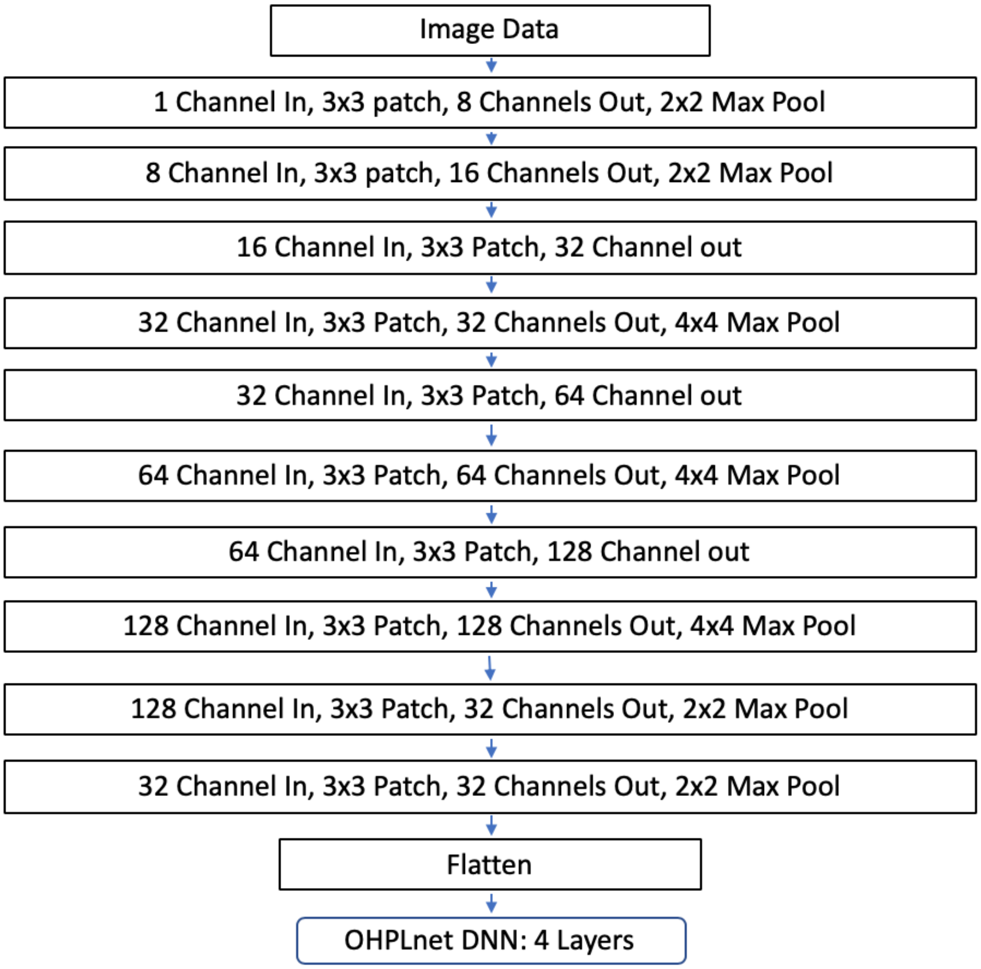


Image Data
1 Channel In, 3x3 patch, 8 Channels Out, 2x2 Max Pool
8 Channel In, 3x3 patch, 16 Channels Out, 2x2 Max Pool
16 Channel In, 3x3 Patch, 32 Channel out
32 Channel In, 3x3 Patch, 32 Channels Out, 4x4 Max Pool
32 Channel In, 3x3 Patch, 64 Channel out
64 Channel In, 3x3 Patch, 64 Channels Out, 4x4 Max Pool
64 Channel In, 3x3 Patch, 128 Channel out
128 Channel In, 3x3 Patch, 128 Channels Out, 4x4 Max Pool
128 Channel In, 3x3 Patch, 32 Channels Out, 2x2 Max Pool
32 Channel In, 3x3 Patch, 32 Channels Out, 2x2 Max Pool
Flatten
OHPLnet DNN: 4 Layers

Figure 40 Convolutional Neural Network Architecture

In evaluating model performance, the ability for a classifier to address this issue and return a distribution of malignancy rates that are more consistent with the BI-RADS definitions may be more important than actual accuracy and mean absolute error performance.

Twenty OHPLall models were generated using both OHPLall and Ordinal Regression. For OHPLall, the mean batch training error, for an epoch is a reasonable metric to use for as a stopping criteria. As can be seen, in Figure 41, mean batch error values that are below 0.5 results in low test set MZE and MAE. While higher mean batch error values may have low test set MZE and MAE, they may also have higher than desired test set MZE and MAE values.

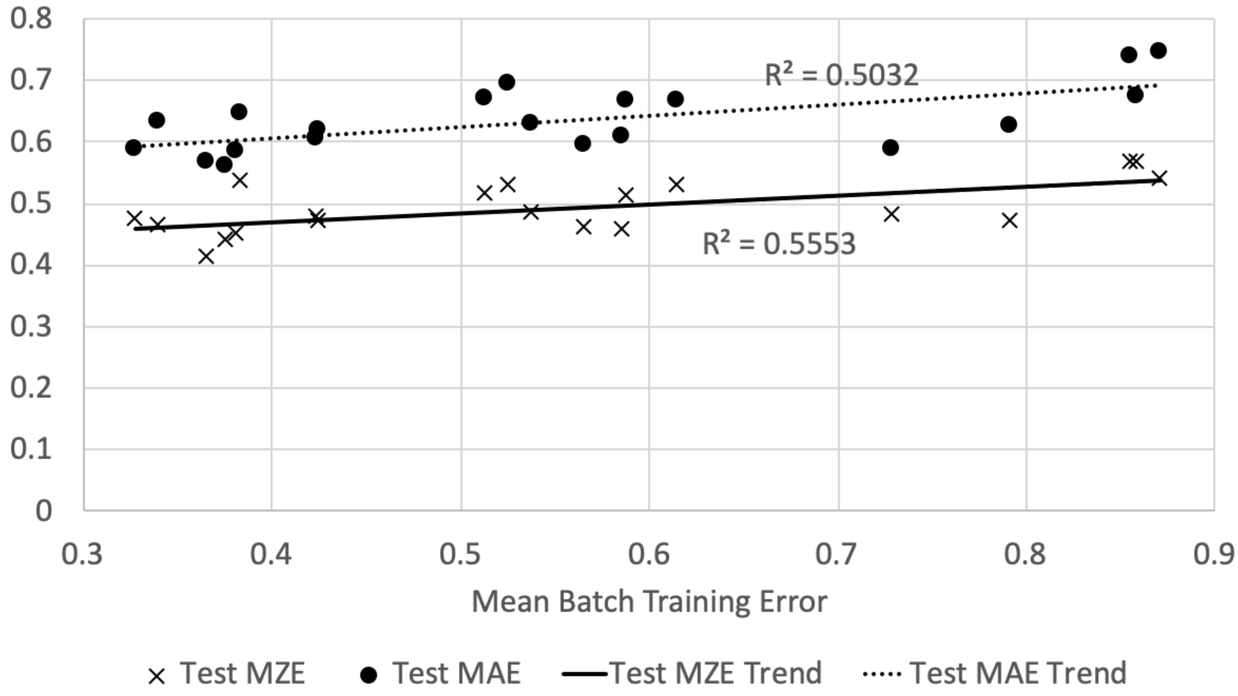


Figure 41 Training Data Mean Batch Error

From the results in the test set it is clear that classifying mammography images into the somewhat subjective BI-RADS classes is a particularly challenging task (see Table 18). Ordinal Regression MZE and MAE are 25% and 43% higher, respectively, than OHPLall, on the mean values of 20 executions of each algorithm (see Table 18). In addition, the MAE values for Ordinal Regression had double the standard deviation for MAE as OHPLall.

Table 18 OHPLall vs Ordinal Regression MAE and MZE Results

Algorithm	Metric	MZE	MAE
OHPLall	Mean	**0.473**	**0.612**
	Std Dev	**0.033**	**0.046**
Ordinal Regression	Mean	0.595	0.877
	Std Dev	0.041	0.099

In addition to assessing standard model performance metrics, it is also worthwhile to assess class predictions relative to biopsy results for the calcifications. For this evaluation, a single well performing model for each algorithm is examined. Table 19 reports the MAE and MZE for the selected models. If a model "struggles" to properly classify records within a given BIRAD rating, it is likely to be desirable for the errors to occur in the lower rating values and perform better in the higher ratings leading to early treatment for a malignancy. Both models perform poorly on BI-RADS '3' and '5' rated images. From the table it is clear to see that Ordinal Regression does a very good job, with BI-RADS '2' rated records, but performs poorly, relative to OHPLall in the other three classes (to the point that MAE for OHPLall is roughly equal to MZE for Ordinal Regression). As mentioned earlier, these two metrics may not be the best assessment of model quality.

Table 19 Rating Level Assessment for a High Performing OHPLall Model vs A High Performing Ordinal Regression Model

BI-RADS	OHPLall MZE	OHPLall MAE	Ord Reg MZE	Ord Reg MAE
2	0.408	0.732	0.211	0.338
3	0.696	0.739	0.739	0.826
4	0.324	0.386	0.574	0.767
5	0.750	0.944	0.889	1.417
Total	0.422	0.559	0.539	0.748

Per the BI-RADS definitions it is expected that malignancy rates would increase with BI-RADS score. The algorithm that produces models that best meet this expectation would provide higher quality predictions. Ordinal Regression predicted a significant shift in BI-RADS rating, towards the low end of the scale, resulting in very good MZE and MAE values for the '2' class, but poor results for the other classes. In addition, images classified as a '5' by OHPLall have over three times the Malignancy Rate (percent of images that were ultimately classified as malignant) as Ordinal Regression. Early identification of malignancy is critical in treating breast cancer, so this skew towards lower values versus OHPLall is less desirable for a model that is intended to be used as a diagnostic tool.

Table 20 Detailed Results for a High Performing Ordinal Regression Model

BI-RADS*	Actual Malignant Counts	OHPLall Malignant Counts	Ord Reg Malignant Counts	Actual Malignancy Rate	OHPLall Malignancy Rate	Ord Reg Malignancy Rate
2	0	7	44	0.0%	12.5%	42.7%
3	16	28	29	69.6%	48.3%	35.8%
4	65	69	40	36.9%	38.8%	37.4%
5	36	13	4	100.0%	92.9%	26.7%
Total	117	117	117	38.2%	38.2%	38.2%

* - Value for reported BI-RADS rating source, per column heading

The image database also contained a number of images with a BI-RADS classification of '0'. This class is designated as "Additional imaging evaluation and/or comparison to prior mammograms is needed". While a specific rating value is not available, the models can be assessed based on the malignancy rates for the predicted classes. As was the case for the test dataset, relative to OHPLall, Ordinal Regression shifts cases to the lower end of the rating scale. This skew towards the lowest available BI-RADS class includes a shift of nine malignant cases, to the ‘2’ class, giving this Ordinal Regression a higher malignancy rate than the rates for the other three classes. OHPLall classifies two malignant cases into class ‘2’. OHPLall classifies over 2/3 malignant cases into classes ‘4’ and ‘5’, while Ordinal Regression classifies just over half of the malignant cases into class ‘4’ and no malignant cases into class ‘5’. The OHPLall results are more consistent with the overall definitions of the BI-RADS measurement system.

Table 21 Results For ‘0’ Rated Cases

BI-RADS	OHPLnet Counts	OHPLnet: Malignant Counts	Ordinal Regression Counts	Ordinal Regression Malignant Counts
2	2	2	15	9
3	14	8	15	7
4	42	16	40	17
5	13	7	1	0
Total	71	33	71	33

For the classification of the available mammography images into BI-RADS rating, a Convolutional Neural Network that uses OHPLall loss provides better results than a Convolutional Neural Networks that use Ordinal Regression. Not only does it provide better overall results, but in the critical secondary assessment OHPLall works well in predicting images that have a malignancy into higher BI-RADS classes.

Chapter 10. APPLICATION: MULTI-CLASS SENTIMENT ANALYZER

In late 2003 Frederick Reichheld originally proposed Net Promoter in a famous Harvard Business Review article [63]. In the intervening years, Net Promoter Score (NPS) became a widely used client feedback system to assess overall perception of a company's products and services.

The basis of Net Promoter measurement systems is a survey program that captures responses from a company's customers who are asked to estimate their likelihood of recommending the company, its products, or its services to a friend or colleague [64]. The responses are given on a 10 or 11-point scale ('1'-'10' or '0'-'10'), with '10' being "extremely likely" and the lowest value being "extremely unlikely." The values are recoded into a 3-point semantic scale (see Table 22) [64]:

Table 22 Net Promoter Value to Semantic Label Recode

Response Value	Semantic Label
'9'-'10':	Promoter
'7'-'8':	Passive
'0'-'6':	Detractor

The Net Promoter Score is calculated by subtracting the percentage of respondents who are Detractors from the percentage who are Promoters, to create a metric that has a scale of -100 to 100 [64]. Many companies use a variety of customer touchpoints for their NPS measurement system [65]. Some companies are so committed to their NPS program that they are embedding the system into all facets of their business. In addition to being surveyed on overall company performance, customers are asked to rate specific product and service offerings. Processes that are internal to the company (e.g., helpdesks that employees use for workstation issues) are also measuring NPS [65]. The ability to assess likely Net Promoter Score in text, in social media (Twitter, Facebook, etc.), blogs (e.g., technical review sites), and customer surveys, provides multiple additional touchpoints for the company to assess. A text based NPS metric may even be viable for rating competitor's Net Promoter Scores.

Current state of the art sentiment analyzers use numerical word embeddings to represent the words in the analyzed text. A word embedding is a numerical vector representation of words, where each word in the corpus (aggregate body of text) is associated a unique vector [66]. Words that are close in semantic/contextual meaning have similar vector values.

Not only do Net Promoter responses provide valuable insights to the company, to provide qualitative interpretation to response scores but they also provide an opportunity to develop a "sentiment" like classifier for short text messages or responses. Companies like Uber, Facebook, and Twitter employ very sophisticated sentiment analysis process to better understand customer attitudes [67]. For a company that is making Net Promoter Score a core KPI (Key Performance Indicator), the ability to correctly and efficiently classify social media comments and survey responses without the need for manual evaluation may open new areas of business analysis and measurement that are not currently available.

The survey database for the IT company that provided the NPS data for analysis has over 60,000 completed surveys with short responses that are linked to a respondents NPS score. This data includes responses from customers across the globe. In the cases where responses are provided in a language other than English, Watson Language Translator was used to provide

English versions of the response. It should be noted that the data have not gone through a secondary screening to validate the class labels. Respondents are free to enter any text response that they choose. In some cases, respondents offer reasons as to why the rating was a ‘9’ and not a ‘10’, so the response may appear to be negative or similar to negative comments that correspond to low rating values. In other cases, a low rating may be provided, and the respondent decided to focus on a positive attribute of a call (e.g., the agent was polite and worked hard to resolve a problem) which may be very similar (or even identical) to a response for a very high rating. In other cases, the respondent may list the technical components/processes that resulted in a problem and a positive or negative sentiment is not clear. In spite of these inconsistencies in some records, there is sufficient data for the algorithm to be able to discern patterns that are associated with each response class, but the pure accuracy likely would not reach that of well documented binary sentiment analyzers that can be found on-line.

This real-world application is a test of OHPLall in analyzing text. While this is a test of verbatim responses of no more than 500 characters, other text applications may be quite large, so this test will use the OHPLall to assess performance. An example application on a very large corpus might be the development of letter grade classifier predicting grade on a corpus of 1,000+ term papers that are each 25 pages in length. Assuming 300 words per page, a single document would have approximately 7,500 words per document (double spaced). If one of the larger word to vector embeddings, with vector length of 100, is used the size of a single document would be almost 100,000 values. While the data used for this application isn’t this large, the text is a valid assessment of real data that is produced by real activities in businesses.

An appropriate benchmark algorithm must be able to take advantage of the power of word embeddings as well as the ability to analyze word sequences that is offered by RNNs and CNNs. An Ordinal Regression with at least one Gated Recurrent Neural Network layer is a good choice that meets these requirements.

If sufficient data sample is available, an analyst may choose to develop a unique word embedding for the corpus of documents that they are analyzing. For smaller projects, particularly those that have a large number of words that have the same meaning as a common body of documents, the analyst may choose to use a word to vector database like GloVe [68] or Word2Vec [69]. These databases provide pretrained word embeddings for 400,000 to several million English words. In developing the word embedding databases, a very large corpus of documents (e.g., Wikipedia) is analyzed to identify word-word “co-occurrences” (frequencies at which two words occur adjacent to each other in the text).

In generating the GloVe database, Stanford researchers identified the 400,000 most common English words and developed a word-word occurrence matrix. To generate the embedding vectors, the log of the values in the word-word occurrence matrix is decomposed resulting in a unique 50-dimensional representation of each word [70]. Two words that frequently occur with the same set of thirds words will have very similar vector representations.

A researcher may also choose to employ a transfer learning approach by using data from one of the word embedding databases as the initial embedding in a deep learning model and enable the model to update the embedding values to maximize the predictive abilities of the classifier. This strategy may be particularly useful when analyzing a corpus of documents from a single topical area where word meanings are different that more common meanings (e.g., the use of the word “default” for software settings versus for credit accounts).

Since word sequences provide important context for the sentiment or attitude of the message, the words in a document are converted to sequences of numerical vectors. These vectors can then be analyzed using either a Recurrent Neural Network or by employing a Convolutional Neural Network that uses one-dimensional filters. Training of the network proceeds as with any other DNN.

The goal of this sentiment analysis effort is to simulate a real-world application of OHPLall to develop a sentiment classifier based on raw survey data. Text records that are single word values and have no expected association with the responder’s attitudes towards the company (e.g., 20 occurrences of the single word “on”) and responses that do not contain any actual English words (e.g., four occurrences of the digit ‘1’) are removed, leaving 60,593 records available for analysis. The data were split into Training, Test, and Validation sets using proportions of 80:10:10. Due to the unbalanced nature of the dataset, random sampling was employed to ensure that all eleven response classes were represented within each sample (see Table 23).

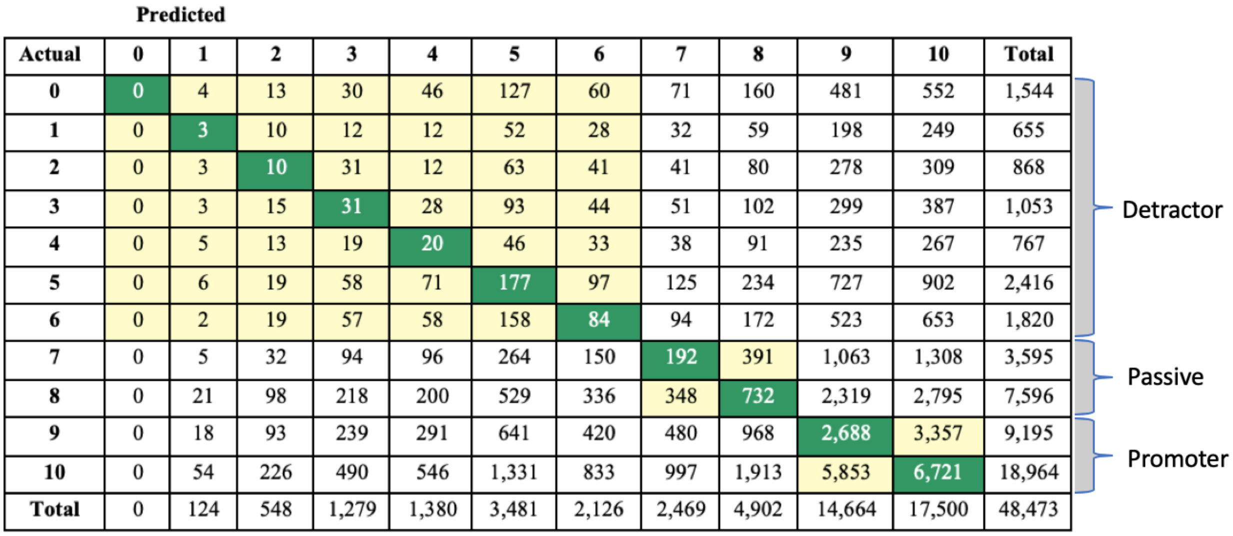
Table 23 NPS Sentiment Analysis Sample Counts by Response Class

Response Class	Training Set Counts	Test Set Counts	Validation Set Counts	Percentage of Sample
0	1,544	193	193	3.2%
1	655	82	82	1.4%
2	868	109	109	1.8%
3	1,053	132	132	2.2%
4	767	96	96	1.6%
5	2,416	302	302	5.0%
6	1,820	227	227	3.7%
7	3,595	449	449	7.4%
8	7,596	950	950	15.7%
9	9,195	1,149	1,149	19.0%
10	18,964	2,371	2,371	39.1%

As is the case with virtually all predictive model development methodologies, deep neural networks struggle to provide desired classification when built on highly unbalanced datasets where the frequency of records for one or two classes are significantly higher than the others. In the case of the available NPS data, class ‘10’ represents 39% of the available data. In addition, class ‘9’ represents an additional 19% of the data (see Table 23). In application, the eleven-point NPS scale is recoded into three classes where classes ‘9’ and ‘10’ are combined. In this final three class version, the highest class, represents 58% of the data (see Table 25).

Unbalance datasets can lead to model results that are not only unusable but have the potential to provide incorrect insights leading to incorrect decisions. A simple Ordinal Regression model was created on the unbalanced training set for this research. The results for the training set provide a good illustration of the problems that may occur by analyzing a highly unbalance dataset, without addressing the imbalance. In Table 24, the dark cells provide the counts for the records that were properly classified, in the eleven-class case. The lighter shaded cells along with the dark shaded cells provide the counts for the three-class grouping of the response values. The predictions skew heavily in the direction of the two largest classes (‘9’ and ‘10’), with the predictions of ‘9’ occurring 50% more frequently than their actual incidence rate in the dataset. In addition, no records are predicted to have a prediction of ‘0’. The model predicts the ‘Detractor’ class (shaded upper left 6 cell by 6 cell section) at a rate that is similar to that found in the actual responses, but the ‘Promoter’ ratings would be 15% higher than actual if the model predictions are used. These results would tend to give the company a false read on how many of their customers were highly satisfied with their products and services and unjustly skew the NPS metric in the positive direction.

Table 24 Confusion Matrix: Counts for Actual versus Predicted Classes



	Predicted											
Actual	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
0	0	4	13	30	46	127	60	71	160	481	552	1,544
1	0	3	10	12	12	52	28	32	59	198	249	655
2	0	3	10	31	12	63	41	41	80	278	309	868
3	0	3	15	31	28	93	44	51	102	299	387	1,053
4	0	5	13	19	20	46	33	38	91	235	267	767
5	0	6	19	58	71	177	97	125	234	727	902	2,416
6	0	2	19	57	58	158	84	94	172	523	653	1,820
7	0	5	32	94	96	264	150	192	391	1,063	1,308	3,595
8	0	21	98	218	200	529	336	348	732	2,319	2,795	7,596
9	0	18	93	239	291	641	420	480	968	2,688	3,357	9,195
10	0	54	226	490	546	1,331	833	997	1,913	5,853	6,721	18,964
Total	0	124	548	1,279	1,380	3,481	2,126	2,469	4,902	14,664	17,500	48,473

As a standard practice weighting or oversampling of low frequency classes is employed to address this issue. In the case of NPS it may be inappropriate to employ either solution to the point that all eleven classes have equal representation when developing a model. Since the data will ultimately be recategorized into three classes, it is more appropriate to use the proportions for the three-class version of the data, to address the imbalance (see Table 25).

Table 25 NPS Three Class Counts by Class

Semantic Response	Response Class	Training Set Counts	Test Set Counts	Validation Set Counts	Percentage of Sample
Detractor	‘0’	9,123	1,141	1,141	19%
Passive	‘1’	11,191	1,399	1,399	23%
Promoter	‘2’	28,159	3,520	3,520	58%

A simple oversampling strategy based on the frequency counts for the three-class semantic scale is employed. Differentiation within the classes in the three-class grouping is not an important result for the company. Therefore, no additional oversampling is performed within the three-class groupings of the 11-class responses. It is important for very low scores (e.g., 0 and 1) to be correctly classified as ‘0’ on the three-class scale whenever possible. To accommodate this requirement, an eleven-class model is specified instead of simply using the three-class labels for model development.

OHPLall is used within a Gated Recurrent Neural Network (GRNN). Gated Recurrent Neural Networks were created to address the vanishing/exploding gradient problem that may occur in RNNs. A GRNN differs from an RNN in their basic nodes. In the RNN, information from a prior node is combined with the new input for the node both are subject to multiplication by weights and summarized before a nonlinear function is applied. In the GRNN, the GRU (Gated Recurrent Unit) receives the same inputs but two “gates” within the node impact behavior. The first gate decides what information from the prior hidden state (i.e., the memory from the prior state) is accepted into the node. The other determines what information from within the current node is passed to the next node [34]. The activation functions within these gates behave as on-off switches but are actually continuous functions, so the weights within them can update the same way that weights in a simple DNN node is updated [34]. GRNNs are useful in sentiment analysis, since they can selectively carry memory though the recurrent network so the specific positioning of key words (e.g. adjacent or two-words apart versus 20 words apart) does not critically hinder classification.

For the current NPS classification problem a GRU layer with 128 outputs is used. The GRU layer is fed by the word embedding layer that is initialized with word embedding vectors from the GloVe database. Words that occur in the texts but are not found in the word embedding database are initialized with random values. Since the content of the text is specific to computers and the IT domain, the word embeddings are further trained to optimize their contribution within the full network. Two additional standard DNN layers are included before the standard scalar output for OHPLall (see Figure 42).

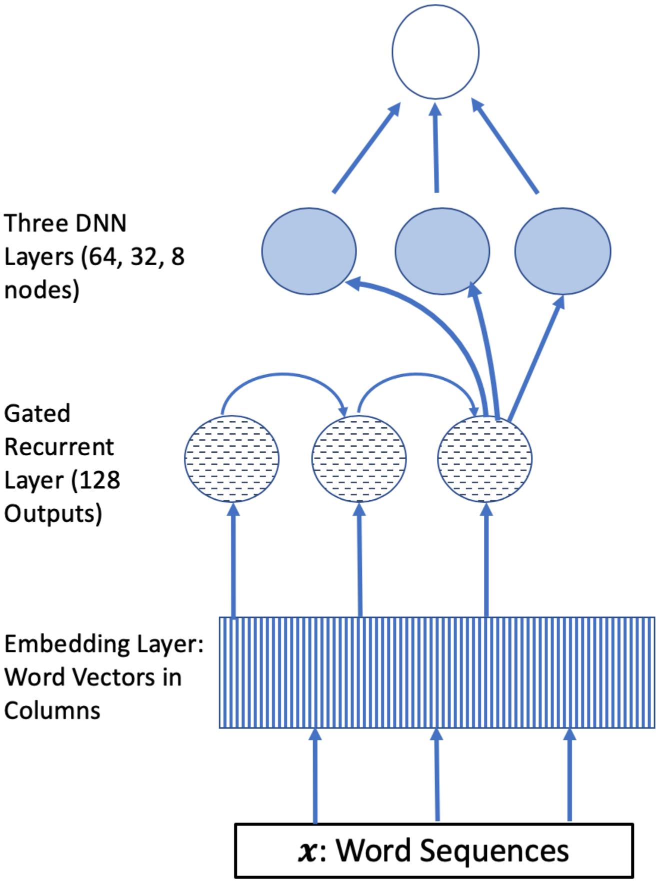

Three DNN Layers (64, 32, 8 nodes)
Gated Recurrent Layer (128 Outputs)
Embedding Layer: Word Vectors in Columns
x: Word Sequences

Figure 42 NPS GRNN Network Graph

The neural network architectures are optimized by comparing model solutions training sample results against the results for the validation sample to arrive at a final network architecture. From that point, twenty models were estimated using the winning architecture for each algorithm. In addition to predicting the eleven ordinal classes, the predicted classes are reclassified into the three-class solution. Mean Zero One Error (MZE) and Mean Absolute Error (MAE) are calculated for each. In Table 26, bold values denote the best performance on the metric between OHPLall and Ordinal Regression.

When NPS data were analyzed relative to customer behaviors and financial relationship with the company, OHPLall provided strong evidence that it would be appropriate to combine some classes. Similarly, in the models that were developed for the NPS Sentiment Classifier, the ‘9’ and ‘10’ classes weren’t separated by the desired one-unit distance. Distances were consistently in the 0.20 range. This result is additional confirmation that the grouping of the ‘9’ and ‘10’ response classes is an appropriate aggregation of those classes.

Table 26 NPS Sentiment Analyzer Results For 20 Iterations of Each Algorithm

		Three Class MZE	Three Class MAE	Eleven Class MZE	Eleven Class MAE
OHPLall	Mean	**0.320**	**0.370**	**0.652**	**1.281**
	Std Dev	**0.006**	**0.007**	0.014	0.032
Ordinal Regression	Mean	0.360	0.406	0.724	1.352
	Std Dev	0.007	0.007	**0.010**	**0.011**
Mean Comparison	Percent Difference	13%	10%	11%	6%

While neither algorithm achieves a stellar classification accuracy or mean error, OHPLall outperforms Ordinal Regression in both MAE and MZE for the eleven-class model. The performance differences are statistically significant at the 95% confidence level. The performance advantage of the eleven-class model in terms of MAE carries over to result in an even stronger performance in the three-class recode. As would be expected the performance difference for the three-class recode of the predictions is also statistically significant.

Low accuracy rates for both methodologies is in part due to the inconsistencies in rating versus the content of verbatim responses. Table 27 provides a sampling of examples where the respondent’s rating is not consistent with his/her verbatim response. In the example cases, the verbatim comments would appear to be inconsistent with the rating while the predictions are consistent with the text. While these cases are explicitly selected to illustrate the potential challenges of developing semantic analyzers based on survey data they suggest that even with a less desirable accuracy and higher mean absolute error than desired, the classifier may provide a good basis to enhance the company's NPS program and associate KPI metrics.\

Table 27 NPS Responses That Are Inconsistent with Verbatim Comments

Survey Response Three-Class Labels	Prediction Three-Class Labels	Verbatim Comments
Detractor	Promoter	1 experience 2 the support was fantastic
Detractor	Promoter	because it was very carefully supported
Detractor	Promoter	because of the quick response
Detractor	Promoter	interface and graphics capabilities
Detractor	Promoter	because we could respond promptly and as expected
Detractor	Promoter	after calling we quickly arranged replacement parts and technical personnel it was very helpful to solve problems in a few hours
Detractor	Promoter	good service
Detractor	Promoter	quick response and accurate answer
Detractor	Promoter	the positive experience prevails
Detractor	Promoter	competent friendly patient
Promoter	Detractor	1 very long and complex bureaucratic procedures 2 long lead times for orders
Promoter	Detractor	we had a performance issue not able to pinpoint that support has been able to come with
Promoter	Detractor	because we cannot access the system without our pcomm in our pc os environment
Promoter	Detractor	because the printing function of acs is not stable when it comes to printing it becomes pcomm which is the way to recommend it
Promoter	Detractor	this pmr has been very long and has already had a predecessor pmr with the same problem which could not be solved at the time
Promoter	Detractor	… price competitiveness is still weaker
Promoter	Detractor	time did not change the quality of the system ie of its granite operating system
Promoter	Detractor	vacations at grundfos and at … prolonged the handling time
Promoter	Detractor	the solution was not satisfactory
Promoter	Detractor	no good communication in this case

From a manual assessment of a sampling of 100 misclassifications, 44% of the misclassified cases in the test sample are cases where the NPS rating is not consistent with the entered score. The precise value is difficult to assess because the assessment can be very subjective for some of the examples. Using the midpoint of the range we would expect an accuracy rate above 80% for the classifier (MZE of 0.20) which is comparable to binary classifiers that are reported in published papers [71].

Chapter 11. APPLICATION: OHPLNET FOR INTERPRETTIVE ASSESSMENT

For decades businesses have used purchase RFM (Recency, Frequency and Monetary Value) to successfully engage with customers to successfully promote future spending. Companies also report a linkage between recency of interaction with the company and Net Promoter Score (NPS) survey response rating (see 114) [39]. While it is not likely that the two are perfectly correlated, it is not unreasonable to expect RFM metrics correspond to NPS survey response ratings. The metric has been demonstrated to have a strong association with future company revenues [72]. Given the relationship with future revenues, it is not unreasonable to assume that there may be some association between Net Promoter rating and customer spend prior to survey response.

A very large IT company with a large B2B (business to business) presence has a database of more than 400,000 NPS survey responses. Not all of the responses can be link to specific customer behavioral data, like products purchased and their timing, but a sufficient number can be linked to attempt to predict survey response, based on the pattern of customer purchase over the prior year. In the IT marketplace, “frequency” has dramatically different meaning for services than for hardware, both of which will have different meanings than for software. As such creating a purchase frequency metric across all purchases may be impractical to attempt. This company’s product portfolio is sufficiently complex, to the point that “frequency” is too difficult to provide a reliable metric, but a detail set of revenue data is available, across a broad timeline.

Historically, predictive analysis using the company’s data, indicated that purchase behavior in a recent time frame (e.g., last month or quarter) as well as for more extended time periods (e.g., prior year or two prior years) have a strong correlation to future purchase. The company’s NPS data has an additional complication in that larger customers have a high likelihood of providing multiple NPS survey responses, particularly when considering an extended period of time. This repeated response presents a significant complicating factor when attempting to build a model that predicts NPS response. To avoid cases where the same customer provided multiple, different responses, in the same time window (quarter), the most frequent response for the time period was chosen.

For the analysis, just over 71,000 B2B response records were available with detailed revenue, firmographic (industry, number of employees, number of business locations, etc.) and company sales “coverage” (larger customers have dedicated sales teams, while small customers may only receive telephone sales support). Customer purchase data (revenues) were totaled by product purchase and time period (prior month, prior quarter, prior half year, prior year and prior two years). In total, over 2,600 data elements were available for classification NPS response, based on known customer behavioral attributes.

Table 28 Net Promoter Response Distribution

Semantic Label	Response Scale	Response 3-Point Rescale	Record Count	Percentage
Detractor	0	0	2,798	3.9%
Detractor	1	0	996	1.4%
Detractor	2	0	1,170	1.6%
Detractor	3	0	1,273	1.8%
Detractor	4	0	1,075	1.5%
Detractor	5	0	3,665	5.1%
Detractor	6	0	2,756	3.8%
Passive	7	1	5,863	8.2%
Passive	8	1	11,672	16.2%
Promoter	9	2	12,549	17.5%
Promoter	10	2	28,085	39.1%

A successful model that the company could use as part of customer advocacy and retention efforts could not be built since there does not appear to be sufficient “signal” in the available feature set that is related to NPS response score. While this application doesn’t provide sufficient predictive power to provide a usable classifier to determine an expected NPS classification based on company purchase behaviors, it does provide sufficient predictive benefit above a “rational” random assignment to do a different assessment of the results, that assessing pure accuracy metrics. The challenges in analyzing the data lead to two valuable results:

1. OHPLnet experienced significant challenges in building a predictive model. The challenge lead to a significant revision of OHPLnet which became Two-Stage OHPLnet. During initial efforts to use OHPLnet the results, including multiple strategies outlined

in Chapter 8, the models demonstrated minimal benefit (incremental performance) over a rational random assignment because the algorithm could not consistently progress to the point of mapping to a new space that provided a proper ordering of the eleven response classes.

2. While Two-Stage OHPLnet provides a small amount of lift in fit over random assignment, it also provided an insight into the traditional aggregation of NPS values in to the documented three-point semantic scale. Since the model cannot be deemed a “success”, this insight into the clustering of scores provides a heuristic reinforcement of part of the NPS process.

The NPS responses were highly skewed towards the high end of the response scale (‘9’ and particularly ‘10’). To address this issue, multiple weighting schemes were tested. The “Weight 11 Point Scale” is a simple ratio of the corresponding cell with the largest cell count. This weighting assumes an equal likelihood of response for all eleven classes. Since NPS scores are aggregated, into fewer scale points the equal likelihood assumption is not likely to be valid, but was used as a test effort to “force” the maximum number of records to be scored in classes ‘0’-‘6’ (the best unweighted model result was 0.11% of the validation sample in classes ‘0’-‘6’). The simple square root of the “Weight 11 Point Scale” value provided weighting that is close to the rescaling but provides differentiated weighting for cells within the normal rescale by frequency. Lastly, a weighting based on the normal three point rescale, using the same methodology as for the “Weight 11 Point Scale” (i.e., the largest value for the three cells is divided by the corresponding value for the three-point class).

Table 29 IT Company NPS Response Counts and Tested Weighting Scales

Semantic Label	Response Scale	Response 3-Point Rescale	Record Count	Percentage	Weight 11-Point Scale	Weight Square Root 11-Point Scale	Weight 3-Point Scale
Detractor	0	0	2,798	3.9%	10.0	3.2	3.0
Detractor	1	0	996	1.4%	28.2	5.3	3.0
Detractor	2	0	1,170	1.6%	24.0	4.9	3.0
Detractor	3	0	1,273	1.8%	22.1	4.7	3.0
Detractor	4	0	1,075	1.5%	26.1	5.1	3.0
Detractor	5	0	3,665	5.1%	7.7	2.8	3.0
Detractor	6	0	2,756	3.8%	10.2	3.2	3.0
Passive	7	1	5,863	8.2%	4.8	2.2	2.3
Passive	8	1	11,672	16.2%	2.4	1.6	2.3
Promoter	9	2	12,549	17.5%	2.2	1.5	1.0
Promoter	10	2	28,085	39.1%	1.0	1.0	1.0

Multiple DNN models were estimated on an 80% training set in an attempted to provide a classification of the eleven-point response that was an improvement over random assignment. Due to the highly skewed nature of the response data, assigning all values to the class ‘10’ provides the best MZE score of 0.609, with a corresponding MAE score of 2.01. Since the goal of the effort is to provide predictions for classes, the eleven-point classes that fall in each of the three-point classes, it is not reasonable to set the random threshold to that of simply labeling all records with a value of ’10.’ Instead, there must be some records that are in classes ‘0’-‘6’. Since ‘6’ has the smallest absolute difference from ten, the random assignments will include random assignments in class ‘6’. The random assignment of records into classes ‘6’-‘10’, with ‘6’ assumed to have the frequency count of the classes ‘0’-‘6’, is referred to as the “rational random assignment”. In Table 30 Random Assignment of Classes, marginal probabilities for the classes were used to estimate a random assignment across classes six through ten based on the proportion of classes in the data set that fall in these 5 classes. The MAE and MZE values for this test were 0.735 and 2.024, respectively.

Table 30 Random Assignment of Classes

	Randomly Assigned Label				
Actual Label	6	7	8	9	10
‘0’	534	228	454	488	1093
‘1’	190	81	162	174	389
‘2’	223	95	190	204	457
‘3’	243	104	207	222	497
‘4’	205	88	175	188	420
‘5’	700	299	595	640	1,432
‘6’	526	225	447	481	1,076
‘7’	1,120	478	952	1,023	2,290
‘8’	2,229	952	1,895	2,037	4,559
‘9’	2,397	1,023	2,037	2,190	4,902
‘10’	5,364	2,290	4,559	4,902	10,970
Total	13,733	5,863	11,672	12,549	28,085

Creating a classification model for these data were particularly difficult. Initially, the OHPLnet Double Batch algorithm, with 8 hidden layers and between 10 and 500 nodes per layer (inverse pyramid design with 500 in the first layer and 10 in the last, with a proportional decrease in node number in layers between the first and last) was used, but could not consistently achieve the proper ordering of the hyperplane centroids, with a minimum distance of one unit between adjacent layers (even with a phenomenally high weighting value of one million, on the HCL component of loss). Multiple hyperplane centroids were ordered incorrectly at the completion of these tests. It was suggested that the algorithm be revised to focus on the ordering of the hyperplane centroids first, then allow the point loss component to be included. This suggestion lead to Two-Stage OHPLnet which represents a very meaningful breakthrough in algorithm speed, accuracy and consistency in models produced.

The new algorithm was able to produce results MZE and MAE that were better than the corresponding values for the proposed a rational random assignment, but the improved predictions weren't a major advancement over rational random assignment and a relatively small number of records were assigned to classes that fall into the "Detractor" classes. More notably, regardless of weighting, the hyperplane centroids that were generated by using the final results had class '1' and class '2' inverted (i.e., class '1' had a larger hyperplane centroid value) with small distances between the two (approximately 0.1 unit or less). This "inversion" of hyperplane centroids with small spacing in the final scoring suggested that classes '1' and '2' should be combined since the available feature set could not properly maintain the ordering, while minimizing the distances of the points from their hyperplane centroids. The collapsing of classes in the lower-class values that make up the "Detractor" class based on hyperplane centroid ordering (and small distances between the offending hyperplane centroids), continued until a seven-class solution was reached (see Table 31).

This same sequence of collapse held true through five model development cycles for each weighting scheme at each level of collapse. Once the seven-class solution was reached, the class ordering was maintained. Table 31 reports the mean MZE for five models produced using the same training and validation sets as well as the sequence of binning classes and related data. As would be expected, as the number of classes decreases, the accuracy metrics improve. At each step that combines two classes, the rational random assignment also improves. As the number of classes the MAE for the random assignment also improves, but not to as high a degree as the Two-Stage OHPLnet assignment improves (see Table 31). The acceleration of improved MAE is strong evidence that the combining of classes is an appropriate step.

Table 31 NPS Weighted Model Results with Binned Classes

Number of Classes	Inverted Classes	MAE	Random MAE	Improvement Over Random	Percent Improvement
11	‘1’ & ‘2’	1.837	2.024	0.190	9.4%
10	‘4’ & ‘5’	1.729	1.974	0.245	12.4%
9	‘1’/’2’ & ‘3’	1.632	1.873	0.241	12.9%
8	‘0’ & ‘1’/’2’/’3’	1.551	1.804	0.253	14.0%
7	None	1.517	1.765	0.248	14.1%
3	None	0.620	0.801	0.181	22.6%

Based on Final Model Hyperplane Centroid Values

At Since NPS scoring systems don’t utilize the eleven-point scale. They collapse the values into a three-point scale it is appropriate to attempt to produce a model that classifies the three-point scale. The problem remains an ordinal classification problem, but with fewer scale points. The results for the best performing model for the three-scale version are also reported in Table 31.

For consistency sake, the Weight 11-Point Scale results are reported. The other two weighting processes generated similar results.

The feature set contained a very high degree of collinearity among the features. In addition to using standard DNN architectures, on the raw feature set two different Nonlinear Principal Components analyses were used. For Nonlinear Principal Components (NPC), a three hidden layer autoencoder was used to reduce the feature set dimensions. To create data reduction solutions with 300 and 1,000 components, respectively. Over 90% of the variance contained in the 2,600-feature set were retained in each NPC solution, with 98% retained in the 1,000-component solution. The NPC versions of models did not perform as well as the models that used the raw data inputs, so they were abandoned.

While this analysis effort did not produce a model that effectively predicted NPS response rating, to the degree that it could be used as a key part of the company's NPS management system, it did provide a useful confirmatory insight for the company, in terms of the relationship of response ratings to the desired three-point semantic scale. These results are specific to the available purchase data. The inclusion of additional data from other company systems may provide sufficient "signal" to result in stronger model performance.

Chapter 12. CONCLUSIONS

This research takes a capable newly developed loss function for use in Deep Neural Network architectures and advances it to a more complete set of neural network strategies for solving ordinal classification problems. The original work that provided a complex loss function for ordinal classification problems was advanced by the development of multiple analysis strategies (Double Batch, Single Stratified, Epoch Stratified Batch). Ultimately OHPLall resulted from tests on a particularly difficult dataset where attempting to optimize Hyperplane Centroid ordering and distance at the same time as minimizing point distances from the point's corresponding Hyperplane Centroids, ended, with results where Hyperplane Centroids were not properly ordered. Developing the ordering of the centroids first, then minimizing point distances relative to fixed Hyperplane Centroids, proved to be a successful strategy for the dataset. The strategy also provided improved results for standard benchmark datasets that are used to evaluate different ordinal classifiers.

Applying OHPLall by attempting to predict Net Promoter Score (NPS) using customer purchase behaviors and customer attributes proved to be very difficult. The resulting models did not provide a significant improvement over using random assignment to a logical subset of the ordinal classes. It did however provide an insightful diagnostic, that verified the NPS assumption that the eleven-point scale could be reduced to a useful three-point scale, with effective class descriptions. The development of a successful "sentiment" type classifier of short verbatim text had significantly better results. While pure classification accuracy may not be as high as desired, a Gated Recurrent Neural Network using the new OHPLall variant performed well versus an

Ordinal Regression algorithm that used the same architecture. This performance advantage was achieved in spite of evidence that the data contained inconsistently labelled records that affected performance metrics.

When applied to classify mammography images into the BI-RADS rating scale, OHPLall required some modifications of the original Mini-Batch variant, to allow very small batches of large images. OHPLall developed Convolutional Neural Networks that consistently outperformed Convolutional Neural Networks that were built using Ordinal Regression. Possibly more importantly, OHPLall provided an appropriate “shift” of images of calcifications that were tested to have a malignancy into higher BI-RADS classes than did the Ordinal Regression models.

OHPLall is a powerful analytic tool that has been demonstrated to develop Deep Neural Network models of both structured and unstructured data. In “real world” applications, the algorithm performed better than benchmarks, even when presented difficult datasets that were filled with improper classifications or that hit system resource limitations. This work is a meaningful advancement in our ability to analyze an important class of predictive model development problems.

The application to real world data suggests that in addition to providing strong accuracy in classification OHPLall may provide diagnostic information regarding the classes. In particular, the algorithm may suggest adjacent classes that may not differ to the degree expected for the rating system in general.

BIBLIOGRAPHY

[1] M. Kim and V. Pavlovic, "Structured output ordinal regression for dynamic facial emotion intensity prediction," in *Proceedings of the 11th European Conference on Computer Vision*, 2010.

[2] K.-Y. Chang, C.-S. Chen and Y.-P. Hung, "Ordinal Hyperplanes Ranker with Cost Sensitivities for Age Estimation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Providence, RI, 2011.

[3] B. Nguyen, Morell, Carlos and De Baetsa, Bernard , "Distance metric learning for ordinal classification based on triplet constraints," *Knowledge-Based Systems,* no. 142, p. 17–28, 2018.

[4] K. Chang, C. Chen and Y. Hung, "Ordinal hyperplanes ranker with cost sensitivities for age estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[5] W. Chu and S. S. Keerthi, "New approaches to support vector ordinal regression," in *Proceedings of the 22nd International Conference on Machine learning*, Bonn, Germany, 2005.

[6] R. Herbrich, T. Graepel and K. Obermayer, "Support Vector Learning for Ordinal Regression," in *Ninth International Conference on Artificial Neural Networks, ICANN 99.*, Edinburgh, UK, 1999.

[7] H. Wang, Y. Shi, L. Niu and Y. Tian, "Nonparallel Support Vector Ordinal Regression," *IEEE TRANSACTIONS ON CYBERNETICS,* vol. 47, no. 10, pp. 3306-3317, 2017.

[8] W. Chu and Z. Ghahramani, "Gaussian Processes for Ordinal Regression," *Journal of Machine Learning Research,* no. 6, p. 1019–1041, 2005.

[9] P. Gutiérrez, M. Pérez-Ortiz and J. Sánchez-Mone, "Ordinal regression methods: survey and experimental study," *IEEE Trans. Knowl. Data Eng. 28,* no. 1, p. 127–146, 2016.

[10] W. Waegeman and L. Boullart, "An ensemble of Weighted Support Vector Machines for Ordinal Regression," in *Proceedings of World Academy of Science Engineering and Technology.* , Vienna, Austria, 2006.

[11] J. D. M. Rennie, "Ordinal Logistic Regression," MIT, 16 February 2005. [Online]. Available: http://people.csail.mit.edu/jrennie/writing/olr.pdf. [Accessed 20 March 2018].

[12] J. Cheng, "A Neural Network Approach to Ordinal Regression," 2007. [Online]. Available: http://arxiv.org/abs/0704.1028. [Accessed 5 July 2019].

[13] J. S. Cardoso, R. Sousa and I. Domingues, "Ordinal Data Classification Using Kernel Discriminant Analysis: A Comparison of Three Approaches," in *Proceedings of the 11th International Conference on Machine Learning Applications*, Boca Raton, FL, 2012.

[14] E. Frank and M. Hall, "A Simple Approach to Ordinal Classification," in *12th European Conference Machine learning: ECML*, Freiburg, Germany, 2001.

[15] J. S. Cardoso and J. Pinto da Costa, "Learning to Classify Ordinal Data: The Data Replication Method," *Journal of Machine Learning Research,* vol. 8, pp. 1393-1429, 2007.

[16] J. F. Pinto da Costa, R. Sousa and J. Cardoso, "An all-at-once Unimodal SVM Approach for Ordinal Classification," in *2010 Ninth International Conference on Machine Learning and Applications*, Washington D.C, 2010.

[17] O. C. Hamsici and A. M. Martinez, "Multiple Ordinal Regression by Maximizing the Sum of Margins," *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS,* vol. 27, no. 10, pp. 2072-2083, 2016.

[18] Schroff, Florian ; Kalenichenko, Dmitry; Philbin, James ;, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Boston, MA, 2015.

[19] K. Q. Weinberger and L. K. Saul, "Distance Metric Learning for Large Margin Nearest Neighbor Classification," *Journal of Machine Learning Research,* no. 10, pp. 207-244, 2009.

[20] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe and S. Singh, "No Fuss Distance Metric Learning using Proxies," arXiv:1703.07464v3, 2017.

[21] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks,* vol. 61, p. 85–117, 13 August 2015.

[22] A. Ng and K. Katanforoosh, "CS229 Lecture Notes," Stanford University, 29 October 2018 . [Online]. Available: http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf. [Accessed 12 March 2019].

[23] L. Bottou, "Stochastic Gradient Learning in Neural Networks," 1991. [Online]. Available: https://leon.bottou.org/publications/pdf/nimes-1991.pdf. [Accessed 23 March 2019].

[24] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Cornell University, 2014. [Online]. Available: https://arxiv.org/abs/1412.6980. [Accessed 11 March 2019].

[25] B. Lytle, "INTRODUCTION TO THE CONVERGENCE OF SEQUENCES," University of Chicago, 2015. [Online]. Available: http://math.uchicago.edu/~may/REU2015/REUPapers/Lytle.pdf. [Accessed 23 March 2019].

[26] V. K. Singh, "Proposing Solution to XOR problem using minimum configuration MLP," *Procedia Computer Science ,* vol. 85, p. 263 – 270 , 2016.

[27] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian, "Deep Residual Learning for Image Recognition," Google, 13 August 2018. [Online]. Available: http://arxiv.org/abs/1512.03385. [Accessed 12 March 2019].

[28] A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition," Stanford University, Spring 2019. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed 12 July 2019].

[29] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR,* vol. abs/1409.1556, no. http://arxiv.org/abs/1409.1556, 2014.

[30] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Neural Information Processing Systems 2012*, Lake Tahoe, NV , 2012.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed3, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015.

[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *CoRR,* vol. abs/1512.00567, 2015.

[33] S. Xie, R. Girshick, P. Dollar´, Z. Tu and K. He, "Aggregated Residual Transformations for Deep Neural Networks," *CoRR,* vol. abs/1611.05431, 2016.

[34] A. Zhang, Z. C. Lipton, M. Li and A. J. ⍰ Smola, "Dive into Deep Learning: Chapter 8.8. Gated Recurrent Units (GRU)," 2019. [Online]. Available: https://www.d2l.ai/chapter_recurrent-neural-networks/gru.html#reset-gate-in-action. [Accessed 5 July 2019].

[35] F. Kasper, "Computational Complexity Of Neural Networks," 25 March 2015. [Online]. Available: https://kasperfred.com/posts/computational-complexity-of-neural-networks. [Accessed 12 March 2019].

[36] D. Masters and C. Luschi, "Revisiting Small Batch Training for Deep Neural Networks," 2018. [Online]. Available: http://arxiv.org/abs/1804.07612. [Accessed 12 March 2019].

[37] V. Vapnik, "An overview of statistical learning theory," *Neural Networks IEEE Transac,* vol. 10, pp. 988-999, 1999.

[38] B. Vanderheyden and Y. Xie, "Ordinal Hyperplane Loss," in *2018 IEEE International Conference on Big Data*, Seattle, WA, 2018.

[39] Tyler, Tim, "Net Promoter® Links to Recency-Frequency-Monetary (RFM)," Genroe, [Online]. Available: https://www.genroe.com/blog/net-promoter-links-to-recency-frequency-monetary-rfm/597. [Accessed 15 March 2019].

[40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard and R. Jozefowicz, "TensorFlow: A System for Large-Scale Machine Learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, Savannah, GA, 2016 .

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau and M. Brucher, "Scikit-learn: Machine Learning in Python,," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[42] T. E. Oliphant, A guide to NumPy, USA:, Trelgol Publishing, 2006.

[43] S. C. C. a. G. V. Stéfan van der Walt, The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 2011.

[44] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Austin, TX, 2010.

[45] D. Dua and K. E. Taniskidou, "UCI Machine Learning Repository," University of California, School of Information and Computer Science., 2017. [Online]. Available: http://archive.ics.uci.edu/ml. [Accessed 15 March 2018].

[46] W. Chu, "BENCHMARK of ORDINAL REGRESSION: datasets and results," UCL Gatsby Computational Neuroscience Unit, 11 April 2004. [Online]. Available: http://www.gatsby.ucl.ac.uk/~chuwei/ordinalregression.html. [Accessed 1 April 2018].

[47] A. B. David, "Monotonicity Maintenance in Information-Theoretic Machine Learning Algorithms," *Machine Leaming,* no. 19, pp. 29-43, 1995.

[48] H.-T. Lin and L. Li, "Large-Margin Thresholded Ensembles for Ordinal Regression: Theory and Practice," in *Algorithmic Learning Theory, 17th International Conference, ALT*, Barcelona, Spain, 2006.

[49] Y. Freund, R. Iyer, R. E. Schapire and Y. Singer, "An Efficient Boosting Algorithm for Combining Preferences," *Journal of Machine Learning Research 4,* no. 4, pp. 933-969, 2003.

[50] C. Ju, A. Bibaut and M. J. van der Laan, "The Relative Performance of Ensemble Methods with Deep Convolutional Neural Networks for Image Classification," https://arxiv.org/abs/1704.01664v1, 2017.

[51] A. Hermans, L. Beyer and B. Leibe, "n Defense of the Triplet Loss for Person Re-Identification," *CoRR,* vol. abs/1703.07737, 2017.

[52] American Cancer Society, "Breast Cancer Facts & Figures 2017-2018," American Cancer Society, Atlanta, GA, 2017.

[53] American Cancer Society, "Understanding Your Mammogram Report," American Cancer Society, 9 October 2017. [Online]. Available: https://www.cancer.org/cancer/breast-cancer/screening-tests-and-early-detection/mammograms/understanding-your-mammogram-report.html. [Accessed 15 March 2019].

[54] K. Clark, B. Vendt, K. Smith, J. Freymann, J. Kirby, P. Koppel, S. Moore, S. Phillips, D. Maffitt, M. Pringle, L. Tarbox and F. Prior, "The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository," *Journal of Digital Imaging,* vol. 26, no. 6, pp. 1045-1057, 2013.

[55] R. S. Lee, F. Gimenez, A. Hoogi, K. K. Miyake, M. Gorovoy and D. L. Rubin, "A curated mammography data set for use in computer-aided detection and diagnosis research.," *Scientific Data,* vol. 4, 2017 volume 4, Article number: 170177 (.

[56] R. S. Lee, F. Gimenez, A. Hoogi and D. Rubin, "Curated Breast Imaging Subset of DDSM," The Cancer Imaging Archive, http://dx.doi.org/10.7937/K9/TCIA.2016.7O02S9CY, 2016.

[57] ksmith01 and Klingerc, "The Cancer Imaging Archive (TCIA) Public Access: CBIS-DDSM," The Cancer Imaging Archive (TCIA), 21 November 2018. [Online]. Available: https://wiki.cancerimagingarchive.net/display/Public/CBIS-DDSM#01fc928dcc1f420f9cd2dd80fdd37b16. [Accessed 5 March 2019].

[58] L. Shen, L. R. Margolies, J. H. Rothstein, R. B. McBride, E. Fluder and W. Sieh, "Deep Learning to Improve Breast Cancer Early Detection on Screening Mammography," *Clinical Orthopaedics and Related Research,* vol. abs/1708.09427, 13 August 2018.

[59] T. Kyono, F. J. Gilbert and M. van der Schaar, "MAMMO: A Deep Learning Solution for Facilitating Radiologist-Machine Collaboration in Breast Cancer Diagnosis," *CoRR,* vol. abs/1811.02661, 2018.

[60] R. Agarwal, O. Diaz, X. Lladó, M. H. Yap and R. Martí, "Automatic mass detection in mammograms using deep convolutional neural networks," *Journal of Medical Imaging,* vol. 6, no. 3, 2019.

[61] I. Moreira, I. Amaral, I. Domingues, A. Cardoso, M. Cardoso and J. Cardoso, "INbreast: toward a full-field digital mammographic database," *Academic Radiology,* vol. 19, no. 2, pp. 129-262, 2012.

[62] H. G. Li, M. L, B. Q. Huynh and N. O. Antropova, "Deep learning in breast cancer risk assessment: evaluation of convolutional neural networks on a clinical dataset of full-field digital mammograms," *Journal of Medical Imaging ,* vol. 4, no. 4, 2017.

[63] F. F. Reichheld, "The One Number You Need to Grow," *Harvard Business Review,* vol. December 2003, 2003.

[64] Medallia Corporation, "Net Promoter Score®," Medallia Corporation, 2019. [Online]. Available: https://www.medallia.com/net-promoter-score/. [Accessed 11 March 2019].

[65] Medallia, Inc, "Medallia Recognizes World's Most Innovative Customer Experience Leaders," Medallia, Inc, 16 May 2018. [Online]. Available: https://www.medallia.com/press-release/medallia-recognizes-worlds-most-innovative-customer-experience-leaders/. [Accessed 11 July 2019].

[66] B. Carremans, "Word embeddings for sentiment analysis," Towards Data Science , 27 August 2018. [Online]. Available: https://towardsdatascience.com/word-embeddings-for-sentiment-analysis-65f42ea5d26e. [Accessed 5 July 2019].

[67] S. Gupta, "Sentiment Analysis: Concept, Analysis and Applications," Towards Data Science, 7 January 2018. [Online]. Available: https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17. [Accessed 5 July 2019].

[68] J. Pennington, R. Socher and C. D. Manning, "GloVe: Global Vectors for Word Representation," Stanford University, August 2014. [Online]. Available: https://nlp.stanford.edu/projects/glove/. [Accessed 5 July 2019].

[69] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in *Proceeding of Neural Information Processing Systems (NIPS)*, Lake Tahoe, 2013.

[70] J. Pennington, R. Socher and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proceedings Empirical Methods in Natural Language Processing*, Doha, Qatar, 2014.

[71] İ. Tarımer, A. Çoban and K. A. Emre, "Sentiment Analysis on IMDB Movie Comments and Twitter Data by Machine Learning and Vector Space Techniques," 2019. [Online]. Available: http://arxiv.org/abs/1903.11983. [Accessed 5 July 2019].

[72] T. L. Keiningham, B. Cooil, T. W. Andreassen and L. Aksoy, "A Longitudinal Examination of Net Promoter and Firm Revenue Growth," *Journal of Marketing,* p. 39–51, 2007.

[73] H.-T. Lin� and L. Li, "Reduction from Cost-sensitive Ordinal Ranking to Weighted Binary Classification," *Neural Computing,* no. 5, pp. 1329-1367, 2012.