2020

Evaluating an Ordinal Output using Data Modeling, Algorithmic Modeling, and Numerical Analysis

Martin Keagan Wynne Brown
Murray State University

Evaluating an Ordinal Output using Data Modeling, Algorithmic Modeling, and Numerical Analysis

A Thesis

Presented to

the Faculty of the Department of Mathematics and Statistics

Murray State University
Murray, Kentucky

In Partial Fulfillment

of the Requirements for the Degree

of Master of Science

by

Martin Brown

May 2020

Evaluating an Ordinal Output using Data Modeling, Algorithmic Modeling, and Numerical Analysis

DATE APPROVED: ____________________

Dr. Donald Adongo, Thesis Advisor

Dr. Christopher Mecklin, Thesis Advisor

Dr. Manoj Pathak, Thesis Committee

Dr. Maeve McCarthy, Graduate Coordinator, Jesse D. Jones College of Science, Engineering, and Technology

Dr. Claire Fuller, Dean, Jesse D. Jones College of Science, Engineering, and Technology

Dr. Robert Pervine, University Graduate Coordinator

Dr. Timothy Todd, Provost

Acknowledgements

I would like to thank the Mathematics and Statistics Department at Murray State University for providing the most rewarding and interesting experience. So many individuals have influenced me in profound ways, and I would like to mention a few of them.

To Dr. Donald Adongo, thank you for making my transition to America as smooth as possible. I have enjoyed our long conversations in your office, and you have made me feel welcomed and comfortable at Murray State. Also, thank for your support in my classes and thesis. Your enthusiasm in helping students is something I will never forget.

To Dr. Christopher Mecklin, thank you for introducing me to machine learning. You helped me to discover my passion and develop grit. Also, thank you for your support in my thesis and being available at a moment's notice to advise me during the tougher moments.

To Dr. Manoj Pathak, thank you for introducing me to the world of statistics; I did not know what I was missing. Also, thank you for your help during the thesis process.

To Claire Ghent, thank you for your help throughout my studies, and thank you for the endless grammer corrections and proof readings as well as bringing me tea during the writing process.

To my parents, Lindsay Brown and Elaine Brown, thank you for giving me the opportunity to study my Master's degree. You have both supported and provided strong encouragement throughout all my endeavors. I could not be where I am today without your help.

Abstract

Data and algorithmic modeling are two different approaches used in predictive analytics. The models discussed from these two approaches include the proportional-odds logit model (POLR), the vector generalized linear model (VGLM), the classification and regression tree model (CART), and the random forests model (RF). Patterns in the data were analyzed using trigonometric polynomial approximations and Fast Fourier Transforms. Predictive modeling is used frequently in statistics and data science to find the relationship between the explanatory (input) variables and a response (output) variable. Both approaches prove advantageous in different cases depending on the data set. In our case, the data set contains an output variable that is ordinal. Using grade records from Murray State University, the goal is to find the best predictive model that can implement an ordinal output by means of data modeling and algorithmic modeling.

To train the models, k-fold cross validation is used to find the optimal tuning parameters and performance for each of the models. The logarithmic loss (logLoss) performance metric is utilized to determine which method has the top predictive accuracy. A comparison of each statistical model and a look at alternative methods is discussed.

Contents

Chapter 1

Introduction

Leo Breiman [1] explains that statistical modeling can be split into two different cultures, data modeling and algorithmic modeling. Breiman uses the analogy of a black box that generates the data, where inputs, X_i for some i, enter the black box producing the output, Y. Data modeling involves making assumptions for the internals of the black box then builds a model based on these assumptions. This is the approach that statisticians traditionally employ. Algorithmic modeling treats the internal nature of the black box as unknown and alternatively finds an algorithm that takes the inputs to predict the output based on patterns found in the data.

Questions on prediction are more prevalent than ever. From using relevant factors to predict consumer shopping habits to using relevant factors to predict if a person has diabetes, we are able to answer more questions with the ever-increasing amount of data. The main goal of this thesis is to determine the best model, in terms of predictive accuracy, that predicts an ordinal output. In our case, the output variable is student grades collected from two different courses offered by the mathematics and statistics department at Murray State University. Specifically, the models are chosen to deal with the ordinal output variable commonly referred to in supervised machine learning as classification. In addition, we combine the fields of statistics, machine learning, and numerical analysis to evaluate our ordinal data. As important as accuracy is, a discussion on the interpretability of each model is included because an explanation of results is important when including a non-technical audience.

The arrangement of this thesis begins in chapter 2 with a description of the data used as well as an introduction of the concepts of the logarithmic loss performance metric (logLoss) and k-fold cross validation (kFCV). In chapter 3, we begin determining the best method to prognosticate student grades with a detailed description of the proportional-odds logistic regression model (POLR). In chapter 4, the vector generalized linear model (VGLM) is considered to evaluate the proportionality of our data set. In chapter 5 and 6, we investigate the classification and regression tree (CART) model and the adapted random forests model (RF), respectively. In chapter 7, a numerical anaylsis study is introduced using trigonometric approximation polynomial and Fourier series to analyze the trends in the data. Lastly, a comparison of the methods is discussed and an examination into further methods for potential use is conducted in chapter 8.

Chapter 2

Training and Testing

In this thesis, the goal is to predict student grades obtained from data records from Murray State University and compare the best predictive model for the ordinal output. The output variable of student grades is ordinal, which Fox et al. [2] describes this as categories that have a natural order. This is different from multinomial data, where the data set is categorical but does not have an intrinsic order. An example of multinomial would be a person's favorite color such as green, blue, or red.

This study omits students who have audited a course and combines the students who have received the grade E and who withdrew from the course with a W on their transcript. Therefore, the data set used throughout the thesis contains an ordered factored output variable as displayed in (G). Models involving this type of output are referred to as multi-class classification problems.

$$EW < D < C < B < A. \tag{G}$$

The input variables in consideration are:

- 'Semester': a factor indicating if the class was held in 'Spring' or 'Fall';
- 'StartTime': a factor indicating the start time of the class;
- 'Size': a number indicating the number of students in the class;

- ‘YrSince2003’: a number indicating how many years since 2003 that the class was taught;
- ‘Class’: a factor indicating if the class is Introduction to Probability and Statistics ‘MAT135’ (called ‘STA 135’ since 2016), or, Calculus and Analytic Geometry I ‘MAT250’,
- ‘Day’: a number indicating the number of days a week that the class meets.

With all the different models, we require a performance metric to be able to determine which method is best suited for predicting student grades (G). The logLoss function is a consistent metric that deals well with multi-class classification problems [3] and can be utilized in areas from machine learning to numerical analysis. Since we cannot wait for a new set of data to test the model without waiting for months or years, we require the data set to be split into a training set and a testing set in an attempt to prevent the models from overfitting the data. The model will be performed on the training set and validated on the testing set, which is a completely seperate portion of the data set and is not used in the training set.

To train the model, we perform k-fold cross validation demonstrated in chapter 2.2. To train our model and calculate the logLoss metric, we use the ‘caret’ package [11] in the statistical program R. $MATLAB$ is used to implement the trigonometric polynomials.

2.1 Logarithmic Loss

Logarithmic loss, or commonly referred to as logLoss, is a metric that penalizes the false classifications given by a model. This works especially well for multi-class classification where the method assigns a probability to each of the classes for all observations [3]. Therefore, the logLoss function was chosen over traditional accuracy metrics because we are not predicting a binary response. Let M be the number of classes in the data set and let N be the number of observations belonging to our classes M. In our case, we have $M = 5$ classes in (G), and we have $N = 1760$ observations. Then, the logLoss function is given by

$$logLoss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\, ln(p_{ij}), \tag{2.1.1}$$

where $y_{ij} = \{0, 1\}$ indicates if observation i belongs to class j, and p_{ij} indicates the probability of observation i belonging to class j [3]. Also, ln refers to the natural log.

Note that the logLoss will only contain the summation of misclassified observations and will heavily penalize the observations that were confident and incorrect [5]. A model that perfectly predicts student grades (G) will have a logLoss of zero therefore, the logLoss takes on values $[0, \infty)$, i.e. no upper bound. A model with higher accuracy has a logLoss closer to zero thus, we must minimize the logLoss to improve the accuracy of our model.

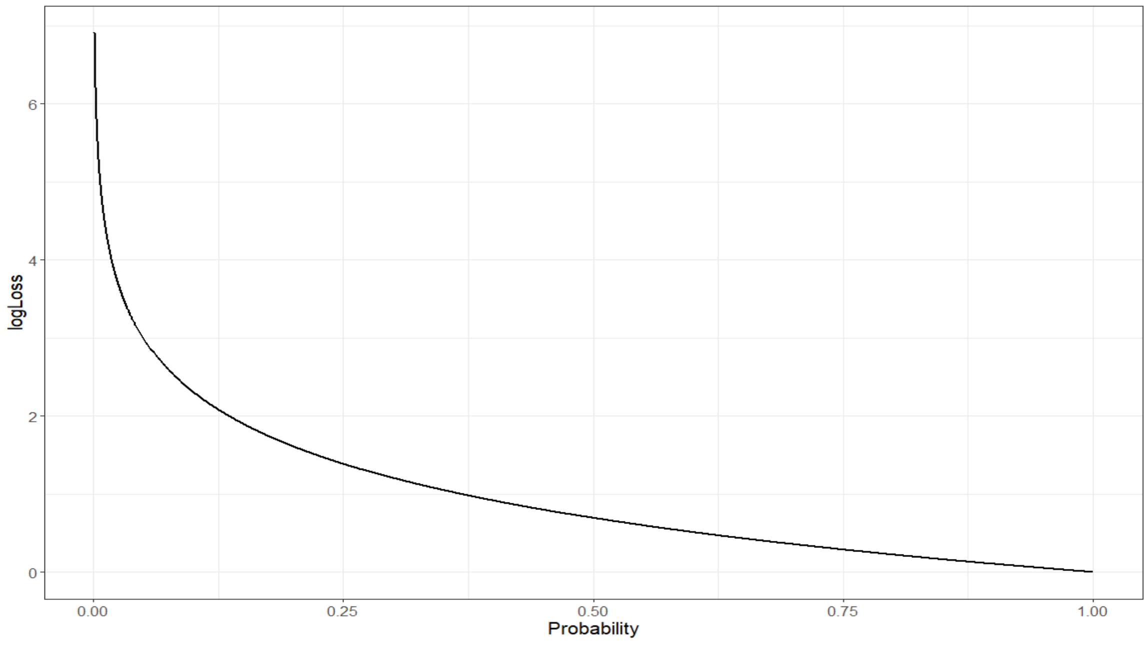

Figure 2.1.1: Plot of logLoss against predicted probability

In figure 2.1.1, the predicted probability is zero for incorrect predictions and one for correct predictions. The logLoss gradually decreases as the predicted probability increases, and on the left of figure 2.1.1, its apparent that the model is heavily penalized on making confident incorrect predictions.

Since we have $M = 5$ classes, we can calculate the "dumb" or non-informative logLoss for our classification problem [6]. This is the same as assuming that students' grades are uniformly distributed, where each grade has a 20% chance of occuring. Hence, the "dumb" logLoss value is

$$logLoss = -ln\Big(\frac{1}{M}\Big) = -ln\Big(\frac{1}{5}\Big) = ln(5) = 1.6094.$$

We can compare the subsequent models to the uniform distribution of the grades (G), which has a logLoss of 1.609437912. Since the distribution of grades is not exactly uniform, as certain grades are more prevalent and others less prevalent in the actual data, the "dumb" logLoss for grade predictions will be slightly less than $ln(5)$. Let us consider a situation when there are 15% A grades, 30% B grades, 25% C grades, 10% D grades and 20% EW grades. Then, the non-informative logLoss can be calculate using the entropy function (5.0.2), which will be defined in chapter 5.

$$\begin{aligned} logLoss &= -(0.15\,ln(0.15) + 0.30\,ln(0.30) + 0.25\,ln(0.25) + 0.10\,ln(0.10) \\ &\quad + 0.20\,ln(0.20)), \\ &= 1.5448. \end{aligned}$$

2.2 K-Fold Cross Validation

The code for processing the data set is in appendix A.1. Firstly, we split the data into a training and testing set. Secondly, we converted this "wide" formated data, which had a column for each of the grades (G), to a "long" format, where we created a column of grades, 'Grade', along with a column of frequencies of each grade, 'Freq'. Lastly, uncounting the frequencies for each grade, we extended the data to the "longest" format. The cross validation will be performed on the "longest" training set, 'Grades.training', and the model will be tested on the "longest" testing set, 'Grades.testing'. Overall, we are going from a "wide" format, where each section of a class is represented by a single row, to a "long" format, where each section is represented by five rows, one for each of the grades. Then, we converted the "long" format to the "longest" format, where each section is represented by a number of rows that is equivalent to the number of students in that class. The "longest" format is useful to fit the models, make predictions, and correctly compute the logLoss using the built-in *R* function.

Our goal is to find a method that predicts student grades (G), minimizes the overall logLoss, and is better than just assuming a uniform distribution to make random predictions. The main objective of cross validation is to use our training set, which is separate from the testing set, and create folds within the training set to train our model and find the optimal model provided certain tuning parameters. A training set is one in which the model is constructed, and the testing set is used to test the validity of the model given the optimal tuning parameters. To begin, the test error is the average error that results from using a statistical or machine learning method to predict the response on a new observation [8]. This is the error produced from using a model on the training set. The test error is the logLoss metric defined in chapter 2.1. Cross validation estimates the testing error by performing the logLoss on the folds from the training set. However, we require a large enough training set to perform the methods but equally a large enough testing set to validate. We separated the full data set into a 2/3-rds portion for the training set, called 'Grades.training', and the remaining 1/3-rd portion for the testing set, called 'Grades.testing'. This is shown in the *R* code in appendix A.1. The splitting was performed on the "wide" data set in order to keep all the students in a particular section of a class together, and then the processing from "wide" to "long" to "longest" was performed after sectioning off a training set and testing set.

Using the training set exclusively, we performed k-fold cross validation. Here, k

refers to the number of partitions or folds that the data is split into [8]. It works by randomly dividing the data set into $\frac{1}{k}$ approximately equally spaced portions. The k-th fold is used as the validation set that is used to estimate the testing error, and all but the k-th fold is used as the training set in which the model is fitted on. A common choice for k, and used in this thesis, is $k = 10$. In our case, 10% of the data is used as the validation set, and the remaining 90% is used to fit a model. This is a good compromise to leave-one-out-cross-validation (LOOCV), where k will be equal to leaving out one observation with replacement and fitting a model, i.e. $k = n$, which is computationally expensive.

Each fold produced is a different iteration, and at each iteration the logLoss values $logLoss_1, \ldots, logLoss_k$ are calculated, which is the testing error for each fold. For $k = 10$ folds, the 10-fold cross validation estimate, [8], is the average of the values $logLoss_i$ for all $i = 1, 2, \ldots, 10$:

$$\begin{aligned} CV_{(k)} &= \frac{1}{10} \sum_{k=1}^{10} logLoss_k, \\ &= \frac{1}{10} \sum_{k=1}^{10} \left(-\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij,k}\, ln(p_{ij,k}) \right). \end{aligned} \tag{2.2.1}$$

As illustated in figure 2.2.1, we see $k = 10$ fold cross validation splits the data into a validation set, $k = 1$, and uses the remaining $k = 9$ as the training set. In the first iteration, the first fold in blue indicates the validation set. In the second iteration, the second fold in blue is used as the validation set; this is repeated until we use all the folds as the validation set. This is used to create a logLoss value at each iteration on each validation set leading to equation (2.2.1).

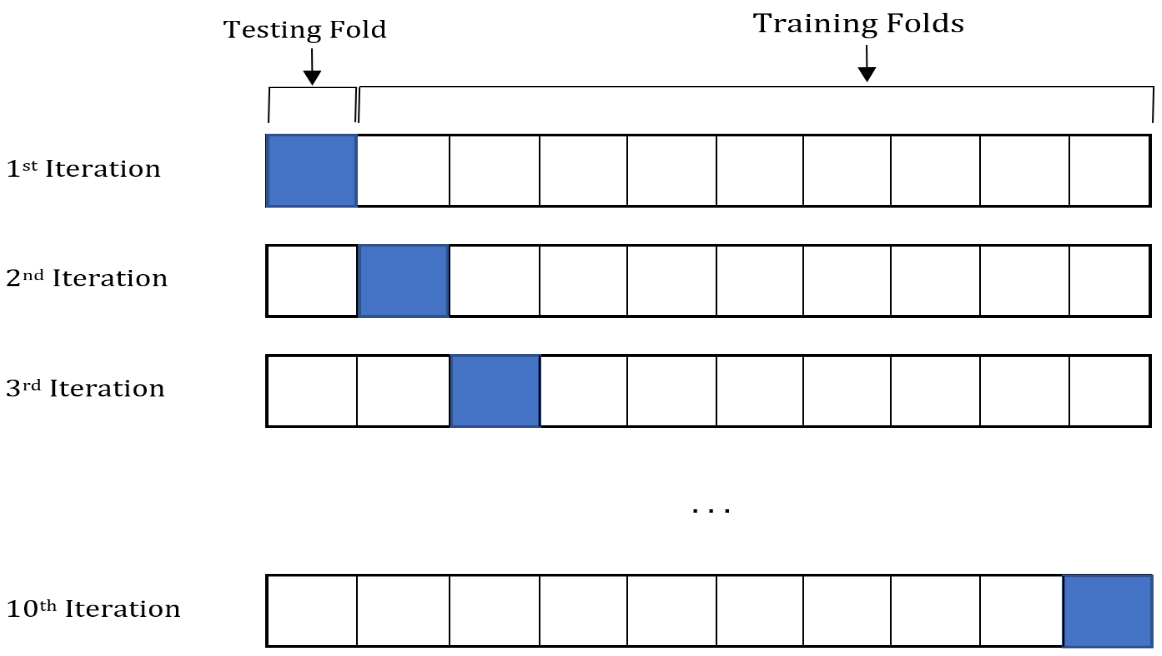

Figure 2.2.1: Demonstration of $k = 10$ fold cross validation

To implement cross validation, we require the use of a statistical program R. In particular, Kuhn et al. [11] created the ‘caret’ package which contains the functions ‘trainControl’ and ‘train’. Using these functions, we can train the models to find the minimum logLoss values for a certain tuning parameter associated with the model.

2.3 Training with the caret Package

This section accompanies the R code in section A.2. As mentioned before, we arbitrarily split our data set into a training set by randomly selecting two thirds of the data set. Then cross validation was preformed on the POLR model, followed by the VGLM model, the CART model, and then the RF model. The optimal tuning parameters will be used on the testing set in the subsequent chapters, and the logLoss of the tested model will be calculated. Cross validation will only be performed on the statistical study.

2.3.1 Cross Validation for POLR

Using the ‘train’ function from the ‘caret’ package, we can find the minimum logLoss value for the proportional-odds logistic regression model. Within the ‘train’ function we use $method = polr$ to train the model on the tuning parameters. We can train the model based on the type of regression model we should use; in other words, the tuning parameter is distinguishing which model is best between using a logistic regression model, *logistic*, or using a normal distribution *probit* model. Equivalently, this is determining if we should use a *logit* link or a *probit* link, respectively. These distributions were chosen because of work done by Fox [14] comparing the two distributions. The R output of ‘train.polr’ is given below.

```
>print(train.polr)
Ordered Logistic or Probit Regression

7703 samples
   6 predictor
   5 classes: ‘EW’, ‘D’, ‘C’, ‘B’, ‘A’

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 6934, 6933, 6933, 6931, 6932, 6931, ...
Resampling results across tuning parameters:

  method    logLoss
  logistic  1.550844
  probit    1.550876

logLoss was used to select the optimal model using the smallest
value.
The final value used for the model was method = logistic.
```

As seen above, logLoss was used to select the optimal model using the smallest value. In this case, the optimal model is using logistic regression with a logLoss value of 1.550844. The *probit* link has a logLoss value that agrees up to about 1.5508 (four decimal places), but we chose the method that produces the slightly lower logLoss while also taking into consideration the interpretability of the model. Fox [14] states that the advantages of using logistic over probit is that it is simpler and easier to work with, since the CDF is very simple and we do not have to evaluate an integral, and the logistic inverse transformation is directly interpretable as odds, which will be defined in the next chaper. To summarize, student grades (G) follow a logistic distribution where we can use a *logit* link for the POLR model.

2.3.2 Cross Validation for the Vector Generalized Linear Model

Alternatively, we have the cumulative probability model for ordinal data, or refered to as the vector generalized linear model. We can train this model to find the optimal model based on the type of link used, such as *logit*, *probit*, *cloglog*, *cauchit*, or *logc* link. This is almost the same as the POLR model, but we are considering more possibilities for the link function with the VGLM model. Furthermore, we trained each of these links based on whether we include the parallelism assumption, which Yee in [15] states this as determining if the data follows a parallel assumption − if the data set has proportional odds, i.e. specifying if the estimated coefficients of the VGLM model have equal/unequal coefficients. Further explanation on the parallelism assumption and the tuning parameter *parallel* can be found in chapter 4. In R, we use the 'train' function with $method = vglmCumlative$ to train the link and parallel assumption, which is implemented by either setting $parallel = TRUE$ or $parallel = FALSE$. The R output of 'train.vglm' is given below.

```
> print(train.vglm)
Cumulative Probability Model for Ordinal Data

7703 samples
   6 predictor
   5 classes: ‘EW’, ‘D’, ‘C’, ‘B’, ‘A’

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 6932, 6933, 6933, 6933, 6934, 6933, ...
Resampling results across tuning parameters:
  parallel  link     logLoss
  FALSE     logit    1.551378
  FALSE     probit   1.551429
  FALSE     cloglog  1.551526
  FALSE     cauchit  1.551348
  FALSE     logc     1.551420
   TRUE     logit    1.550876
   TRUE     probit   1.550868
   TRUE     cloglog  1.552892
   TRUE     cauchit  1.553208
   TRUE     logc     1.552920

logLoss was used to select the optimal model using the smallest
value.
The final values used for the model were parallel = TRUE,
and link = probit.
```

As we can see above, the optimal VGLM model has a logLoss value of 1.550868 with $parallel = TRUE$ with a *probit* link. Again, the *probit* and *logit* links are very similar in terms of logLoss, so we will use a *logit* link as it is more interpretable and is commonly used in data modeling. This is almost exactly the same logLoss value of the POLR model with the same link and assumption of parallelism. These models are very similar but the implementation is slightly different: the POLR model uses maximum likelihood estimates and the VGLM model uses matrices. We can deduce by cross validation that both the POLR and VGLM model fit a proportional-odds model. Note that under the parallel assumption, each of the links trained above have a logLoss value that agree up to about 1.55 (two decimal places), which indicates that the link chosen is arbitrary. To summarize, we will use a *logit* link and the parallel assumption, $parallel = TRUE$.

2.3.3 Cross Validation for CART

Similarly, using the 'train' function we can find the optimal parameters of the CART method. Firstly, using $method = rpart$ we can find the best complexity parameter, cp. Secondly, using $method = rpart1SE$ we can find the optimal model using the one-standard error method [12]. Lastly, we canl find the optimal maxdepth for the classification tree holding the cp value constant at the value we deduce from *method* = *rpart*. The maximum classification tree depth, maxdepth, is the tuning parameter that controls the maximum depth of the terminal nodes. An explanation of each of the tuning parameters and the CART model can be found in chapter 5. To start, we let the 'train' function find the optimal cp value from 100 randomly selected cp values by setting *tuneLength* = 100. We have the following results for 'train.rpart', leaving out most of the cp values explored.

```
> print(train.rpart)
CART

7703 samples
   6 predictor
   5 classes: 'EW', 'D', 'C', 'B', 'A'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 6932, 6933, 6933, 6933, 6934, 6933, ...
Resampling results across tuning parameters:

  cp            logLoss
  1.189925e-03  1.555764
  1.232423e-03  1.555437
  1.274920e-03  1.555437
  1.317417e-03  1.555450
  1.359915e-03  1.555340
  1.402412e-03  1.554837
  1.444909e-03  1.554513
  1.487407e-03  1.555046
  1.529904e-03  1.555084
  1.572402e-03  1.554646
  1.614899e-03  1.554571
  1.657396e-03  1.554571
  1.699894e-03  1.554571
  1.742391e-03  1.554777
  1.784888e-03  1.555037
  1.827386e-03  1.550890
  1.869883e-03  1.550890
  1.912380e-03  1.551645
  1.954878e-03  1.551645
  1.997375e-03  1.551645
```

```
logLoss was used to select the optimal model using the smallest
value.
The final value used for the model was cp = 0.001869883.
```

The ‘train’ function deduces that the optimal model used a complexity parameter value of 0.001869883 with a logLoss of 1.550890.

Hastie et al. [13] states that the one-standard error method is a rule used with cross validation: which is used we to choose the most parsimonious model whose error is no more than one standard error above the error of the best model. This can be applied to finding the optimal cp value, where we find the one-standard error threshold of the logLoss values. Therefore, using $method = rpart1SE$ we get a logLoss value of 1.558576 with complexity parameter of 0, which tells us that using the one-standard error method to deduce the cp value is not as accurate as using a cp of 0.001869883. In chapter 5, we will see that a cp of 0 will allow the classification tree to grow until there are only a few observations left in each node, which is not helpful in making predictions.

Now, we can train the CART model to find the maximum tree depth, maxdepth, controlling for a cp of 0.001869883. Using $method = rpart2$ in the ‘train’ function, we can find the optimal classification tree depth. The logLoss value for all tree depths ranging from two to 30 produced a logLoss of 1.558576. This tells us that tuning for maxdepth is arbitrary, and since we want an interpretable tree that is not too big, we will use $maxdepth = 5$ in R. We will see in chapter 5 that the complexity parameters greater than zero control the tree depth. This is the reason the logLoss for the $maxdepth$ does not change for $cp = 0.001869883$. The R code for $maxdepth$ training can be found in A.2. To summarize, we will use a cp of 0.001869883 with $maxdepth = 5$.

2.3.4 Cross Validation for Random Forests

The last of our machine learning models is the random forest. The tuning parameters involved are *mtry* and *num.trees*. Briefly, the tuning parameter *mtry* is the number of input variables to possibly split in each node in the classification tree, and *num.trees* is the total number of classification trees grown by the random forest model. The tuning parameter *splitrule* is held constant with "gini" to implement the Gini index splitting criterion 5.0.1 described in chapter 5. Also, the tuning parameter *min.node.size* is held constant at one so that the minimum number of observations that can remain in a terminal node is a single observation. A detailed explanation of each tuning parameter is included in chapters 5 and 6.

Firstly, using *method* = *ranger* in the 'caret' package and holding *splitrule* = "*gini*" and *min.node.size* = 1 constant, we can train for the tuning parameter *mtry*, producing the following *R* output for 'train.rf'.

```
> print(train.rf)
Random Forest

7703 samples
   6 predictor
   5 classes: 'EW', 'D', 'C', 'B', 'A'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 6932, 6933, 6933, 6933, 6934, 6933, ...
Resampling results across tuning parameters:

  mtry  logLoss
  2     1.541947
  3     1.543003
  4     1.551272
  5     1.563008
  6     1.575714

Tuning parameter 'splitrule' was held constant at a value of gini
Tuning parameter 'min.node.size' was held constant at a value of 1
logLoss was used to select the optimal model using the smallest
value.
```

```
The final values used for the model were mtry = 2,
splitrule = gini , and min.node.size = 1.
```

As shown above, an optimal random forest model uses $mtry = 2$ with a minimum logLoss value of 1.541947. In other words, for each node there will be only two random input variables available to split the node.

Secondly, holding $splitrule =$ "$gini$", $min.node.size = 1$, and $mtry = 2$ constant, we can use the training set to build 100 different random forests, increasing the tuning parameter $num.trees$ to find the minimum logLoss value. The results are summarised in figure 2.3.4.1.

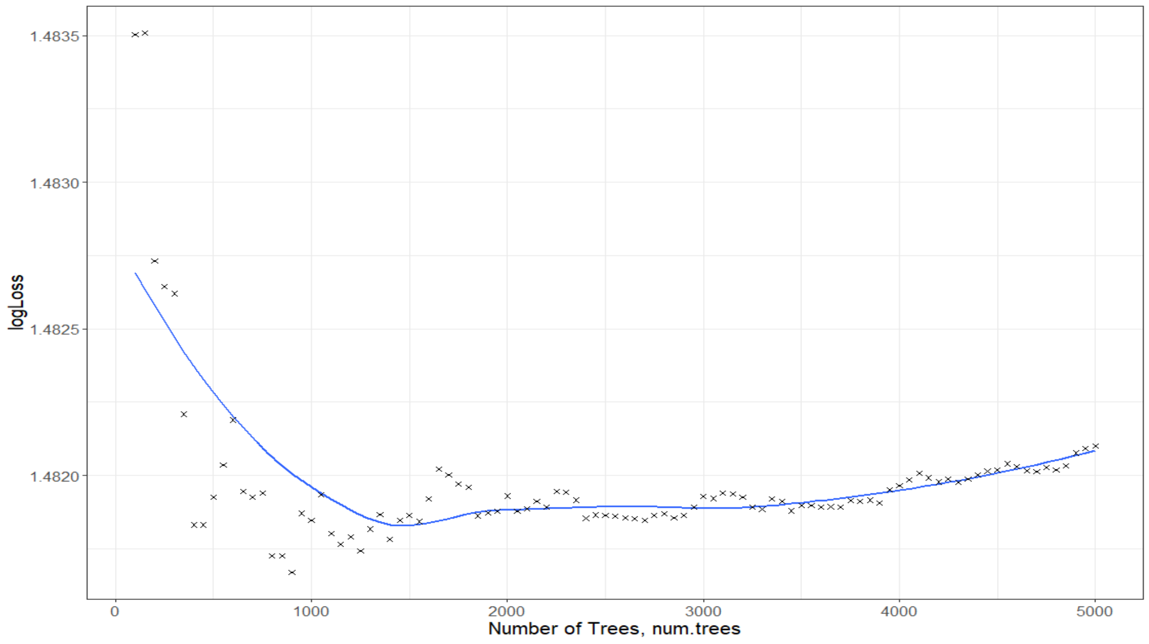

Figure 2.3.4.1: Plot to find the optimal total number of trees

The optimal random forests model will use $num.trees = 1500$, since in figure 2.3.4.1 the logLoss value is minimum around 1500 classification trees. To conclude, the final tuning parameters are $mtry = 2$ and $num.trees = 1500$. Chapter 8 will include a comparison of results of all the methods considered.

Chapter 3

Proportional-Odds Logistic Regression

One possible data modeling approach for grade prediction is the proportional-odds logistic regression method (POLR). Models like POLR are designed for our output variable (G) which is ordered categorical responses [14]. Based on the definition obtained in [17], we define the POLR model below. Note that training from chapter 2 produced an optimal model using the *logit* link, which is implemented into the definition.

Definition. Let Y be an ordinal output variable with M classes, or referred to as categories. Then, the cumulative probability of Y less than or equal to a specific class level $m = 1, \ldots, M-1$, is given by $P(Y \leq m)$. Note that $P(Y \leq M) = 1$. Then, the **odds**, or ratio, of being less than or equal to a particular class is given by

$$\frac{P(Y \leq m)}{P(Y > m)}. \tag{3.0.1}$$

Now, the **logit**, or **log odds**, is given by

$$\begin{aligned} logit[P(Y \leq m)] &= ln\left(\frac{P(Y \leq m)}{P(Y > m)}\right), \\ &= \beta_{0,m} + \beta_{1,m}X_1 + \beta_{2,m}X_2 + \cdots + \beta_{k,m}X_k, \end{aligned} \tag{3.0.2}$$

where k is the number of inputs in the data set.

We only consider the classes $m = 1, \dots, M-1$ since

$$P(Y > M) = 1 - P(Y \leq M) = 1 - 1 = 0,$$

which will lead to undefined odds in equation (3.0.1). In our case, grades consists of $M = 5$ classes with class levels $m = 1, \dots, 4$, excluding $m = 5$ because $P(Y \leq 5) = 1$. Hence,

$$Y = \begin{cases} \text{level 1,} & \text{if the grade is an EW,} \\ \text{level 2,} & \text{if the grade is a D or lower,} \\ \text{level 3,} & \text{if the grade is a C or lower,} \\ \text{level 4,} & \text{if the grade is a B or lower,} \\ \text{level 5,} & \text{if the grade is an A or lower.} \end{cases} \tag{3.0.3}$$

In other words, for grades EW, D, C, B, and A we have $Y = 1, 2, 3, 4, \text{and} \, 5$, respectively. It makes sense that $P(Y \leq 5) = 1$ since this equates to the probability of getting an A or lower, which will always be one.

The POLR method intuitively has a proportional-odds assumption between each class in the output. In equation (3.0.2), the proportional odds assumption, or parallel regression assumption [17], describes the coefficients, $\beta_{i,m}$ for $i = 1, 2, \dots, k$, of the *logit* model as being the same at each class level. Or, the odds between each cumulative probability link is equal. Therefore, equivalently equation (3.0.2) becomes

$$logit[P(Y \leq m)] = \beta_{0,m} + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k. \tag{3.0.4}$$

Using the 'MASS' package [19], we can fit the POLR model using the function 'polr'. As a generalized notation cannot be reached, R uses the following *logit* format from [17]:

$$\begin{aligned} logit[P(Y \leq m)] &= \beta_{0,m} - \eta_1 X_1 - \eta_2 X_2 - \cdots - \eta_k X_k, \\ &= \zeta_m - \eta. \end{aligned} \tag{3.0.5}$$

Here, we see that $-\eta_i = \beta_i$ for $i = 1, 2, \ldots, k$, where ζ_m is used to represents the intercepts, and η is used to represent the estimated coefficients. The η estimated coefficients can be calculated using maximum likelihood estimates, which are implemented in R. To find the inverse of the logit link function, we begin by setting $logit[P(Y \leq m)] = log_e\big(\frac{P(Y \leq m)}{1-P(Y \leq m)}\big) = \zeta_m - \eta$, then,

$$\begin{aligned}
ln\left(\frac{P(Y \leq m)}{1 - P(Y \leq m)}\right) &= \zeta_m - \eta, \\
\frac{P(Y \leq m)}{1 - P(Y \leq m)} &= e^{\zeta_m - \eta}, \\
P(Y \leq m) &= \frac{e^{\zeta_m - \eta}}{1 + e^{\zeta_m - \eta}}, \\
P(Y \leq m) &= \frac{1}{1 + e^{-(\zeta_m - \eta)}}.
\end{aligned}$$

Finally, we get $logit^{-1}[P(Y \leq m)] = P(Y \leq m) = 1/(1 + e^{-(\zeta_m - \eta)})$. From the inverse logit, we can find out the probability of a getting a certain grade or below based on the levels (7.0.1). Additionally, we can exponentiate both sides of equation (3.0.5) to get

$$\begin{aligned}
\frac{P(Y \leq m)}{1 - P(Y \leq m)} &= exp[\zeta_m - \eta], \\
&= exp(\zeta_m)exp(-\eta). \qquad (3.0.6)
\end{aligned}$$

The ratio (3.0.6) is the called the **odds**. We observe that the POLR model is an additive model using (3.0.5) but also a multipicative model for the odds, as seen in the ratio (3.0.6), [14].

3.1 Implementing POLR in R

Using the 'MASS' package [19], we can implement proportional odds logistic regression where the R code can be found in appendix B. We can now use the

‘Grades.testing’ portion of the data set, which is the portion the data set meant for testing and is seperate from the training set to avoid overtraining. This is used to obtain an accurate comparison of the models later. The following output are the results from the ‘polr’ function, which I defined in *R* as ‘Grades.polr’.

```
> summary(Grades.polr)
Call:
polr(formula=Grade~Semester+Size+StartTime+Day+Class+
    YrSince2003, data=Grades.testing, Hess=TRUE,
    method="logistic")

Coefficients:
                   Value  Std. Error t value
SemesterSpring -0.12325   0.059286 -2.0789
Size            0.01187   0.006776  1.7518
StartTime08:30 -0.38526   0.347003 -1.1102
StartTime09:30 -0.22228   0.340185 -0.6534
StartTime10:30 -0.12748   0.344215 -0.3703
StartTime11:30 -0.11453   0.342200 -0.3347
StartTime12:30 -0.12909   0.346552 -0.3725
StartTime13:30 -0.07864   0.345196 -0.2278
StartTime14:30 -0.20326   0.303381 -0.6700
Day             0.13213   0.238469  0.5541
ClassMAT250    -0.18119   0.248163 -0.7301
YrSince2003     0.02851   0.006344  4.4951

Intercepts:
     Value   Std. Error t value
EW|D -0.6259  0.8958    -0.6987
D|C  -0.0833  0.8957    -0.0930
C|B   0.8052  0.8958     0.8989
B|A   1.9372  0.8962     2.1616
```

In the POLR *R* output, the term ‘EW | D’ corresponds to level 1; the term ‘D | C’ corresponds to level 2; the term ‘C | B’ correspondes to level 3; and the term ‘B | A’ corresponds to level 4 of (7.0.1). Equivalently, the intercepts correspond to ζ_m for $m = 1, 2, 3, 4$. To incorporate the categorical inputs, the POLR model uses dummy variables. We can see that for the input ‘Semester’ there are 2 levels, “Spring” and “Fall”, so we see one indictor variable “SemesterSpring”. In other words, in the final *logit* link equations the fall semester will be treated as a zero, and the spring semester will be treated as a one. Similarily, for the 'Class' input variable we identify 'MAT250' as a one and 'MAT135' as a zero. Lastly, for the input 'StartTime' we have 8 levels where the class time $08:00$ am is indicated with a zero, and the remaining start times have indicators $1, 2, \ldots, 7$, leading to 12 inputs including indicators.

The estimated values for the coefficients are given in terms of ordered log odds [18]. For example, for 'SemesterSpring' we would say that for a one-unit increase in the semester (i.e., going from 0 to 1, or, fall to spring), we expect a 0.12325 decrease in the cumulative probability of the student's grade on the log odds scale, given all of the other variables in the model are held constant. Or, if we consider $exp(-0.12325) = 0.88404$ with reciprocal 1.13117 (five decimal places), this tells us that fall students have 1.1318 better odds of having a better grade compared to spring students.

For convenience, I will set the spring semester as X_1; the class size value as X_2; the 'StartTime' as $X_3, X_4, \ldots, X_9$ for each of the start times starting at $08:30$ am up until $02:30$ pm; the number of days the class meets as X_{10}; the class being a calculus I class as X_{11}; and the years since 2003 when the data began as X_{12}. Since the POLR model assumes the parallel regression assumption, the coefficients are the same for each *logit* link at each level (7.0.1). So, the η term is the same for each link and is given by (3.1.1)

$$\begin{aligned}\eta = &-0.12325\,X_1 + 0.01187\,X_2 - 0.38526\,X_3 - 0.22228\,X_4 - 0.12748\,X_5\\ &- 0.11453\,X_6 - 0.12909\,X_7 - 0.07864\,X_8 - 0.20326\,X_9 + 0.13213\,X_{10}\\ &- 0.18119\,X_{11} + 0.02851\,X_{12}. \end{aligned} \tag{3.1.1}$$

Using the η term, (3.1.1), and the intercepts, we have the POLR equations as follows:

$$logit[P(Y \le 1)] = -0.6259 - \eta,$$
$$logit[P(Y \le 2)] = -0.0833 - \eta,$$
$$logit[P(Y \le 3)] = 0.8052 - \eta,$$
$$logit[P(Y \le 4)] = 1.9372 - \eta.$$

Notice that the η term will change sign in the *logit* equations. Effectively, we are modeling the probability of getting a certain grade (or lower) as opposed to getting the grades above it. For example, the probability $P(Y \le 2)$ means the probability of getting a grade of "EW" or "D" versus getting a grade of "C" or above.

To demonstrate the interpretation of results, we will consider the case when we have a 'MAT135' class in the spring semester in 2003; with a start time of 08 : 30 am that meets 4 days a week; and a class with 44 students. Then, $X_1 = 1$, $X_2 = 44$, $X_3 = 1$, $X_4 = X_5 = \cdots = X_9 = 0$, $X_{10} = 4$, $X_{11} = 0$, and $X_{12} = 0$. Let's consider predicting a grade of D (or lower) as opposed to getting a C or above. Thus, we have $\eta = 0.45229$, $logit[P(Y \le 2)] = -0.0833 - 0.45229 = -0.53559$, and

$$P(Y \le 2) = logit^{-1}[-0.53559] = 1/(1 + e^{-(-0.53559)}) = 0.3692141.$$

In other words, about 36.92% of grades with the outlined terms received a D or worse.

```
  Semester Size StartTime Day  Class YrSince2003 Freq Grade
1   Spring   44     08:30   4 MAT135           0   11     A
2   Spring   44     08:30   4 MAT135           0   11     B
3   Spring   44     08:30   4 MAT135           0    8     C
4   Spring   44     08:30   4 MAT135           0    2     D
5   Spring   44     08:30   4 MAT135           0   12    EW
```

Above are the five observations of the actual data set under our input variables defined for the interpretation example. Hence, the actual grades with these inputs were $A = 11$, $B = 11$, $C = 8$, $D = 2$, and $EW = 12$. The probability of receiving a D or lower equates to $(2 + 12)/(11 + 11 + 8 + 2 + 12) = 0.3182$, or 31.82%. The POLR model predicts the students' grade cumulative probability relatively well in this case. Note that this may not necessarily be the case for a different set of inputs. Similarly, we can calculate $P(Y \leq 1) = 0.2538487$; therefore, the predicted probability of getting exactly a D is $P(Y = 2) = P(Y \leq 2) - P(Y \leq 1) = 0.3692141 - 0.2538487 = 0.1153654$, or about 11.54%.

3.2 Prediction and logLoss for POLR

To assess the accuracy of our POLR results, we use the logLoss equation (2.1.1). Conveniently, we can use the 'mlogLoss' function in the 'ModelMetrics' package [20] to find the final logLoss value for the model. The R code to implement this is found in appendix B. Creating predicted probabilities for each student grade from 'Grades.testing', we have the following predicted probability results using the inputs from the example in the previous section.

```
> predict.polr[1,]
       EW         D         C         B         A
0.2371759 0.1113345 0.2168307 0.2360309 0.1986280
```

Notice that the probability of getting a D in the first row is 11.13%, which is what we computed in the previous section with some rounding errors. Finally, using the 'mlogLoss' function we get a logLoss value of 1.541234.

To demonstrate the calculation, if the actual grades from the testing set was an A, then the logLoss value is given by $-ln(0.1986280) = 1.616322$. The logLoss value of 1.541234 is the average of all cases in the data set.

Chapter 4

Vector Generalized Linear Model

Yee [21] states that classical regression models for categorical response, such as POLR and multinomial logit, can be readily handled by the vector generalized linear model (VGLM), an alternative that is similar to the POLR model. This approach is well-suited for data with an ordinal response variable such as a student's grade. There are several extensions and versions of VGLM, but for the purposes of this study, we will focus on the VGLMs exclusively. The 'polr' function is useful for fitting a proportional model; however, the VGAM package [15] offers alternatives to the proportionality with the 'vglm' function. From the training in chapter 2, we see that the optimal VGLM model still assumes proportional-odds; nevertheless, an outline of VGLM is below based on the definition given in [22].

Definition. Suppose we have output Y with M levels, as shown in (7.0.1), and suppose we have k input variables including indicators. Let the inputs $\mathbf{X}$ be given by $(X_1, X_2, \ldots, X_k)^T$, and let $\mathbf{B}$ be given by $(\boldsymbol{\beta_1}|\boldsymbol{\beta_2}|\ldots|\boldsymbol{\beta_{M-1}})$ – a k-by-$(M-1)$ matrix of unknown coefficients. Then, VGLMs are defined as a model for which the conditional distribution of output Y given intputs X is of the form

$$f(Y|X; \mathbf{B}, \phi) = h(Y, \eta_1, \eta_2, \ldots, \eta_{M-1}, \phi), \tag{4.0.1}$$

for some known function $h(\cdot)$, where η_m are the linear predictions and ϕ is an optional scaling parameter, which is ignored in this study. The m-th linear predictor is given by

$$\eta_m = \boldsymbol{\beta_m}^T \boldsymbol{X} = \alpha_m + \sum_{i=1}^{k} \beta_{i(m)} X_i, \quad m = 1, 2, \ldots, M-1. \tag{4.0.2}$$

Finally, the *logit* link, similar to chapter 2, is given by

$$logit[P(Y \leq m)] = \eta_m. \tag{4.0.3}$$

Yee [21] states that equation (4.0.2) shows that all the parameters may be potentially modelled as functions of X. VGLMs are like GLMs but allow for multiple linear predictors, which is helpful to predict our five-level ordinal output Y. The coefficients in $\boldsymbol{\beta_m}$ can be approximated using maximum likelihood estimation.

For our case, we have $k = 12$ predictors as we see in the POLR output from chapter 2. These 12 inputs include the indicators from the dummy variables. Also, from equation (7.0.1) we see that there are $M = 5$ levels with $m = 1, 2, 3, 4$. We can write the inputs for our data as $\mathbf{X} = (X_1, X_2, \ldots, X_{12})^T$, and we can write the coefficient matrix $\mathbf{B} = (\boldsymbol{\beta_1}|\boldsymbol{\beta_2}|\boldsymbol{\beta_3}|\boldsymbol{\beta_4})_{12\times 4}$, or

$$\mathbf{B} = \begin{pmatrix} \beta_{1(1)} & \beta_{1(2)} & \beta_{1(3)} & \beta_{1(4)} \\ \beta_{2(1)} & \beta_{2(2)} & \beta_{2(3)} & \beta_{2(4)} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{12(1)} & \beta_{12(2)} & \beta_{12(3)} & \beta_{12(4)} \end{pmatrix}.$$

In chapter 2, we saw that one of the parameters under training was the parallel assumption. In R, this is implemented using the 'cumulative' function in the 'VGAM' package with either $parallel = TRUE$ or $parallel = FALSE$. Cross validation indicated that the optimal VGLM model assumes proportional-odds. This means that the coefficients for each $logit[P(Y \leq m)]$ link in (4.0.3) are the same, whereas the intercepts will differ. Hence, in the matrix $\mathbf{B}$ we will have $\beta_{i(m)} = \beta_i$ for all $m = 1, 2, 3, 4$ and $i = 1, 2, \ldots, 12$, or, $\boldsymbol{\beta_m}^T = (\beta_1, \beta_2, \ldots, \beta_{12})$. Note that if $parallel = FALSE$ then for each link we would have a different equation for η_m, which the 'polr' function could not implement. Equation 4.0.3 becomes

$$logit[P(Y \leq m)] = \eta_m = \boldsymbol{\beta_m}^T\mathbf{X} = \alpha_m + \beta_1\, X_1 + \beta_2\, X_2 + \cdots + \beta_{12}\, X_{12}. \qquad (4.0.4)$$

4.1 Implementing VGLM in R

Using the ‘VGAM’ package [15], we can implement the vector generalized linear model using the R code in appendix C. Again, we use the ‘Grades.testing’ portion in this section, and we will use the ‘vglm’ function to apply the vector generalized linear model, which I defined in R as ‘Grades.vglm’.

```
> summary(Grades.vglm)

Call:
vglm(formula=Grade~Semester+Size+StartTime+Day+Class+
    YrSince2003, family=VGAM::cumulative(link="logit",
    parallel=TRUE), data=Grades.testing)

Coefficients:
                 Estimate Std. Error z value
(Intercept):1   -0.625973   0.901879  -0.694
(Intercept):2   -0.083342   0.901724  -0.092
(Intercept):3    0.805120   0.901790   0.893
(Intercept):4    1.937142   0.902221   2.147
SemesterSpring   0.123250   0.059293   2.079
Size            -0.011870   0.006720  -1.766
StartTime08:30   0.385285   0.350419   1.099
StartTime09:30   0.222301   0.343638   0.647
StartTime10:30   0.127507   0.347556   0.367
StartTime11:30   0.114558   0.345592   0.331
StartTime12:30   0.129123   0.350075   0.369
StartTime13:30   0.078666   0.348877   0.225
StartTime14:30   0.203294   0.306014   0.664
Day             -0.132130   0.241028  -0.548
ClassMAT250      0.181196   0.249830   0.725
YrSince2003     -0.028515   0.006317  -4.514

Names of linear predictors: logitlink(P[Y<=1]), logitlink(P[Y<=2]),
logitlink(P[Y<=3]), logitlink(P[Y<=4])
```

From the VGLM *R* output, the linear predictors are

$$\boldsymbol{\eta} = (logit[P(Y \leq 1)],\, logit[P(Y \leq 2)],\, logit[P(Y \leq 3)],\, logit[P(Y \leq 4)])^T,$$

and the input variables are given by $\boldsymbol{X} = (X_1, X_2, \ldots, X_{12})^T$, by setting the input variables as X_i in a convenient format like we did in chapter 2. Furthermore, we see that there is only one set of coefficients β_i due to the parallel assumption; otherwise, we would see a similar syntax like for the '(Intercept)' term, in which we would see $12 \times 4 = 56$ predictors, leading to a different set of coefficients for each linear predictor η_m.

The intercepts for each of the linear predictors are $\alpha_1 = -0.625973$, $\alpha_2 = -0.083342$, $\alpha_3 = 0.805120$, and $\alpha_4 = 1.937142$. Lastly, for each column $\boldsymbol{\beta_m}$ in $\mathbf{B}$ we have

$$\boldsymbol{\beta_m} = \begin{pmatrix} 0.123250 \\ -0.011870 \\ 0.385285 \\ \vdots \\ -0.028515 \end{pmatrix}, \quad \forall\, m = 1, 2, 3, 4.$$

Considering η_2, we have $logit[P(Y \leq 2)] = -0.083342 + \boldsymbol{\beta_2}^T\mathbf{X}$. The interpretation is very similiar to the POLR model: if we wish to calculate the proportion of student grades that are a D or lower in a introductory statistics class, 'MAT135'; in the spring semester in 2003; with a start time of $08:30$ am that meets 4 days a week; and the class that has 44 students; then $\mathbf{X} = (1, 44, 1, 0, 0, 0, 0, 0, 0, 4, 0, 0)$. Yielding, the linear predictor $logit[P(Y \leq 2)] = -0.083342 - 0.542265 = -0.625607$, and

$$P(Y \leq 2) = logit^{-1}[-0.625607] = 1/(1 + e^{-(-0.625607)}) = 0.3485073.$$

Therefore, the probability of getting a D or EW versus a C or above is 34.85%. In chapter 2, we saw that the actual probability of getting a D or lower is 31.82%, so the VGLM model predicts the probability relatively well under these conditions. Similarily, $P(Y \leq 1) = 0.2371736$ leading to a probability of getting exactly a D under the specified inputs is $P(Y = 2) = 0.3485073 - 0.2371736 = 0.1113337$, or 11.13%.

This is exactly the same as the POLR model, and note that the estimated coefficients are the same as the POLR model with opposite sign. Therefore, we expect the logLoss value of the testing set to be the same as the POLR model.

4.2 Prediction and logLoss for VGLM

To assess the accuracy of the VGLM model, we use the logLoss equation (2.1.1). Again, we can use the 'mlogLoss' function in the 'ModelMetrics' package [20] to find the final logLoss value for the model. The R code to implement this is found in appendix C. Creating predicted probabilities for each student grade for the same input values as in our example, we have the same probabilities as in the POLR model.

```
> predict.vglm[1,]
       EW         D         C         B         A
0.2371759 0.1113345 0.2168307 0.2360309 0.1986280
```

Notice that the probability of getting a D in the first row is 11.13%, as was computed in the previous section. We get a logLoss value of 1.541234 using the 'mlogLoss' function; note that this is identical to the POLR model fit in the previous chapter.

The logLoss value using $parallel = FALSE$ was 1.537636. This is close to the value 1.541234. Since there are less terms to consider and less estimated coefficients under the parallel assumption, we are inclined to use $parallel = TRUE$ as it is more interpretable.

Chapter 5

Classification and Regression Tree Model

The CART model is an abbreviation for the classification and regression tree model and is commonly referred to as recursive partitioning. This method was introduced by Leo Breiman et al. in 1984 [9], where Breiman exposes the data modeling culture as being a limited method compared to the algorithmic modeling culture. This is a strong statement which we will explore in this thesis. However, both cultures come with positives and negatives. Both cultures are concerned with the black box, which associates the input variables to the output variable. Typically, statisticians from the dominant data modeling culture do not think of data models, such as linear regression or logistic regression, as a black box since they know quite a bit about the assumptions and the mathematical properties of these models. In contrast, the algorithmic modeling culture deals with thinking outside the black box and finds patterns within the data using an algorithm. Since both cultures are concerned with predictive accuracy, Breiman is able to make relevant comparisons between the two cultures. Breiman discusses limitations to data modeling leading to worse accuracy, and provides a solution by introducing algorithmic modeling, particularly the CART model and later on the random forests model. However, questions of interpretability are just as important as accuracy when solving a problem or presenting solutions to a non-technical audience. Description of the CART model is primarily taken from

James et al.[8].

Breiman [1] evaluates the interpretability of the CART model as an 'A+'; however, in most cases the CART model scores a 'B' on prediction. Thus, a positive in the CART model is that it is highly interpretable, but a negative is that the prediction may not be any better than the models from the data modeling culture, such as those discussed in chapter 3 and 4. The aim of this chapter is to evaluate whether the CART model is suitable for our ordinal output and if it worth implementing compared to the POLR and VGLM models.

The CART model can be implemented on a regression problem, where the input Y is quantitative, or the CART model can be implemented on a classification problem, where the input Y is categorical. In our case the data is ordinal, so for the purposes of this study I will overview classification trees for the CART model.

James et al. [8] describes the CART model as segmenting the predictor space into J simple regions, denoted $R_1, R_2, \ldots, R_J$ and referred to as terminal nodes. The splitting rules, which we will define later, can be summarised in a tree, or commonly known as decision trees. For each region in a classification tree, the predicted response for an observation will be given by the most commonly occuring class, or in our case the most common grade in terms of percentage in each region. In fact, the classification tree will include the probability of each grade for that region. However, how do we determine these J regions? Typically, we use high-dimensional rectangles or boxes [8]. We find the regions $R_1, R_2, \ldots, R_J$ that minimize a classification error measure that is sensitive to the tree growth. Two available splitting criterion are the Gini index, G, and entropy, E.

Firstly, the Gini index is defined as

$$G = \sum_{m=1}^{M} \hat{p}_{jm}(1 - \hat{p}_{jm}), \tag{5.0.1}$$

where $0 \leq \hat{p}_{jm} \leq 1$ represents the probability or proportion of a particular class

$m = 1, 2, \ldots, M$ for observations in the j-th region. For small values of G, with $\hat{p}_{jm}$ close to zero or one, we define the region j as pure $-$ the region contains predominantly one class m.

Secondly, entropy (or Shannon's Diversity) is defined as

$$E = -\sum_{m=1}^{M} \hat{p}_{jm}\, log(\hat{p}_{jm}), \tag{5.0.2}$$

where $0 \leq \hat{p}_{jm}\, log(\hat{p}_{jm})$. Here, a $\hat{p}_{jm}$ close to zero or one gives an entropy close to zero $-$ the jth node will be thought of as pure as well. In fact, the Gini index and entropy are similar numerically [8].

It is computationally expensive to consider every possible segmentation of the predictor space; therefore we require a method called recursive binary partitioning. The 'binary' in recursive binary partioning refers to a left and right split from an initial node $-$ a node that contains all the observations in the data set. For our analysis, the initial node will include all observations from the testing set.

Each split creates a branch that goes further down the tree until we reach a terminal node R_j, for some j. For example, consider an input X_j and a cutpoint t that splits the predictor space into a branch $\{X|X_j < t\}$ and into a branch $\{X|X_j \geq t\}$, where t leads to the greatest possible reduction in G or E [8]. For each of the branches from the initial node to all subsequent nodes, we repeat the process considering all inputs $X_1, X_2, \ldots, X_k$, and all possible cutpoints t for each of the inputs, choosing the input and cutpoint combination that gives the lowest classification error, either G or E.

From the initial node we produce two nodes, then for each of these two nodes we find the best input and cutpoint that lowers the classification error G or E. This splits one of these subsequent nodes leading to three nodes in the tree, and we repeat binary splitting on one of these three nodes. This could be repeated to obtain an infinite tree depth. Typically, the stopping criterion for tree growth is when the terminal nodes have no more that five observations in each region, or for classification it is typically to stop when we only have one class left in the terminal node. The results using the testing set will be used as the demonstration.

To avoid overtraining and building a overly complex tree, we can build the full tree, T_0, and prune this tree, or cut back the tree to obtain a subtree. Complexity pruning is a method which uses a complexity parameter, cp, in which a sequence of trees are considered using this cp value, then a subtree with the lowest test error is used. In [8], the complexity parameter is refered to as α. Below is an outline of cross validation for chosing the cp and and building the tree on the testing set.

Algorithm 5.1 Training, Testing, and Fitting a Classification Tree

Training Set (using logLoss):

1. Apply K- fold cross validation to a sequence of cp values and select the cp value with the lowest logLoss.

Testing Set and Building the Tree (using the Gini index or Entropy):

2. Implement recursive binary partitioning to grow a full tree T_0, terminating when a node has one class left.
3. Apply the cp value from the training set to obtain the corresponding subtree T and the number of terminal nodes $|T|$. This is based on
$$G + cp\,|T| \quad or \quad E + cp\,|T|.$$
4. Return the subtree, $T \subset T_0$, from step 3 that corresponds to the cp value.

Note that the number of terminal nodes $|T|$ is given by $|T| = nsplit + 1$, where 'nsplit' is the number of splits there are in the tree. Also, a cp value of zero will lead to a very large overfitted tree as we will not have a penalizing '$cp\,|T|$' term in the algorithm, leading to the full tree T_0. In chapter 2, a control parameter *maxdepth* was considered, which controls the depth of the tree. Since the cp values can be thought of as a penalizing term for letting the tree grow, it makes sense that the training set showed that the choice of tree depth is unimportant since the cp value already controls for tree depth.

5.1 Implementing CART in *R*

Using the ‘rpart’ package [12], we can implement the CART model for our classification problem. Again, we use the designated ‘Grades.testing’ portion and the tuning parameters deduced from chapter 2 to build a classification tree. The R code implementing the CART model using the ‘rpart’ function can be be found in appendix D. The optimal cp value used is 0.001869883, with the default maxdepth of 30 since it was deemed arbitrary. Using the Gini index and finding the resulting subtree given a cp of 0.001869883 can be calculated using R with the following results.

```
> Grades.cart$cptable
           CP nsplit rel error    xerror        xstd
1 0.008594019      0 1.0000000 1.0000000 0.009727934
2 0.006073106      1 0.9914060 1.0044689 0.009692125
3 0.005156411      4 0.9731867 1.0010313 0.009719727
4 0.003093847      5 0.9680303 0.9948436 0.009768455
5 0.002406325      6 0.9649364 0.9948436 0.009768455
6 0.001869883      9 0.9577174 0.9948436 0.009768455
```

We see that the final cp value in the table above is the cp from training the CART model. The ‘rpart’ object ‘cptable’ prints a matrix of information on the optimal prunings based on a complexity parameter [12]. This is step 3 and 4 from algorithm 5.1. Therefore, the optimal tree has a cp of 0.001869883 with 9 splits and $|T| = 9 + 1 = 10$ terminal nodes, namely $R_1, R_2, \ldots, R_{10}$. The resulting classification tree is given in figure 5.1.1. Note that the resulting classification tree only considers the inputs ‘YrSince2003’, ‘Size’, ‘StartTime’, and ‘Semester’, whereas all the other inputs did not increase the node purity from the Gini index. Also, it appears that the probability of receiving a D in each node is never the highest probability.

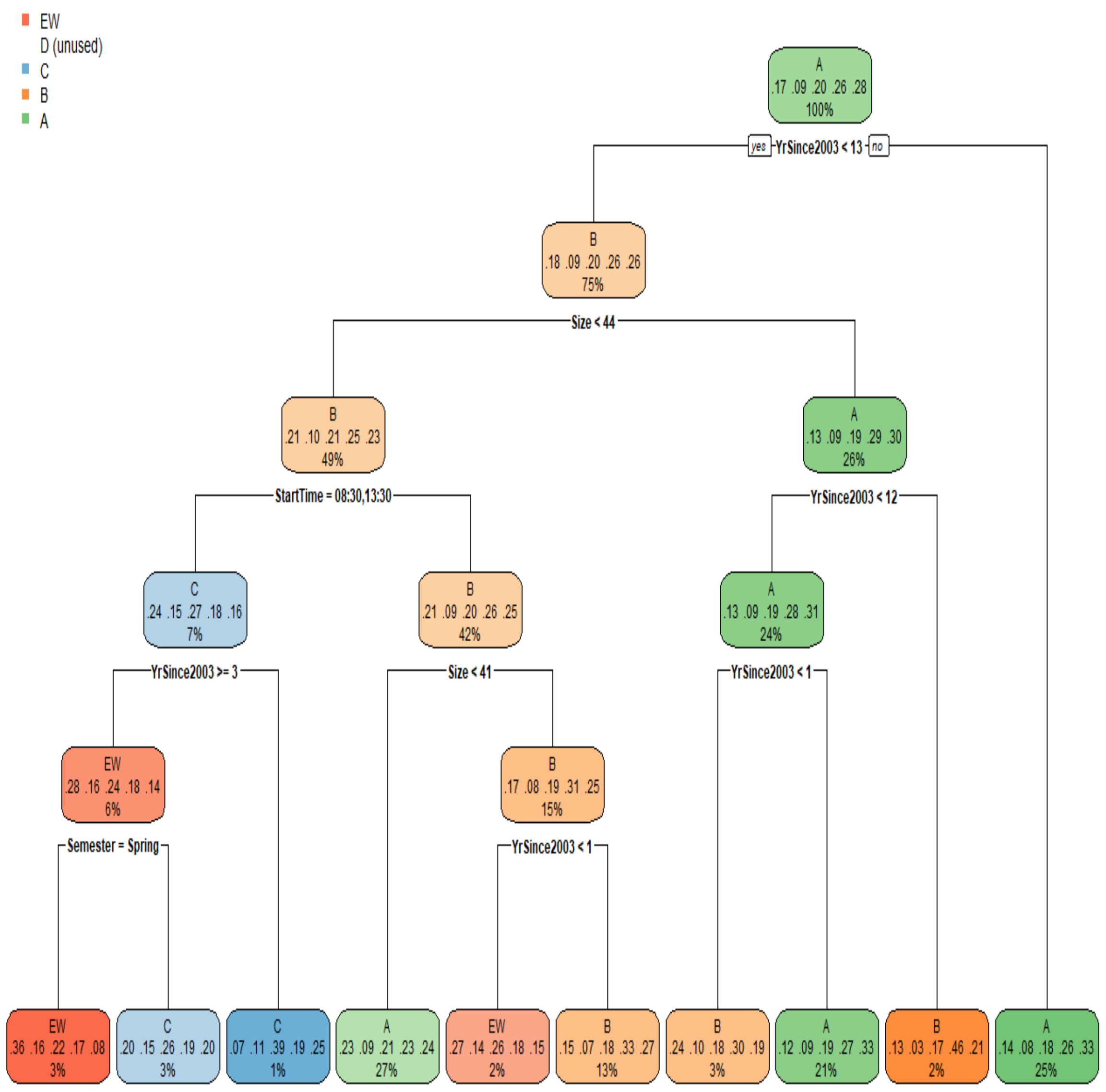
EW
D (unused)
C
B
A
A .17 .09 .20 .26 .28 100%
yes YrSince2003 < 13 no
B .18 .09 .20 .26 .26 75%
Size < 44
B .21 .10 .21 .25 .23 49%
A .13 .09 .19 .29 .30 26%
StartTime = 08:30,13:30
YrSince2003 < 12
C .24 .15 .27 .18 .16 7%
B .21 .09 .20 .26 .25 42%
A .13 .09 .19 .28 .31 24%
YrSince2003 >= 3
Size < 41
YrSince2003 < 1
EW .28 .16 .24 .18 .14 6%
B .17 .08 .19 .31 .25 15%
Semester = Spring
YrSince2003 < 1
EW .36 .16 .22 .17 .08 3%
C .20 .15 .26 .19 .20 3%
C .07 .11 .39 .19 .25 1%
A .23 .09 .21 .23 .24 27%
EW .27 .14 .26 .18 .15 2%
B .15 .07 .18 .33 .27 13%
B .24 .10 .18 .30 .19 3%
A .12 .09 .19 .27 .33 21%
B .13 .03 .17 .46 .21 2%
A .14 .08 .18 .26 .33 25%

Figure 5.1.1: CART model for cp=0.001869883 and $maxdepth = 5$

Each of the colored nodes in figure 5.1.1 contain three pieces of information working from the top of the node to the bottom: the most likely student grade; the probabilities of each grade; and the percentage of the data set that this observation represents, respectively. Denote the terminal nodes $R_1, R_2, \ldots, R_{10}$ moving from the bottom left to the bottom right of figure 5.1.1. In the middle of the node, the probabilities of each grade starts with the probability of EW on the left and ends with the probability of A on the right.

Looking at the initial the initial node, the recursive binary splitting to the left, indicating a ‘yes’, is $\{X \,|\, YearSince2003 < 13\}$, and the recursive binary splitting to the right, indicating a ‘no’, is $\{X \,|\, YrSince2003 \geq 14\}$.

Looking at the terminal node R_9, we observe that this node predicts a B, with a probability of 46%, which contains 2% of the ‘Grades.testing’ testing set. To reach the terminal node R_9, we have to have the years since 2003 be less than 13, the class size greater than or equal to 45, and the years since 2003 be, more specifically, greater than or equal to 12. This can be summarized as a class with 45 students or more, and there are exactly 12 years since 2003, i.e. the year is 2015. Equivalently, the terminal R_9 can be summarized as

$$R_9 = \{X \,|\, YearSince2003 = 12, \;\&\; Size \geq 45\}.$$

Consider the same conditions as for POLR and VGLM: a class ‘MAT135’ class in the spring of 2003; with a start time of $08:30$ am; that meets 4 days a week; and that contains 44 students. Since the classification tree above only uses four inputs we can summarize this as a class with 44 students; a start time of $08:30$ am; the years since 2003 as zero; and the class in the spring. If we following these rules down the classification tree, we will reach the terminal node R_7 which predicts that a student under these conditions will most likely get the grade B with a 30% probability. Also, the probability of getting an A is 19%, the probability of getting a C is 18%, the probability of getting a D is 10%, and the probability of getting an EW is 24%. This CART model is visually more interesting and easier to interpret, as we do not need to deal with interpreting *logit* links and cumulative probabilities.

5.2 Prediction and logLoss for CART

Just as before, we can use the ‘mlogloss’ function to generate the logLoss value of the CART models with a cp of 0.001869883 and a *maxdepth* of 5. The *R* code that implements this is found in appendix D. Creating predicted probabilities for each student grade for the same input values as our example, we have the results below.

```
> predict.cart[1,]
        EW          D          C          B          A
0.23880597 0.09701493 0.17910448 0.29850746 0.18656716
```

Notice that terminal node R_7 predicted the grade B with a 30% probability. Also, all the other probabilities are the same as in our example. Finally, we get a logLoss of 1.52279 for cp of 0.001869883 and $maxdepth = 5$. We see an improvement compared to the data models, as the POLR and VGLM models had a logLoss of 1.541234.

Chapter 6

Random Forests Tree Model

A problem associated with classification trees is that they have high variance [8]. In other words, a small change in the data set leads to a large change in the classification tree with different results. Also, the CART model can be defined as a weak learner: where introducing too many predictors to the model will cause inaccuracy, as if the algorithm gets overwhelmed. We will see that the RF model is a technique of combining a large number of weak learners to make a strong learner. The following overview of the random forests model is taken from [8] and [13].

The bootstrap method is used to fix the high variance in the CART model. Basically, bootstrapping averages a set of observations to reduce the variance. So, to reduce variance and increase accuracy one can take several training sets, build a model for each of the training sets, and then average the resulting predictions. Since we do not have access to several training sets, a technique called bagging, or bootstrapped aggregation, can be used.

For bagging, we take our data set, in our case the testing set, and select B, a number of random (bootstrap) samples with replacement, where each bootstrap will be the same size as the original data set. In fact, the number B is the total number of classification trees grown by the random forests model. For the b-th bootstrap we apply the model, obtain the prediction, and average the prediction results. For the RF model, we calculate the classification predictions $\hat{C}_1(\boldsymbol{X'}), \hat{C}_2(\boldsymbol{X'}), \ldots, \hat{C}_B(\boldsymbol{X'})$ on the B bootstraps, where $\boldsymbol{X'}$ are the inputs we wish to make a prediction. Each of the classification trees are grown without complexity pruning. This can be done by setting cp $= 0$ for the CART model, allowing each tree to have high variance and low bias.

Averaging reduces the variability of the low biased classification trees. The averaging part of bagging for a classification problem is done by taking a majority vote from each of the predictions $\hat{C}_b(\boldsymbol{X})$ for some bootstrap b. Majority voting is the overall prediction of the most commonly occuring class in the B bootstraps [8]. Bagging is a simplified method to bootstrapping as we take some number $n < N$, where N is the number of observations in the data set. A problem that arises with randomized samping is we may get a set of bootstraps that are similar, creating highly correlated trees. If we average correlated classification trees, we do not improve the variance issue. This problem is fixed in the RF model.

Random forests uses bagging and decorrelates the classification trees [8]. Instead of building a classification tree with k inputs under consideration for each node, a subset of predictors, namely $mtry$, are considered at each node. This avoids a computationally expensive model when building a large number of B trees. A common choice is $mtry \approx \sqrt{k}$. From chapter 2, cross validation determined that the number of input variables to consider at the splitting of each node is two, i.e. $mtry = 2$. The splitting rule used in the RF model is the Gini index, G from 5.0.1.

The RF model takes into account the strength of each of the inputs. From the CART model, the stronger inputs were 'StartTime', 'Size', 'Semester', and 'YrSince2003'. For each of the classification trees, the bootstrap sample takes into account the stronger inputs and these are used in the initial node split more frequently. These stronger inputs are the inputs that maximize node purity in the Gini index.

To summarise, the RF model builds B number classification trees, decorrelates the classification trees by only selection $mtry$ inputs to split at each node, and implements a majority vote to find the probabilities. Algorithm 6.1 outlines how to build RFs and is based on the work done in [13]. For algorithm 6.1, $\boldsymbol{X}$ are the inputs in the testing set.

Algorithm 6.1 Training, Testing, and Fitting the Random Forests model

```
Training Set (using logLoss):

  1. Apply $K$- fold cross validation to find the number of inputs to split at each
     node and the number of bootstraps $B = num.trees$.

Testing Set and Building the Tree (using the Gini index and logLoss):

  2. Obtain $B$ bootstraps from the testing set.

For $b = 1, 2, \ldots, B$:

  3. Obtain the corresponding classification tree $T_b$ for each bootstrap $b$ using the
     CART algorithm 5.1 with cp = 0.

       (a) Select a $mtry$ random number of inputs from $\boldsymbol{X}$.
       (b) Select which of the random inputs increase node purity using the Gini
           index.
       (c) Binary split the node using one of the inputs, and stop when there remains
           only one class left in the node, i.e. $min.node.size = 1$.

  4. Return all the classification trees for each bootsrap $b$: $\{T_b\}_1^B$.

Let $\boldsymbol{X'}$ be vector of input values that you want to use to make a prediction:

  5. Obtain the classification prediction $\hat{C}_b(\boldsymbol{X'})$ for each of the $b$-th random forest
     classification tree. Then,

         $\hat{C}_{rf}^{B}(\boldsymbol{X'}) = majority\ vote\ \hat{C}_b(\boldsymbol{X'})$.
```

Here, ‘forest’ refers to the fact that we are growing a large number of classifications trees. A downside to RF is we do not have any equations, such as POLR and VGLM, to use to make predictions on paper. We require a statistical program to make predictions using the RF model. Therefore, bagging improves the prediction accuracy at the expense of interpretability [8]. Note that after a probability distribution is created using $\hat{C}^B_{rf}(\boldsymbol{X'})$ in step 5, we can test the random forest model against the actual grades obtained from the ‘datatest’ set, use the logLoss equation 2.1.1, and find the final logLoss value of the model.

6.1 Implementing Random Forests in *R*

First, the RF model is built with *R*, then that model is used to make predictions. The RF model was built using the Gini index splitting rule, implemented by *splitrule* = “*gini*”; only allowing the classification trees T_b to terminate growth when there is only one class left in a node, implemented by *min.node.size* = 1; only allowing two random predictors to be considered in splitting each node, implemented by *mtry* = 2; and finally, setting the number of classification trees the RF model will grow as $B = num.trees = 1500$. It is not feasible to plot all of the 1500 trees, so the *R* program stores the model instead. To grow the random forest, we used the ‘ranger’ function in the ‘ranger’ package [23]. The output from the ‘ranger’ function, where the RF is stored in ‘Grades.rf’, is given below.

```
> print(Grades.rf)
Ranger result

Call:
 ranger(Grade~Semester+Size+StartTime+Day+Class
        +YrSince2003, data=Grades.testing, mtry = 2,
         splitrule="gini", importance="impurity",
         min.node.size=1, num.trees=1500,
         probability = TRUE, classification = TRUE)
```

```
Type:                              Probability estimation
Number of trees:                   1500
Sample size:                       4014
Number of independent variables:   6
Mtry:                              2
Target node size:                  1
Variable importance mode:          impurity
Splitrule:                         gini
```

The *importance* = *impurity* syntax in the 'ranger' function allows the RF to print out the variable importance. Since the RF model uses the strongest predictors in the top splits, a calculation of the importance of each input variable is given below.

```
> Grades.rf$variable.importance
   Semester        12.098243
   Size            52.624820
   StartTime       33.277749
   Day             5.829171
   Class           4.063113
   YrSince2003     53.669897
```

Just as we saw in the CART model, with 'YrSince' as the most important variable, the RF model uses 'YrSince2003', 'Size', 'StartTime', and 'Semester' as the strongest inputs, indicated by a larger number.

6.2 Prediction and logLoss for Random Forests

Lastly, to assess the accuracy of the RF method, we use the logLoss equation 2.1.1. Using the 'predict' function in the 'stats' package, we can find the probability distribution of the grades G.

```
> predict.rf$predictions[1,]
       EW          D          C          B          A
0.24693478 0.08180274 0.18950940 0.25930729 0.22244579
```

Finally, applying the 'mlogloss' function we return the final logLoss value of 1.460193. This is the lowest logLoss out of all the statistical models. We will discuss the preformance and merits of the various models in chapter 8.

Chapter 7

Trigonometric Functions and Fourier Series

For the numerical study, we will consider interpolatory trigonometric polynomials and Fast Fourier Transforms which will be a different approach than the statistical study. This approximation technique has been utilized in areas such as quantum mechanics and optics. The trigonometric interpolating polynomials will be used to evaluate certain scenarios within the data set. For instance, trigonometric interpolating polynomials will be constructed to model the proportion of students' grade (G) with respect to the start times in all spring 'MAT135' classes. Trigonometic interpolating polynomials fit large amounts of equally spaced data very well, and we used this to test the model on the non-traditional class start times: 02:30 pm and 08:00 am. We also investigated how the cyclic performance has been occuring over the years, that is, how much the proportion of students getting different grades repeat. After visual examination of the data plots, time periods that appear to model a complete cycle were selected. In the above treatments, the two inputs were chosen as they were significant factors in the statistical study. The theory of trigonometric polynomial interpolation is discussed in many numerical methods texts, for example [24].

Definition. The interpolatory trigonometric polynomial, $S_n(x)$, on a set of m data points $\{(x_j, y_j)\}_{j=0}^{m-1}$ with the assumption that $x_j = \frac{2\pi j}{m}$ is given by

$$S_n(x) = \begin{cases} \frac{a_0}{2} + \sum_{k=1}^{n}(a_k\, cos(kx) + b_k\, sin(kx)), & \text{if m = 2n+1,} \\ \frac{a_0}{2} + \sum_{k=1}^{n-1}(a_k\, cos(kx) + b_k\, sin(kx)) + \frac{a_n\, cos(nx)}{2}, & \text{if m = 2n,} \end{cases} \tag{7.0.1}$$

where

$$a_k = \frac{1}{m}\sum_{j=0}^{m-1} y_j\, cos(kx_j), \quad \text{and} \tag{7.0.2}$$

$$b_k = \frac{1}{m}\sum_{j=0}^{m-1} y_j\, sin(kx_j). \tag{7.0.3}$$

The interpolation of m data points requires approximately $O(m^2)$ multiplications and m^2 additions by direct computations. Since data generally consists of thousands of data points, the computation time is prohibitive in addition to round-off errors dominating the approximation. We use the Fast Fourier Transform (FFT) method which requires only $O(m\, log_2(m))$ multiplications and $O(m\, log_2(m))$ additions. The complex version of the trigonometric interpolating polynomial is

$$S_n(x) = \sum_{k=-n}^{n} z_k\, e^{ikx}, \quad \text{with } z_k \equiv z_{m-k}, \tag{7.0.4}$$

where the vector z is the Discrete Fourier Transform (DFT) of the data vector y with $z_k = \frac{1}{m}\sum_{j=1}^{m-1} y_j e^{-i2\pi jk/n}$. Thus the coefficients are given by

$$a_j = 2Re(z_j) = \frac{2}{m}\sum_{k=0}^{m-1} y_k\, cos(jx_k), \tag{7.0.5}$$

$$b_j = -2Im(z_j) = \frac{2}{m}\sum_{k=0}^{m-1} y_k\, sin(jx_k). \tag{7.0.6}$$

We start by investigating how the grade proportions vary with respect to start times. The models were constructed using the traditional start times; then, we determined if the non-traditional start times followed the cyclic behavior of the preceeding start times. Also, we evaluated whether this is a beneficial time to teach the specified class. The second investigation looks at how the grade proportions vary with respect to time since 2003. We assumed a complete cycle pattern between 2007 and 2013. We then extended this periodic trigonometric function in time and observed how well it models the rest of the data. The experiment was repeated for data from 2005 to 2011 and the two models were compared. To compare the two models, the error was calculated using the L2 norm (7.0.7) [24],

$$||\boldsymbol{x}||_2 = \sqrt{\sum_{i=1}^{m} x_i^2}. \tag{7.0.7}$$

The numerical simulations were carried out using *MATLAB*, which can be seen in appendix F.

7.1 Implementing Trigonometric Functions in *MATLAB* for the Start Times

Let us consider the 'MAT135' class in the fall. The trigonometric interpolating polynomial was computed using the traditional start times: 09:30 am, 10:30am, 11:30 am, 12:30 pm, and 1:30 pm. For each student's grade (G) we find the corresponding trigonometric interpolating polynomial based on the the traditional start times. We call these start times "traditional" because only recently did Murray State University offer the course outside of these times. The non-traditional start time is 02:30 pm, since a class at this time is uncommon for this particular class. We used this non-traditional start time to check the accuracy of our model. Similarily, for spring 'MAT135' and fall 'MAT250', we considered the non-traditional start time of 08:00 am. Spring 'MAT250' did not contain any non-traditional start times. Based on the accuracy of the models from the data with non-traditional start times, we may determine whether it is worth introducing a 'MAT250' class earlier or later during the day.

To demonstrate the trigonometric polynomials for figure 7.1.1, we used the A grade proportions.

‘StartTime’	09:30	10:30	11:30	12:30	13:30	14:30
Hours after 09:30 am	0	1	2	3	4	5
x	0	$\frac{2\pi}{5}$	$\frac{4\pi}{5}$	$\frac{6\pi}{5}$	$\frac{8\pi}{5}$	2π
Grade A actual proportions	0.3309	0.2984	0.2763	0.2556	0.3096	0.3171

Since the model was constructed using the first five start times, we have $m = 5$ and $n = \left\lfloor \frac{m}{2} \right\rfloor = \left\lfloor \frac{5}{2} \right\rfloor = 2$. We found the trigonometric interpolating polynomial for the proportion of A grades as

$$S_2(x) = 0.2941 + 0.0356\, cos(x) + 0.0012\, cos(2x) + 0.0007\, sin(x) - 0.0107\, sin(2x).$$

Therefore, the predicted proportion of A grades at 02:30 pm is $S_2(2\pi) = 0.3309$. Note that this is the same value as $S_2(0)$ since the trigonometric interpolating polynomials have period 2π. In other words, at 02:30 pm we have the same predicted proportion of A grades as the actual proportion of A grades at 09:30 am.

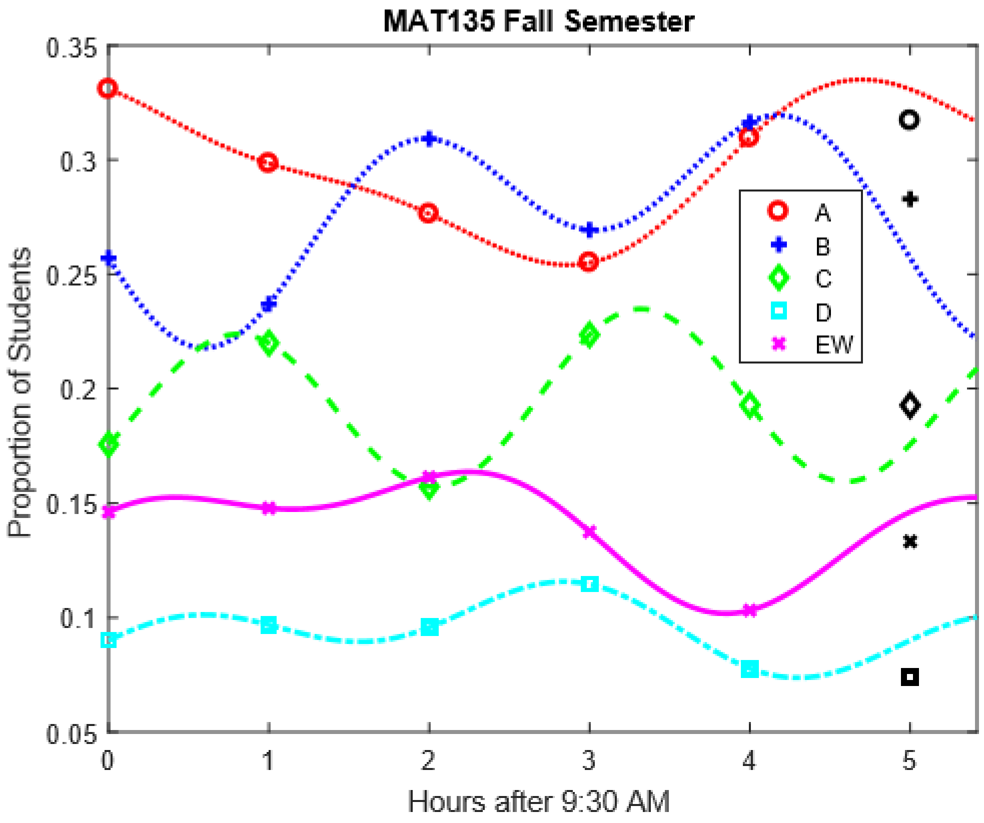

Figure 7.1.1: Proportion of student grades in 'MAT135' against the start times in the fall

Figure 7.1.1 includes the trigonometric interpolating polynomial for each students' grade computed using the traditional start times. Here, the zero on the x-axis represents the start time 09:30 am. The markers at time five represent the true proportion of the grades at 02:30 pm. At 2:30 pm, the model is predicting about 0.01 points higher in the proportion of A grades; about 0.01 points lower for the actual proportion of B grades; about 0.005 points lower for the actual proportion of C grades; about 0.005 points higher for the actual proportion of EW grades; and about 0.01 points higher for the actual proportion of D grades. The models for the proportion of A, B, C, and EW grades follow the trend of an increase or decrease in actual proportion; however, the model for the D grades predicts an increase when they actually decreased. Overall, the models accurately predict the trend in the proportions. Also, it would not be beneficial to offer a 'MAT135' class at 02:30 pm in the fall as the proportion of D and EW grades increase.

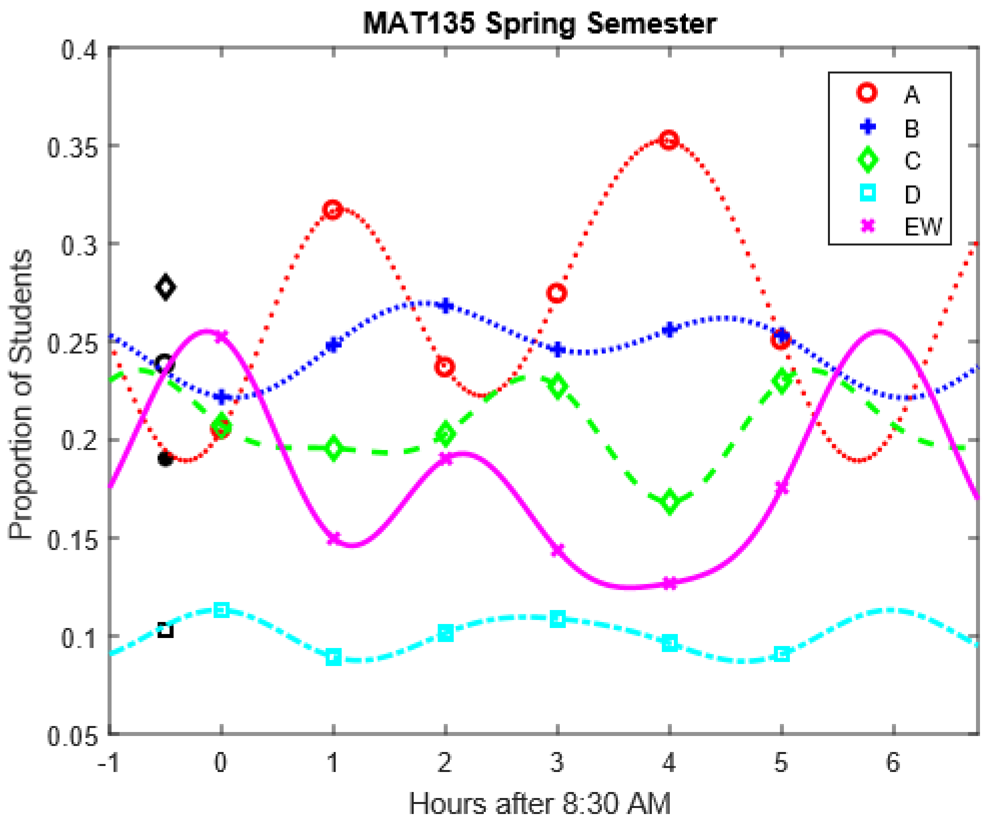

Figure 7.1.2: Proportion of student grades in 'MAT135' against the start times in the spring

Figure 7.1.2 modeled the proportion of grades in the spring 'MAT135' class. In this case, the non-traditional start time is 08:00 am. The model closely predicts the C and D grades; however, the model predicts about 0.04 points lower than the actual proportion of A grades and about 0.05 points higher than the actual proportion of B and EW grades. Overall, the models follow the trend in the actual proportion. The model predicts a decrease in the proportion of D and EW proportion rates, so it is worth introducing a 'MAT135' class at 08:00 am in the spring.

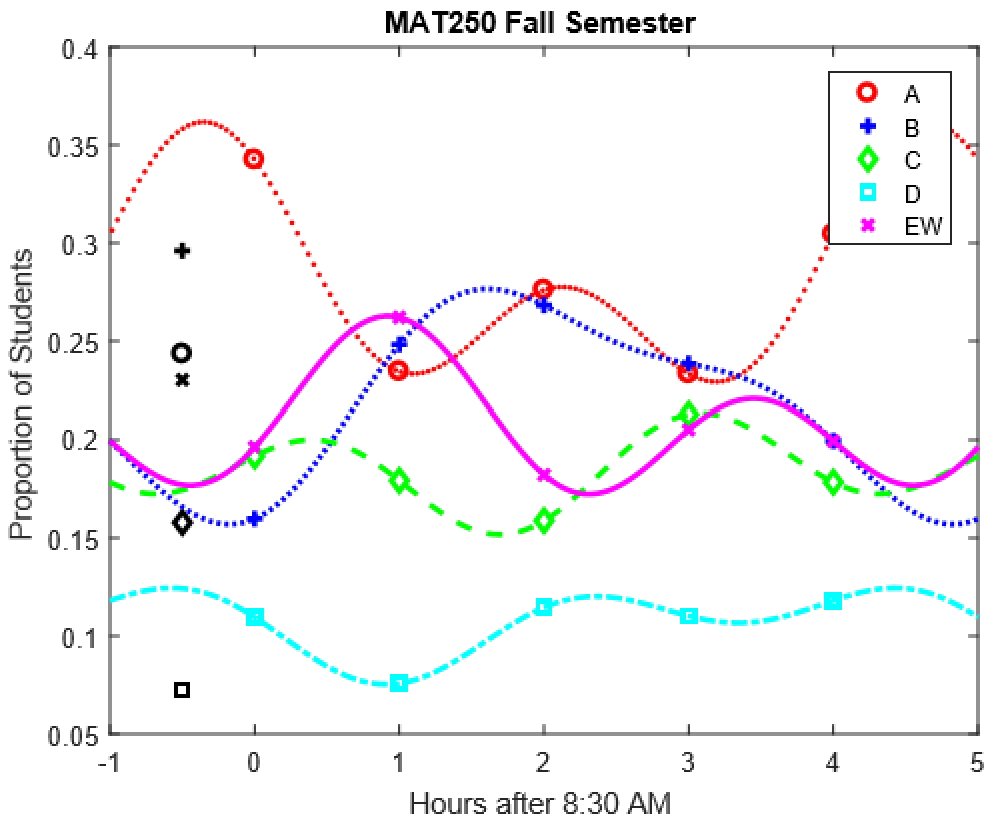

Figure 7.1.3: Proportion of student grades in 'MAT250' against the start times in the fall

Figure 7.1.1 modeled the proportion of grades in the fall 'MAT250' class. In this case, the non-traditional start time is 08:00 am. The model does not follow a trend for the A and EW grades. For the B, C, and D grades, the model severly underestimates or overestimates the grades. Since there is a high predicted proportion of A grades and a decrease in EW grades at 08:00 am, it would be beneficial to hold a 'MAT250' class at 08:00 am in the fall, according to the model.

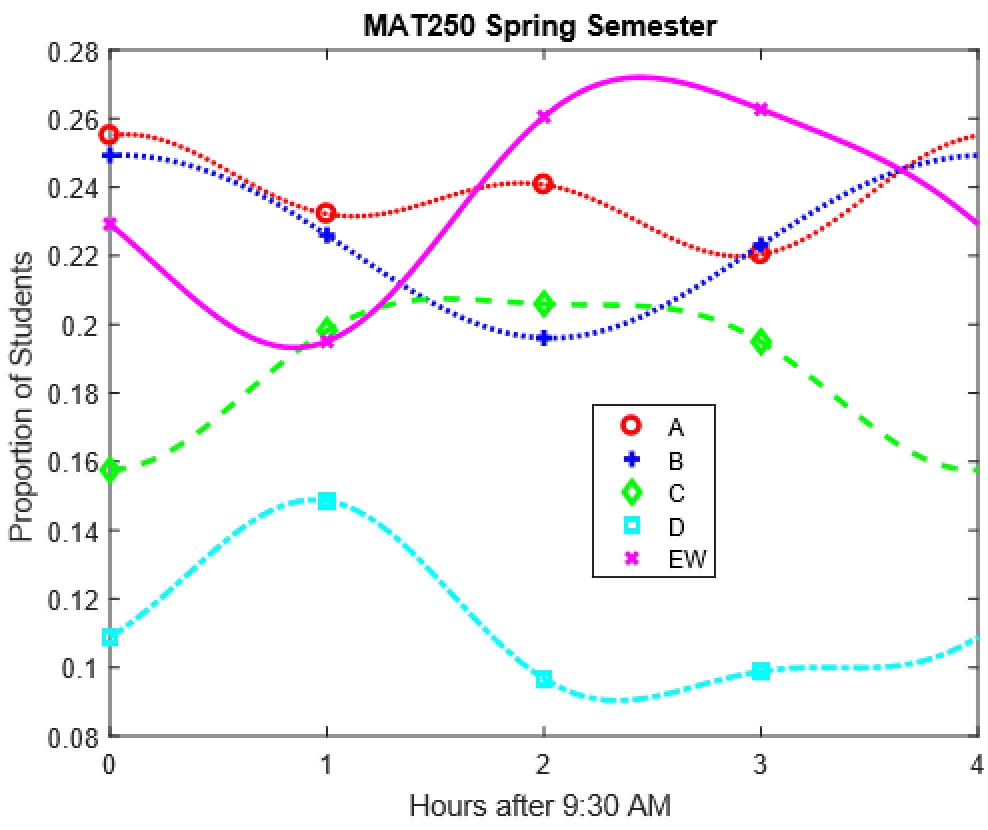

Figure 7.1.4: Proportion of student grades in'MAT250' against the start times in the spring

Figure 7.1.4 shows that students in a 'MAT250' spring class have a high proportion of EW grades with the proportion of A grades decreasing significantly after 11:30 am. These would be the worst times to attend a spring 'MAT250' class. The best times to attend a spring 'MAT250' class is at 09:30 am and 10:30 am when the proportion of A and B grades are at their highest levels, whereas the EW grades are at their lowest.

7.2 Implementing Trigonometric Functions in *MATLAB* for the Years Since 2003

When we visually assessed the cyclic behavior in the data, we determined that the best cyclic behavior appeared to be between 2007 and 2013. Also, an analysis of cyclic behavior between 2005 and 2011 was conducted, and we calculated the error via

L2 norms (7.0.7). Trigonometric interpolating polynomials are cyclic, and in general we expected about a six to seven year period.

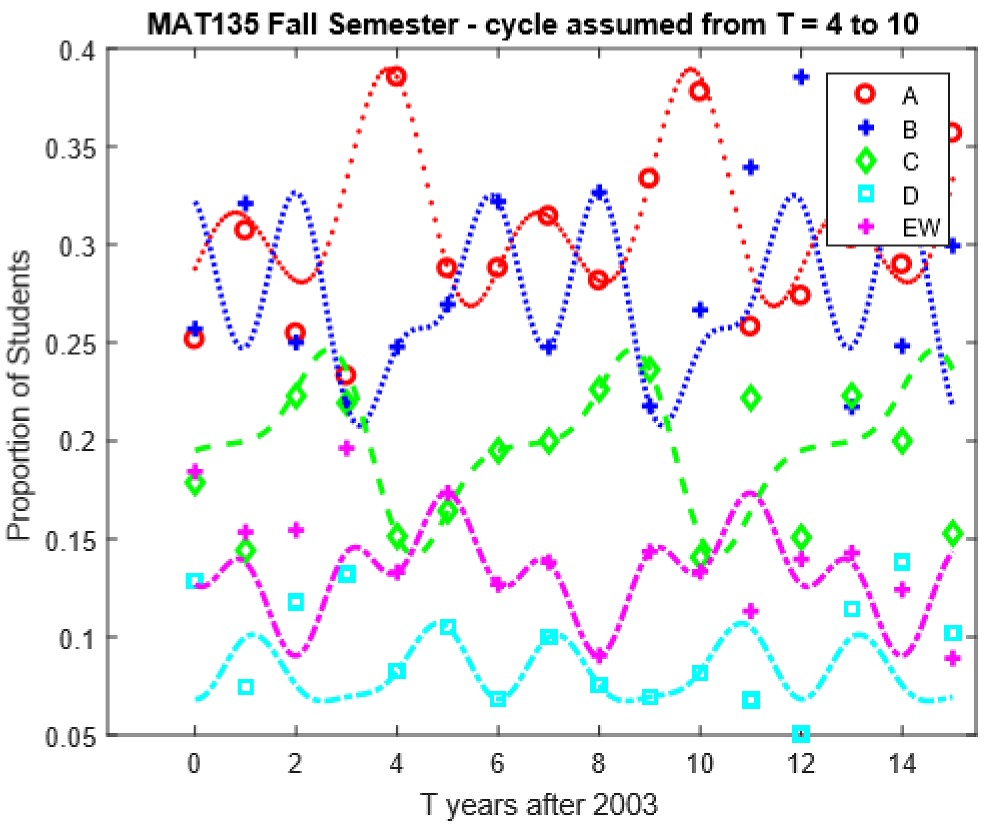

Figure 7.2.1: Proportion of student grades in ‘MAT135’ against the years since 2003 in the fall

Figure 7.2.1 is a plot of the trigonometric interpolating polynomials of the grade proportions for the fall ‘MAT135’ class. There appears to be a cycle between $T = 4$ and $T = 10$, that is, the years 2007 and 2013. The trend in the data suggests that the top ‘MAT135’ students appear in the years 2007 and 2013, or every 6 years, since the A grades peak and the C and EW grades decrease. Lastly, the model appears to represent the data well.

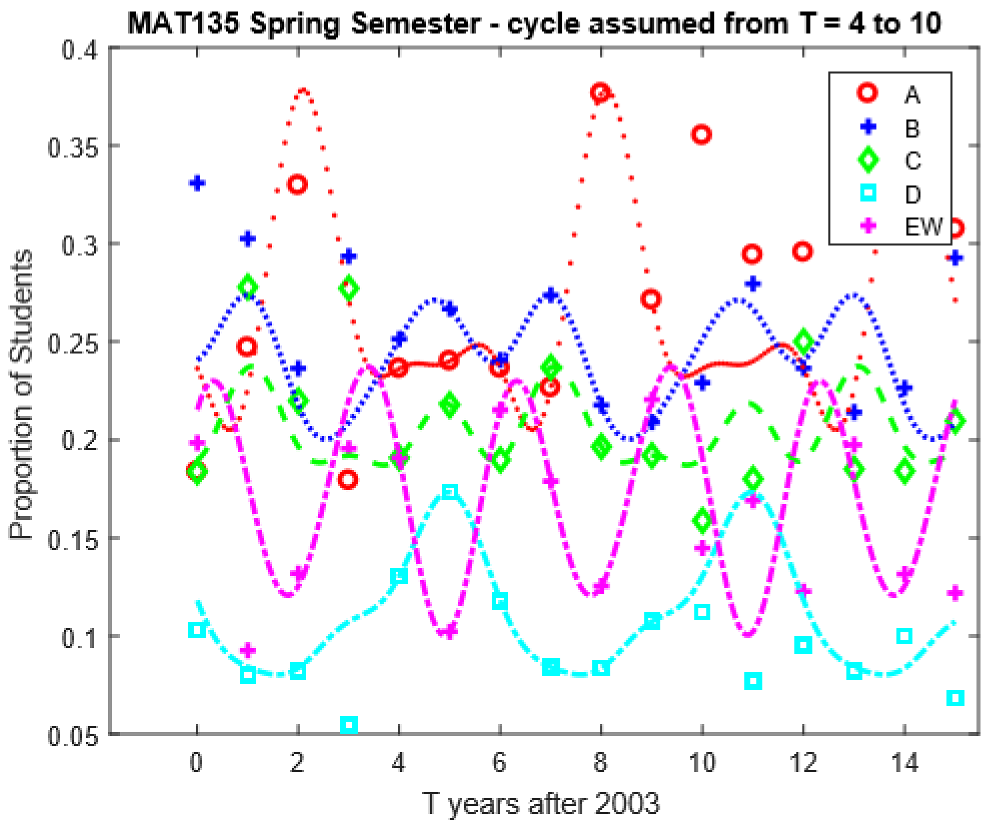

Figure 7.2.2: Proportion of student grades in ‘MAT135’ against the years since 2003 in the spring

Figure 7.2.2 suggests that the top spring ‘MAT135’ students occur in 2004 and 2011, in which the trend in the data suggests that roughly every seven years there is a large increase in A grades and a significant decrease in the EW grades.

Figure 7.2.3 suggests that the top fall ‘MAT250’ students occur in 2009 and 2015, in which the trend in the data suggests that roughly every 6 years there is a large increase in A grades and a significant decrease in the EW grades.

Lastly, figure 7.2.4 suggests that overall the students preform worse as the proportion of EW grades are large. In fact, the A, B, C, and EW grades follow similar trends. It appears that the top spring ‘MAT250’ students occur roughly in 2008 and 2015, in which the trend in the data suggest that roughly every seven years there is a peak in A grades and a decrease in the EW grades.

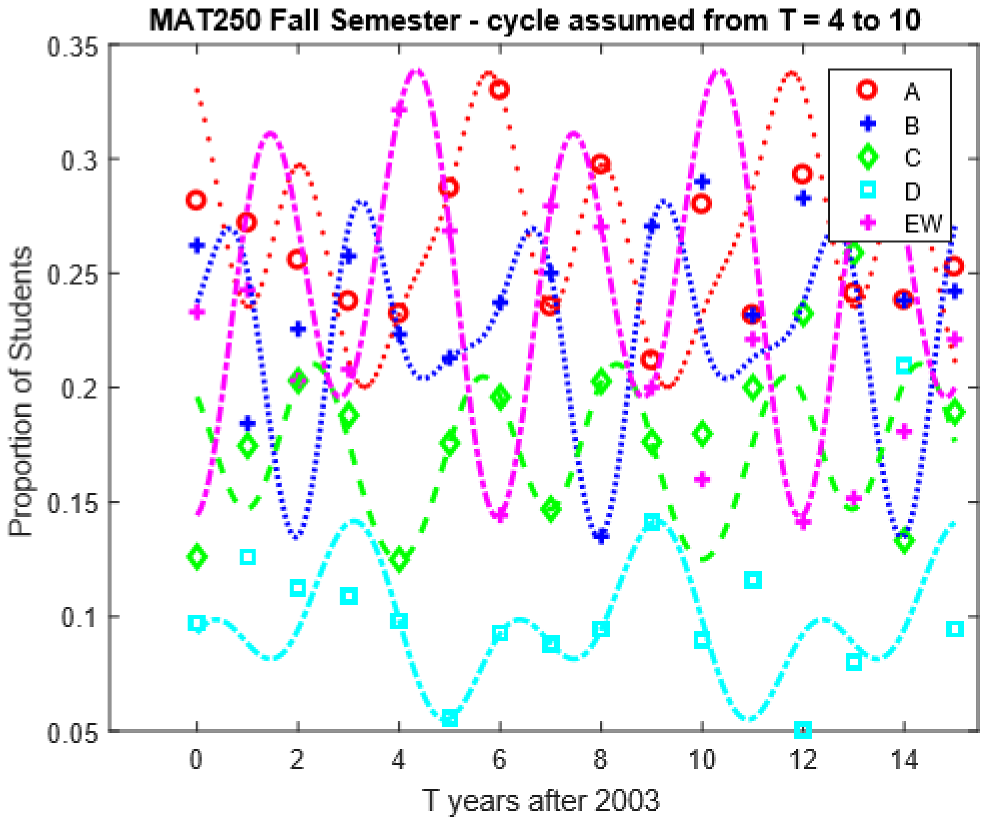

Figure 7.2.3: Proportion of student grades in 'MAT250' against the years since 2003 in the fall

Overall, the trends and cyclic behavior appears to be consistent in the data regardless of the class taught. This is shown by the models fitting the data fairly well in the same cycle of $T = 4$ and $T = 10$. Calculating the L2-norm errors for each grade proportions as well as the overall error we have the results in Table 7.1.

A	B	C	D	EW	Total Error	Class
0.1186	0.1963	0.1311	0.1302	0.1355	0.3242	Fall 'MAT135'
0.2122	0.1667	0.1367	0.1223	0.1833	0.3743	Spring 'MAT135'
0.1342	0.1786	0.1679	0.1541	0.2585	0.4107	Fall 'MAT250'
0.1428	0.1344	0.2250	0.1684	0.2114	0.4026	Spring 'MAT250'

Table 7.1: L2-norm error for cycle $T = 4$ and $T = 10$

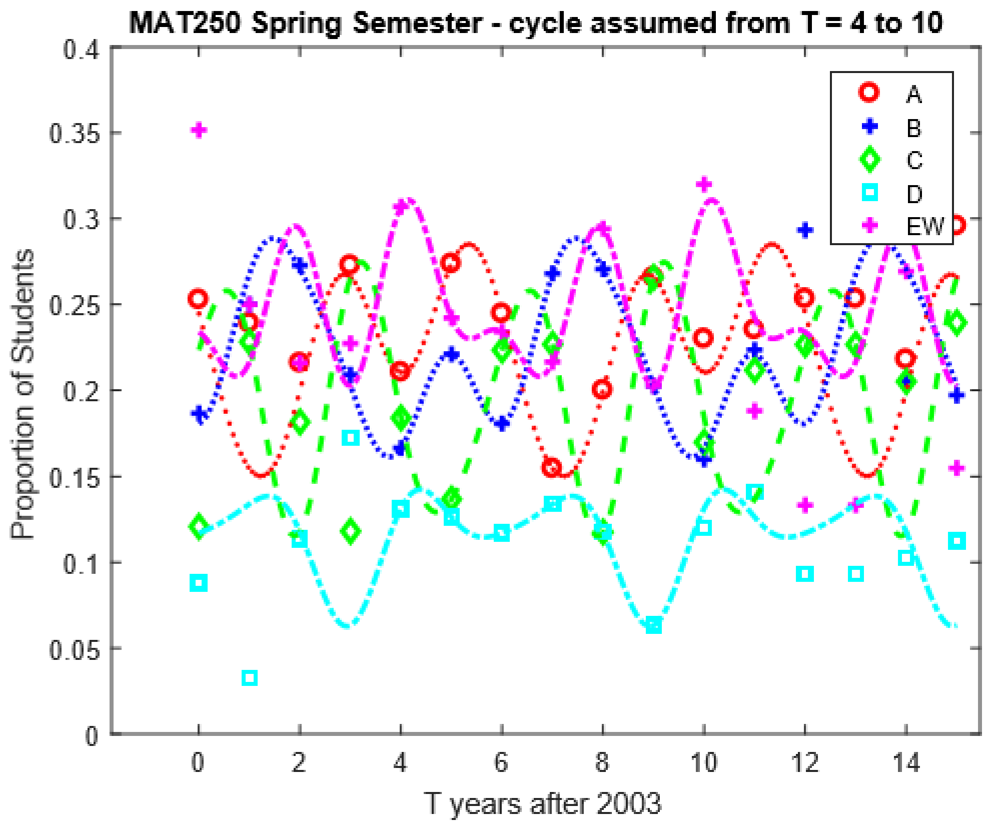

Figure 7.2.4: Proportion of student grades in‘MAT250’ against the years since 2003 in the spring

Similarly, calculating the L2-norm errors for each grade proportions as well as the overall error and assuming a cyclic pattern between $T = 2$ and $T = 8$, we have the results in Table 7.2.

A	B	C	D	EW	Total Error	Class
0.1733	0.1792	0.1203	0.1154	0.1634	0.3415	Fall ‘MAT135’
0.2460	0.1443	0.1550	0.1169	0.1712	0.3851	Spring ‘MAT135’
0.1157	0.1445	0.1675	0.1339	0.2430	0.3732	Fall ‘MAT250’
0.1404	0.1359	0.2398	0.1715	0.2234	0.4183	Spring ‘MAT250’

Table 7.2: L2-norm error for cycle $T = 2$ and $T = 8$

From Table 7.1 and 7.2 we deduce that the assumption of a cyclic behavior between $T = 4$ and $T = 10$ more accurately predicts student grades; however, the L2-norms are very similar, so assuming either cyclic behavior will follow the general trend.

Chapter 8

Comparing the Models

We now summarize the results of the thesis and briefly discuss alternative methods. Firstly, for the statistical study we have the following final logLoss values for the POLR, VGLM, CART, and RF model in Table 8.1.

POLR	1.5412
VGLM (parallel = TRUE)	1.5412
VGLM (parallel = FALSE)	1.5376
CART	1.5228
RF	1.4602
"Dumb" logLoss	1.6094
Non-informative logLoss	1.5448

Table 8.1: Final logLoss values using the testing set

The POLR, VGLM, and CART model have logLoss values that agree up to 1.5 (one decimal place). Therefore, these models are very similar in terms of performance. Since the CART model is more interpretable than the data models, we do not require prior knowledge of log odds and logistic regression to interpret the results, and the logLoss is slightly lower, we are inclined to choose the CART model to prognosticate student grades. Also, all the models produce logLoss values that are less than the "dumb" and non-informative logLoss values. Our models are performing better than assuming a uniform distribution of students' grades, or assuming a specified non-uniform distribution of grades. However, if the goal is to find the most accurate model that predicts the ordinal students' grades then we are inclined to choose the random forests model but lose the interpretability.

The caret package [11] offers a large range of classification and regression models that could be potential alternatives to the models used in this thesis. Some of the alternative models were different variations of the CART and RF models. We used the methods of cross validation and trained all of these models to determine if there were any other models that could be an attractive alternative. However, the models we selected gave the lowest logLoss by training the models of the training set. Also, since our data is not that large we could possible consider a cost-sensitive logistic regression that contains weights which account for unbalanced data sets.

The variables used in this thesis were easily accessible from Murray State's records; however, in a future study we could potentially collect data on which professor taught each class to evaluate their performances. Obtaining the instructors was less accessible from the data we collected; however, if we did obtain this information we could consider the instructors as a random effect in the models. Furthermore, we could investigate a linear mixed model variation for the ordinal regression models such as POLR and VGLM. Also, we could consider more mathematics and statistics courses or collect the data before 2003 to obtain a longer data set, and we could consider finals week exam times, previous GPA results on each individual, or previous high school data to make the data set wider. Lastly, we could consider alternative machine learning models that could out-perform the RF model, such as neural networks, to evaluate our ordinal output.

From the numerical study we ascertained if classes should be taught at specified non-traditional times. In a majority of the cases, the trigonometric interpolating polynomial models are a reasonable predictor of the proportion of student grade distributions. For a fall 'MAT135' class, it was not beneficial to introduce a class at 02:30 pm as the models predicted a drop in the proportion of A and B grades and a significant increase in the D and EW grades. The data refects this statement. For a spring 'MAT135' class, it is beneficial to hold an 08:00 am class as performance would improve compared to an 08:30 am class. For a fall 'MAT250' class, it is beneficial to hold an 08:00 am class. Lastly, for a spring 'MAT250' class, it is best to attend the 09:30 am and 10:30 am class.

From the random forests model we deduced that the 'Class' variable was not important, with an importance value of 4.0631, and the 'YrSince2003' was very important, with an importance value of 53.6699. This is reflected in the numerical study, as regardless of the class all the models followed an accurate cyclic pattern between 2007 and 2013, or 2005 and 2011.

In fact, there were 202 'MAT135' classes and 143 'MAT250' classes, which is likely due to the fact than the 'MAT135' classes are easier that the 'MAT250' classes. Also, discussions with faculty suggested that students with weaker algebra skills tend to be heavily penalized in 'MAT250' classes, whereas weaker statistical students do not get as heavily penalized in a 'MAT135' class. This leads to more students attending a 'MAT135' classes which has the results that the 'Size' input is statistically important. In recent years there has been an increase in statistical classes offered in high school and generally a decrease in algebraic skills, this leads to students being more prepared for a 'MAT135' class and less prepared for a 'MAT250' class. This would explain the statistical significance of the 'YrSince2003' input, in which statistical classes are producing more A grades and algebraic classes are producing more C, D, and EW grades.

Appendix A

R Code

Here is the *R* code for the thesis, separated into code for each chapter.

A.1 Data Set Up

```
#Importing Data
require(readxl)
require(tidyverse)
import.data <- read_excel("datafile.location.xlsx")
View(import.data)

#Fixing the time component in the data set
import.data.new <- import.data %>% #Calculus 1 (MAT250) &
                                   #Intro Statistics (MAT135)
  mutate(StartTime=format(as.POSIXct(import.data$StartTime,
                                     format='%m/%d/%Y %H:%M:%S'),
                          format='%H:%M')) #fix time component
View(import.data.new) #In Semester: 0=fall, 1=spring
str(import.data.new)

#Splitting into training and testing set
set.seed(1234)
trainind <- sort(sample(1:nrow(import.data.new),
                        size=floor(nrow(import.data.new)*(2/3))))
testind <- setdiff(1:nrow(import.data.new), trainind)
datatrain <- import.data.new[trainind,]
datatest <- import.data.new[testind,]

#Change the training data from "wide" to "long" format
TrainingLong <- datatrain %>%
  pivot_longer(cols=-c(Year,Semester,Section,Day,StartTime,
                       Class,Size),names_to="Grade",
               values_to="Freq") %>%
  mutate(Grade=factor(Grade,ordered=TRUE,levels=c("EW","D","C",
                                                  "B","A"))) %>%
  mutate(Section=factor(Section)) %>%  # treat as a factor
  mutate(Semester=ifelse(Semester==0,"Fall","Spring")) %>%
  mutate(YrSince2003=Year-2003) %>%
  mutate(StartTime=factor(StartTime)) %>%
  mutate(Class=factor(Class))
View(TrainingLong)

#Changing the training data into "longest" format
TrainingLongest <- TrainingLong %>%
  uncount(Freq)

#Change the testing data from ``wide" to ``long" format
TestingLong <- datatest %>%
  pivot_longer(cols=-c(Year,Semester,Section,Day,StartTime,
                       Class,Size),names_to="Grade",
               values_to="Freq") %>%
  mutate(Grade=factor(Grade,ordered=TRUE,levels=c("EW","D","C",
                                                   "B","A"))) %>%
  mutate(Section=factor(Section)) %>%  # treat as a factor
  mutate(Semester=ifelse(Semester==0,"Fall","Spring")) %>%
  mutate(YrSince2003=Year-2003) %>%
  mutate(StartTime=factor(StartTime)) %>%
  mutate(Class=factor(Class))
View(TestingLong)

#Changing the testing data into "longest" format
TestingLongest <- TestingLong %>%
  uncount(Freq)

#Selecting the appropriate inputs
library(tidyr)
Grades.training <- TrainingLongest[,c("Semester","Size","StartTime",
                                      "Day","Class","YrSince2003",
                                      "Grade")] %>%
  mutate(Semester=factor(Semester)) #make Semester a factor
Grades.training <- as.data.frame(Grades.training)
Grades.testing <- TestingLongest[,c("Semester","Size","StartTime",
                                    "Day","Class","YrSince2003",
                                    "Grade")] %>%
  mutate(Semester=factor(Semester)) #make Semester a factor
Grades.testing <- as.data.frame(Grades.testing)
```

A.2 Training the Models Code

```
#Non-informative logLos
p.A<-0.15
p.B<-0.3
```

```
p.C<-0.25
p.D<-0.10
p.EW<-0.20
 -(p.A*log(p.A))-(p.B*log(p.B))-(p.C*log(p.C))
                 -(p.D*log(p.D))-(p.EW*log(p.EW))
[1] 1.54448

#Implementing k-fold Cross Validation
require(caret)
set.seed(1234)
train_controlKFCV <- trainControl(method="cv",
                                  number=10,
                                  classProbs=TRUE,
                                  summaryFunction=mnLogLoss)

#Ordered Logistic or Probit Regression
require(MASS)
set.seed(1234)
tune.gridpolr <- expand.grid(method = c("logistic","probit"))
train.polr <- train(Grade~Semester+Size+StartTime+Day+Class+
                      YrSince2003,
                    data=Grades.training,
                    tuneGrid = tune.gridpolr,
                    trControl=train_controlKFCV,
                    method="polr",
                    metric="logLoss")
print(train.polr)

#Cumulative Probability Model for Ordinal Data
require(VGAM)
set.seed(1234)
train.vglm <- train(Grade~Semester+Size+StartTime+Day+Class+
                      YrSince2003,
                    data=Grades.training,
                    trControl=train_controlKFCV,
                    method="vglmCumulative",
                    metric="logLoss")
print(train.vglm)

#CART
require(rpart)
require(rpart)
set.seed(1234)
train.rpart <- train(Grade~Semester+Size+StartTime+Day+Class+
                      YrSince2003,
                    data=Grades.training,
                    trControl=train_controlKFCV,
                    tuneLength=100,
                    method="rpart",
                    metric="logLoss")
print(train.rpart)
train.rpart$finalModel$cp
# cp=0.001869883 w/ logLoss of 1.555764

#CART One SE Rule
require(rpart)
set.seed(1234)
train.rpart1SE <- train(Grade~Semester+Size+StartTime+Day+Class+
                       YrSince2003,
                     data=Grades.training,
                     trControl=train_controlKFCV,
                     tuneLength=100,
                     method="rpart1SE",
                     metric="logLoss")
print(train.rpart1SE)
train.rpart1SE$finalModel$cp
# cp=0 w/ logLoss of 1.558576

#Maxdepth controlling for cp=0.001869883
require(rpart)
require(magicfor)
magic_for(print, silent = TRUE)
for (i in seq(from=2,to=30,by=1)) {
  set.seed(1234)
  tune.gridrpart2<-expand.grid(maxdepth=i)
  train.rpart2 <- train(Grade~Semester+Size+StartTime+Day+Class+
                          YrSince2003,
                        data=Grades.training,
                        trControl=train_controlKFCV,
                        tuneGrid=tune.gridrpart2,
                        cp=0.001869883,
                        method="rpart2",
                        metric="logLoss")
  print(train.rpart2$results)
}
require(forcats)
train.cart.maxdepth <- magic_result_as_dataframe()
## maxdepth does not change

#Random Forests
require(ranger)
set.seed(1234)
tune.gridranger <- expand.grid(mtry = c(2:6),
                               splitrule="gini",
                               min.node.size=1)
train.rf <- train(Grade~Semester+Size+StartTime+Day+Class+
                    YrSince2003,
                  data=Grades.training,
                  trControl=train_controlKFCV,
                  tuneGrid=tune.gridranger,
                  method="ranger",
                  metric="logLoss")
print(train.rf)

require(magicfor)
magic_for(print, silent = TRUE)
for (i in seq(from=100,to=5000,by=50)) {
  set.seed(1234)
  Grades.rf.trees <- ranger(Grade~Semester+Size+StartTime+Day+
                              Class+YrSince2003,
                            case.weights=Grades.training$Freq,
                            data=Grades.training,
                            num.trees=i,
                            mtry=2,
                            splitrule="gini",
                            min.node.size=1,
                            probability=TRUE)
  predict.rf.trees <- stats::predict(Grades.rf.trees,
                                     data=Grades.training,
                                     type="response")
  print(ModelMetrics::mlogLoss(actual=Grades.training$Grade,
                               predicted=predict.rf.trees$predictions))
}
require(forcats)
train.rf.trees <- magic_result_as_dataframe()
colnames(train.rf.trees)
# Rename columns
names(train.rf.trees)[names(train.rf.trees) == "i"] <- "num.trees"
names(train.rf.trees)[names(train.rf.trees) == "ModelMetrics::
          mlogLoss(actual=Grades.training$Grade,
          predicted=predict.rf.trees$predictions)"] <- "logLoss"
view(train.rf.trees)
require(ggplot2)
ggplot(data=train.rf.trees,aes(train.rf.trees$num.trees,
                               train.rf.trees$logLoss))+
  theme_bw()+geom_smooth(method="loess",se=FALSE)+
  geom_point(shape=4)+
  xlab("Number of Trees, num.trees")+
  ylab("logLoss")+
  theme(text = element_text(size=15))
#low at about 1500 trees
```

Appendix B

R Code for POLR

```
#Testing and Implementing the 'polr' function in the 'MASS'
#package
require(MASS)
Grades.polr <- polr(Grade~Semester+Size+StartTime+Day+Class+
                      YrSince2003,
                    data = Grades.testing ,
                    method ="logistic",
                    Hess = TRUE)
summary(Grades.polr)

#Example
eta<--0.21325*(1)+0.01187*(44)-0.38526*(1)+0.13213*(4)
logit.d.or.lower<--0.0833-eta
d.or.lower<-1/(1+exp(-logit.d.or.lower))
logit.ew.or.lower<--0.6259-eta
ew.or.lower<-1/(1+exp(-logit.ew.or.lower))

#Prediction and final logLoss value of the model
require(stats)
predict.polr <- predict(Grades.polr,newdata=Grades.testing,
                        type = "probs")
predict.polr[1,]
require(ModelMetrics)
mlogLoss(actual = Grades.testing$Grade , predicted = predict.polr)
# [1] 1.541234
```

Appendix C

R Code for VGLM

```
#Testing and Implementing the 'vglm' function in the 'VGAM'
#package
require(VGAM)
Grades.vglm <- vglm(Grade~Semester+Size+StartTime+Day+Class+
                      YrSince2003,
                    data=Grades.testing,
                    family = VGAM::cumulative(link = "logit",
                                              parallel = TRUE))
summary(Grades.vglm)

#Example
alpha.1<--0.625973
alpha.2<--0.083342
beta.m<-c(0.123250,-0.011870,0.385285,0.222301,0.127507,0.114558,
          0.129123,0.078666,0.203294,-0.132130,0.181196,-0.028515)
x.example<-c(1,44,1,0,0,0,0,0,0,4,0,0)
etaD<-alpha.2+sum(beta.m*x.example)
prob.D.lower<-1/(1+exp(-1*etaD))
etaEW<-alpha.1+sum(beta.m*x.example)
prob.EW.lower<-1/(1+exp(-1*etaEW))
prob.d<-prob.D.lower-prob.EW.lower
prob.d

#Prediction and final logLoss value of the model
require(stats)
predict.vglm <- predict(Grades.vglm,newdate=Grades.testing,
                         type="response")
predict.vglm[1,]
require(ModelMetrics)
mlogLoss(actual=Grades.testing$Grade,predicted=predict.vglm)
# [1] 1.541234
```

Appendix D

R Code for CART

```
#Testing and Implementing the 'rpart' function in the 'rpart'
#package for cp=0.001869883
require(rpart)
Grades.cart <- rpart(Grade~Semester+Size+StartTime+Day+Class+
                       YrSince2003,
                     data=Grades.testing,
                     cp=0.001869883,
                     maxdepth=5)
print(Grades.cart)
Grades.cart$cptable

#Creating the Classification Tree
require(rpart.plot)
rpart.plot(Grades.cart,box.palette=list("Reds", "Blues","Oranges",
                                        "Greens","Grays"),)

##Prediction and logLoss values
require(stats)
predict.cart <- predict(Grades.cart,newdata=Grades.testing,
                        probability=TRUE)
require(ModelMetrics)
mlogLoss(actual=Grades.testing$Grade,predicted=predict.cart)
# [1] 1.52279
```

Appendix E

R Code for Random Forests

```
#Testing and Implementing the 'ranger' function in the 'rpart'
#package
set.seed(1234)
Grades.rf <-ranger(Grade~Semester+Size+StartTime+Day+Class+
                     YrSinceW2003,
                   data=Grades.testing,
                   mtry=2,
                   splitrule="gini",
                   importance="impurity",
                   min.node.size=1,
                   num.trees=1500,
                   probability=TRUE,
                   classification=TRUE)
print(Grades.rf)
Grades.rf$variable.importance

##Prediction and logLoss values
require(stats)
predict.rf <- predict(Grades.rf,data=Grades.testing,type="response")
predict.rf$predictions[1,]
require(ModelMetrics)
mlogLoss(actual=Grades.testing$Grade,predicted=predict.rf$predictions)
# [1] 1.460193
```

Appendix F

MATLAB Code for Trigonometric Polynomials

Matlab Code

```
function [px, py, perror] =
   computeTrigonometricInterpolationError(y,px,ex,exy)
m = length(y);
n = floor((m+1)/2);
z = fft(y)/m;
a0 = z(1) ;
an = 2*real(z(2:n));
anp1 = z(n+1);
bn = -2*imag(z(2:n));
k = 1:length(an);
py = a0 + an*cos(k'*px) + bn*sin(k'*px);
perror = a0 + an*cos(k'*ex) + bn*sin(k'*ex);
if (mod(m,2)== 0)
    py = py + anp1*cos(n*px);
    perror = perror + anp1*cos(n*ex);
end
perror = norm(perror-exy);
end

function [ NT_A, NT_B, NT_C, NT_D, NT_EW] =
   getAllNonTraditionalCumulativeStartTimeData(data,term,
   course,STimes)

NT_A = getNonTraditionalCumulativeStartTimeData(data,term,
   course,STimes,‘A’);
NT_B = getNonTraditionalCumulativeStartTimeData(data,term,
   course,STimes,‘B’);
NT_C = getNonTraditionalCumulativeStartTimeData(data,term,
   course,STimes,‘C’);
```

```
NT_D = getNonTraditionalCumulativeStartTimeData(data,term,
   course,STimes,'D');
NT_EW = getNonTraditionalCumulativeStartTimeData(data,term,
   course,STimes,'EW');
end

function H = getCumulativeStartTimeData(data,term, course,
   STimes,myGrade)
[~,k] = size(STimes);
H = zeros(1,k);

posSemester = strcmp(data.Semester(:),term);
posClass = strcmp(data.Class(:),course);
pos = eq(posSemester,1) & eq(posClass,1);
D = data(pos,:);
posGrade = strcmp(D.Grade,myGrade);
D1 = D(posGrade,:);

for i = 1:k

    posTimes = strcmp(D1.StartTime,STimes(1,i));
    p = find(posTimes==1);
    H(i) = sum(D1(p,:).Freq) / sum(D1(p,:).Size);
end

end

function H = getNonTraditionalCumulativeStartTimeData(data,
   term, course,STimes,myGrade)
[~,k] = size(STimes);
H = zeros(1,k);
posSemester = strcmp(data.Semester(:),term);
posClass = strcmp(data.Class(:),course);
pos = eq(posSemester,1) & eq(posClass,1);
D = data(pos,:);
posGrade = strcmp(D.Grade,myGrade);
D1 = D(posGrade,:);
for i = 1:k

    posTimes = strcmp(D1.StartTime,STimes(1,i));
    p = find(posTimes==1);
    H(i) = sum(D1(p,:).Freq) / sum(D1(p,:).Size);
end

end

function YGD = getYearGradeData(data,term, course,STimes,
   myGrade, year)
```

```
[~,k] = size(STimes);
cy = size(year);
YGD = zeros(1,cy(2));
posSemester = strcmp(data.Semester(:),term);
posClass = strcmp(data.Class(:),course);
pos = (eq(posSemester,1) & eq(posClass,1));
D = data(pos,:);
posGrade = strcmp(D.Grade,myGrade);
D = D(posGrade,:);
rD = size(D);
posTimes = zeros(rD(1),1);
for i = 1:k
    posTimes = posTimes + strcmp(D.StartTime,STimes(1,i));
end
D = D(eq(posTimes,1),:);
for i = 1:cy(2)
    posYear = eq(D.YrSince2003,year(1,i));
    D1 = D(eq(posYear,1),:);
    YGD(i) = sum(D1.Freq) / sum(D1.Size);
end
end

clear
clc
statisticsFallStartTimes = {‘9:30’, ‘10:30’, ‘11:30’,
   ‘12:30’,‘13:30’};
statisticsFallNonTraditionalStartTimes = {‘14:30’};
T = readtable(‘Grades.csv’);
data = T(:,{‘Semester’,‘Size’,‘StartTime’,‘Day’,‘Class’,‘
   YrSince2003’,‘Freq’,‘Grade’});
H_A = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘A’);
H_B = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘B’);
H_C = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘C’);
H_D = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘D’);
H_EW = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘EW’);
% H_A + H_B + H_C + H_D + H_EW

[ NT_A, NT_B, NT_C, NT_D, NT_EW] =
   getAllNonTraditionalCumulativeStartTimeData(data,‘Fall’, ‘
   MAT135’,statisticsFallNonTraditionalStartTimes);

m = length(H_A);
x = linspace(0,2*pi,m+1);
```

```
figure
plot(x(1:end-1),H_A,‘ro’,x(1:end-1),H_B,‘b+’,x(1:end-1),H_C,‘
   gd’,x(1:end-1),H_D,‘cs’,x(1:end-1),H_EW,‘mx’,‘LineWidth’,2)
ylabel(‘Proportion of Students’)
xlabel(‘Hours after 9:30 AM’)
title(‘MAT135 Fall Semester’)
legend(‘A’,‘B’,‘C’,‘D’,‘EW’)
xticks([0 2*pi/5 4*pi/5 6*pi/5 8*pi/5 2*pi 12*pi/5])
xticklabels({‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’})
xlim([0 2*pi+pi/6])
hold on
%
% plot Non Traditional data
% plot(x(end),NT_A, ‘ko’,  x(end),NT_B, ‘k+’, x(end),NT_C, ‘kd
   ’, x(end),NT_D, ‘ks’,‘LineWidth’,2)
plot(x(end),NT_A, ‘ko’,  x(end),NT_B, ‘k+’, x(end),NT_C, ‘kd’,
    x(end),NT_D, ‘ks’, x(end),NT_EW, ‘kx’,‘LineWidth’,2)
%
% compute the interpolating polynomial
%
NumInt = 200;
x_int = linspace(-pi/2,2*pi+pi/2,NumInt);
[pxA, pyA] = computeTrigonometricInterpolation(H_A,x_int);
[pxB, pyB] = computeTrigonometricInterpolation(H_B,x_int);
[pxC, pyC] = computeTrigonometricInterpolation(H_C,x_int);
[pxD, pyD] = computeTrigonometricInterpolation(H_D,x_int);
[pxEW, pyEW] = computeTrigonometricInterpolation(H_EW,x_int);
% plot(pxA,pyA, ‘r.’,pxB,pyB, ‘b.’,pxC,pyC, ‘g--’,pxD,pyD, ‘c
   -.’,‘LineWidth’,2)
plot(pxA,pyA, ‘r.’,pxB,pyB, ‘b:’,pxC,pyC, ‘g--’,pxD,pyD, ‘c
   -.’,pxEW,pyEW, ‘m-’,‘LineWidth’,2)

%[pyA(167),pyB(167),pyC(167),pyD(167),pyEW(167)]
%[ NT_A, NT_B, NT_C, NT_D, NT_EW]

clear
clc
calculusFallStartTimes = {‘8:30’,‘9:30’, ‘10:30’, ‘11:30’,
   ‘12:30’};
calculusFallNonTraditionalStartTimes = {‘8:00’};
T = readtable(’Grades.csv’);
data = T(:,{‘Semester’,‘Size’,‘StartTime’,‘Day’,‘Class’,‘
   YrSince2003’,‘Freq’,‘Grade’});
H_A = getCumulativeStartTimeData(data,‘Fall’, ‘MAT250’,
   calculusFallStartTimes,‘A’);
H_B = getCumulativeStartTimeData(data,‘Fall’, ‘MAT250’,
   calculusFallStartTimes,‘B’);
H_C = getCumulativeStartTimeData(data,‘Fall’, ‘MAT250’,
   calculusFallStartTimes,‘C’);
H_D = getCumulativeStartTimeData(data,‘Fall’, ‘MAT250’,
   calculusFallStartTimes,‘D’);
H_EW = getCumulativeStartTimeData(data,‘Fall’, ‘MAT250’,
   calculusFallStartTimes,‘EW’);
% H_A + H_B + H_C + H_D + H_EW

[ NT_A, NT_B, NT_C, NT_D, NT_EW] =
   getAllNonTraditionalCumulativeStartTimeData(data,‘Fall’, ‘
   MAT250’,calculusFallNonTraditionalStartTimes);

m = length(H_A);
x = linspace(0,2*pi,m+1);
figure
% plot(x(1:end-1),H_A,‘ro’,x(1:end-1),H_B,‘b+’,x(1:end-1),H_C
   ,‘gd’,x(1:end-1),H_D,‘cs’,‘LineWidth’,2)
plot(x(1:end-1),H_A,‘ro’,x(1:end-1),H_B,‘b+’,x(1:end-1),H_C,‘
   gd’,x(1:end-1),H_D,‘cs’,x(1:end-1),H_EW,‘mx’,‘LineWidth’,2)

ylabel(‘Proportion of Students’)
xlabel(‘Hours after 8:30 AM’)
title(‘MAT250 Fall Semester’)
legend(‘A’,‘B’,‘C’,‘D’,‘EW’)
xticks([-2*pi/5 0 2*pi/5 4*pi/5 6*pi/5 8*pi/5 2*pi 12*pi/5])
xticklabels({‘-1’,‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’})
xlim([-2*pi/5 2*pi])
hold on
%
% plot Non Traditional data
%
% plot(-0.5*x(2),NT_A, ‘ko’,  -0.5*x(2),NT_B, ‘k+’, -0.5*x(2),
   NT_C, ‘kd’, -0.5*x(2),NT_D, ‘ks’, ‘LineWidth’,2)
plot(-0.5*x(2),NT_A, ‘ko’,  -0.5*x(2),NT_B, ‘k+’, -0.5*x(2),NT
   _C, ‘kd’, -0.5*x(2),NT_D, ‘ks’,-0.5*x(2),NT_EW, ‘kx’, ‘
   LineWidth’,2)

%
% compute the interpolating polynomial
%
NumInt = 200;
x_int = linspace(-pi/2,2*pi+pi/2,NumInt);
[pxA, pyA] = computeTrigonometricInterpolation(H_A,x_int);
[pxB, pyB] = computeTrigonometricInterpolation(H_B,x_int);
[pxC, pyC] = computeTrigonometricInterpolation(H_C,x_int);
[pxD, pyD] = computeTrigonometricInterpolation(H_D,x_int);
[pxEW, pyEW] = computeTrigonometricInterpolation(H_EW,x_int);
% plot(pxA,pyA, ‘r.’,pxB,pyB, ‘b:’,pxC,pyC, ‘g--’,pxD,pyD, ‘c
   -.','LineWidth',2)
plot(pxA,pyA, 'r.',pxB,pyB, 'b:',pxC,pyC, 'g--',pxD,pyD, 'c
   -.',pxEW,pyEW, 'm-','LineWidth',2)

clear
clc
statisticsSpringStartTimes = {'8:30','9:30', '10:30', '11:30',
    '12:30','13:30'};
statisticsSpringNonTraditionalStartTimes = {'8:00'};
T = readtable('Grades.csv');
data = T(:,{'Semester','Size','StartTime','Day','Class','
   YrSince2003','Freq','Grade'});
H_A = getCumulativeStartTimeData(data,'Spring', 'MAT135',
   statisticsSpringStartTimes,'A');
H_B = getCumulativeStartTimeData(data,'Spring', 'MAT135',
   statisticsSpringStartTimes,'B');
H_C = getCumulativeStartTimeData(data,'Spring', 'MAT135',
   statisticsSpringStartTimes,'C');
H_D = getCumulativeStartTimeData(data,'Spring', 'MAT135',
   statisticsSpringStartTimes,'D');
H_EW = getCumulativeStartTimeData(data,'Spring', 'MAT135',
   statisticsSpringStartTimes,'EW');
% H_A + H_B + H_C + H_D + H_EW

[ NT_A, NT_B, NT_C, NT_D, NT_EW] =
   getAllNonTraditionalCumulativeStartTimeData(data,'Spring',
   'MAT135',statisticsSpringNonTraditionalStartTimes);

m = length(H_A);
x = linspace(0,2*pi,m+1);
figure
% plot(x(1:end-1),H_A,'ro',x(1:end-1),H_B,'b+',x(1:end-1),H_C
   ,'gd',x(1:end-1),H_D,'cs','LineWidth',2)
plot(x(1:end-1),H_A,'ro',x(1:end-1),H_B,'b+',x(1:end-1),H_C,'
   gd',x(1:end-1),H_D,'cs',x(1:end-1),H_EW,'mx','LineWidth',2)

ylabel('Proportion of Students')
xlabel('Hours after 8:30 AM')
title('MAT135 Spring Semester')
legend('A','B','C','D','EW')
xticks([-pi/3 0 pi/3 2*pi/3 pi 4*pi/3 5*pi/3 2*pi])
xticklabels({'-1','0', '1', '2', '3', '4', '5', '6'})
xlim([-x(2) 2*pi+pi/4])
hold on
%
% plot Non Traditional data
%
% plot(-0.5*x(2),NT_A, 'ko',  -0.5*x(2),NT_B, 'k+', -0.5*x(2),
   NT_C, ‘kd’, -0.5*x(2),NT_D, ‘ks’, ‘LineWidth’,2)
plot(-0.5*x(2),NT_A, ‘ko’,  -0.5*x(2),NT_B, ‘k+’, -0.5*x(2),NT
   _C, ‘kd’, -0.5*x(2),NT_D, ‘ks’,-0.5*x(2),NT_EW, ‘kx’, ‘
   LineWidth’,2)

%
% compute the interpolating polynomial
%
NumInt = 200;
x_int = linspace(-pi/2,2*pi+pi/2,NumInt);
[pxA, pyA] = computeTrigonometricInterpolation(H_A,x_int);
[pxB, pyB] = computeTrigonometricInterpolation(H_B,x_int);
[pxC, pyC] = computeTrigonometricInterpolation(H_C,x_int);
[pxD, pyD] = computeTrigonometricInterpolation(H_D,x_int);
[pxEW, pyEW] = computeTrigonometricInterpolation(H_EW,x_int);
% plot(pxA,pyA, ‘r.’,pxB,pyB, ‘b.’,pxC,pyC, ‘g--’,pxD,pyD, ‘c
   -.’,‘LineWidth’,2)
plot(pxA,pyA, ‘r.’,pxB,pyB, ‘b:’,pxC,pyC, ‘g--’,pxD,pyD, ‘c
   -.’,pxEW,pyEW, ‘m-’,‘LineWidth’,2)

clear
clc
calculusSpringStartTimes ={‘9:30’, ‘10:30’, ‘11:30’, ‘12:30’};

T = readtable(‘Grades.csv’);
data = T(:,{‘Semester’,‘Size’,‘StartTime’,‘Day’,‘Class’,‘
   YrSince2003’,‘Freq’,‘Grade’});
H_A = getCumulativeStartTimeData(data,‘Spring’, ‘MAT250’,
   calculusSpringStartTimes,‘A’);
H_B = getCumulativeStartTimeData(data,‘Spring’, ‘MAT250’,
   calculusSpringStartTimes,‘B’);
H_C = getCumulativeStartTimeData(data,‘Spring’, ‘MAT250’,
   calculusSpringStartTimes,‘C’);
H_D = getCumulativeStartTimeData(data,‘Spring’, ‘MAT250’,
   calculusSpringStartTimes,‘D’);
H_EW = getCumulativeStartTimeData(data,‘Spring’, ‘MAT250’,
   calculusSpringStartTimes,‘EW’);
%H_A + H_B + H_C + H_D + H_EW
m = length(H_A);
x = linspace(0,2*pi,m+1);
figure
% plot(x(1:end-1),H_A,‘ro’,x(1:end-1),H_B,‘b+’,x(1:end-1),H_C
   ,‘gd’,x(1:end-1),H_D,‘cs’,‘LineWidth’,2)
plot(x(1:end-1),H_A,‘ro’,x(1:end-1),H_B,‘b+’,x(1:end-1),H_C,‘
   gd’,x(1:end-1),H_D,‘cs’,x(1:end-1),H_EW,‘mx’,‘LineWidth’,2)

ylabel(‘Proportion of Students’)
xlabel(‘Hours after 9:30 AM’)
```

```
title('MAT250 Spring Semester')
legend('A','B','C','D','EW')
xticks([0 pi/2 pi 3*pi/2 2*pi ])
xticklabels({'0', '1', '2', '3', '4'})
xlim([0 2*pi])
hold on
NumInt = 200;
x_int = linspace(-pi/2,2*pi+pi/2,NumInt);
[pxA, pyA] = computeTrigonometricInterpolation(H_A,x_int);
[pxB, pyB] = computeTrigonometricInterpolation(H_B,x_int);
[pxC, pyC] = computeTrigonometricInterpolation(H_C,x_int);
[pxD, pyD] = computeTrigonometricInterpolation(H_D,x_int);
[pxEW, pyEW] = computeTrigonometricInterpolation(H_EW,x_int);
% plot(pxA,pyA, 'r.',pxB,pyB, 'b:',pxC,pyC, 'g--',pxD,pyD, 'c
   -.','LineWidth',2)
plot(pxA,pyA, 'r.',pxB,pyB, 'b:',pxC,pyC, 'g--',pxD,pyD, 'c
   -.',pxEW,pyEW, 'm-','LineWidth',2)

clear;
clc;
close all;
statisticsFallStartTimes = {'9:30', '10:30', '11:30',
   '12:30','13:30'};
statisticsFallNonTraditionalStartTimes = {'14:30'};
%
statisticsSpringStartTimes = {'8:30','9:30', '10:30', '11:30',
    '12:30','13:30'};
statisticsSpringNonTraditionalStartTimes = {'8:00'};
%
calculusFallStartTimes = {'8:30','9:30', '10:30', '11:30',
   '12:30'};
calculusFallNonTraditionalStartTimes = {'8:00'};
%
calculusSpringStartTimes ={'9:30', '10:30', '11:30', '12:30'};

Years = [0, 1, 2, 3,4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];

T = readtable('Grades.csv');
data = T(:,{'Semester','Size','StartTime','Day','Class','
   YrSince2003','Freq','Grade'});
%
% ************************************************************
   ****************
% MAT 135 FALL MODELING
% ************************************************************
   ****************
%
YGD_A = getYearGradeData(data,'Fall', 'MAT135',
   statisticsFallStartTimes,‘A’,Years);
YGD_B = getYearGradeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘B’,Years);
YGD_C = getYearGradeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘C’,Years);
YGD_D = getYearGradeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘D’,Years);
YGD_EW = getYearGradeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘EW’,Years);
%
CSTD_A = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘A’);
CSTD_B = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘B’);
CSTD_C = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘C’);
CSTD_D = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘D’);
CSTD_EW = getCumulativeStartTimeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘EW’);

% [ NT_A, NT_B, NT_C, NT_D, NT_EW] =
   getAllNonTraditionalCumulativeStartTimeData(data,‘Fall’, ‘
   MAT135’,statisticsFallNonTraditionalStartTimes);
%
%
% looking at Grades A, B, & C plots suggest that we have
   period from 4 to 10 years after 2003
%
m = length(YGD_A);
xx = linspace(0,2*pi,m+1);
x = -4*pi/3:pi/3:12*pi/3; %linspace(-4*pi/3,11*pi/3,m+1);
figure
plot(x(1:end-1),YGD_A,‘ro’,x(1:end-1),YGD_B,‘b+’,x(1:end-1),
   YGD_C,‘gd’,x(1:end-1),YGD_D,‘cs’,‘LineWidth’,2)
ylabel(‘Proportion of Students’)
xlabel(‘T years after 2003’)
title(‘MAT135 Fall Semester - cycle assumed from T = 4 to 10’)
legend(‘A’,‘B’,‘C’,‘D’,‘EW’)
 xticks([-4*pi/3 -2*pi/3 0 2*pi/3 4*pi/3 6*pi/3  8*pi/3 10*pi/
    3 12*pi/3])
xticklabels({‘0’, ‘2’, ‘4’, ‘6’, ‘8’, ‘10’, ‘12’, ‘14’})

% xlim([0 2*pi+pi/6])
 hold on
% %
% % plot Non Traditional data
% %
% plot(x(end),NT_A, ‘ko’,  x(end),NT_B, ‘k+’, x(end),NT_C, ‘kd
   ’, x(end),NT_D, ‘ks’, ‘LineWidth’,2)
% %
% % compute the interpolating polynomial
% %
NumInt = 200;
xY_int = linspace(-4*pi/3,11*pi/3,NumInt);
[pxYA, pyYA] = computeTrigonometricInterpolation(YGD_A(5:10),
   xY_int);
[pxYB, pyYB] = computeTrigonometricInterpolation(YGD_B(5:10),
   xY_int);
[pxYC, pyYC] = computeTrigonometricInterpolation(YGD_C(5:10),
   xY_int);
[pxYD, pyYD] = computeTrigonometricInterpolation(YGD_D(5:10),
   xY_int);

xST_int = linspace(-pi/2,2*pi+pi/2,NumInt);
[pxCSTDA, pyCSTDA] = computeTrigonometricInterpolation(CSTD_A,
   xST_int);
[pxCSTDB, pyCSTDB] = computeTrigonometricInterpolation(CSTD_B,
   xST_int);
[pxCSTDC, pyCSTDC] = computeTrigonometricInterpolation(CSTD_C,
   xST_int);
[pxCSTDD, pyCSTDD] = computeTrigonometricInterpolation(CSTD_D,
   xST_int);
[pxCSTDEW, pyCSTDEW] = computeTrigonometricInterpolation(CSTD_
   EW,xST_int);
plot(pxYA,pyYA, ‘r.’,pxYB,pyYB, ‘b:’,pxYC,pyYC, ‘g--’,pxYD,
   pyYD, ‘c-.’,’LineWidth’,2)
%
figure
[X, Y] = meshgrid(xY_int,xST_int);
 zA = flip(pyYA)’.* flip(pyCSTDA);
 surf(X, Y, zA)
title(‘MAT135 FALL - Grade A’)
zlabel(‘Proportion of students’)
xlabel(‘Years since 2003’)
ylabel(‘Hours after 9:30 AM’)
xticks([-4*pi/3 -2*pi/3 0 2*pi/3 4*pi/3 6*pi/3  8*pi/3 10*pi/3
    12*pi/3])
xticklabels({‘0’, ‘2’, ‘4’, ‘6’, ‘8’, ‘10’, ‘12’, ‘14’})
yticks([0 2*pi/5 4*pi/5 6*pi/5 8*pi/5 2*pi 12*pi/5])
yticklabels({‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’})

clear;
clc;
close all;
```

```
statisticsFallStartTimes = {‘9:30’, ‘10:30’, ‘11:30’,
   ‘12:30’,‘13:30’};
statisticsFallNonTraditionalStartTimes = {‘14:30’};
%
statisticsSpringStartTimes = {‘8:30’,‘9:30’, ‘10:30’, ‘11:30’,
    ‘12:30’,‘13:30’};
statisticsSpringNonTraditionalStartTimes = {‘8:00’};
%
calculusFallStartTimes = {‘8:30’,‘9:30’, ‘10:30’, ‘11:30’,
   ‘12:30’};
calculusFallNonTraditionalStartTimes = {‘8:00’};
%
calculusSpringStartTimes ={‘9:30’, ‘10:30’, ‘11:30’, ‘12:30’};

Years = [0, 1, 2, 3,4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
%
% Error Matrix:  [MAT135F MAT135S MAT250F MAT250S]’ X [ A B C
   D EW]
%
ErrorMatrix = zeros(4,5);
%
% Examination of data give an appearance of data cycling every
    six years with a cycle [4,5,6,7,8,9,10]
%
T = readtable(‘Grades.csv’);
data = T(:,{‘Semester’,’Size’,’StartTime’,’Day’,’Class’,’
   YrSince2003’,’Freq’,’Grade’});
%
% ************************************************************
   ****************
% MAT 135 FALL MODELING
% ************************************************************
   ****************
%
YGD_A = getYearGradeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘A’,Years);
YGD_B = getYearGradeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘B’,Years);
YGD_C = getYearGradeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘C’,Years);
YGD_D = getYearGradeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘D’,Years);
YGD_EW = getYearGradeData(data,‘Fall’, ‘MAT135’,
   statisticsFallStartTimes,‘EW’,Years);

 m = length(YGD_A);
%x = -4*pi/3:pi/3:12*pi/3; %
```

```
x = linspace(-4*pi/3,12*pi/3,m+1);
figure
% plot(x(1:end-1),YGD_A,‘ro’,x(1:end-1),YGD_B,‘b+’,x(1:end-1),
   YGD_C,‘gd’,x(1:end-1),YGD_D,‘cs’,‘LineWidth’,2)
plot(x(1:end-1),YGD_A,‘ro’,x(1:end-1),YGD_B,‘b+’,x(1:end-1),
   YGD_C,‘gd’,x(1:end-1),YGD_D,‘cs’,x(1:end-1),YGD_EW,‘m+’,‘
   LineWidth’,2)

ylabel(‘Proportion of Students’)
xlabel(‘T years after 2003’)
title(‘MAT135 Fall Semester - cycle assumed from T = 4 to 10’)
legend(‘A’,’B’,’C’,’D’,’EW’)
xticks([-4*pi/3 -2*pi/3 0 2*pi/3 4*pi/3 6*pi/3  8*pi/3 10*pi/3
    12*pi/3])
xticklabels({‘0’, ‘2’, ‘4’, ‘6’, ‘8’, ‘10’, ‘12’, ‘14’})
hold on
% %
% % compute the interpolating polynomial
% %
NumInt = 200;
x_int = linspace(-4*pi/3,11*pi/3,NumInt);

[pxA, pyA, pyAerror] = computeTrigonometricInterpolationError(
   YGD_A(5:10),x_int,x(1:end-1),YGD_A);
[pxB, pyB, pyBerror] = computeTrigonometricInterpolationError(
   YGD_B(5:10),x_int,x(1:end-1),YGD_B);
[pxC, pyC, pyCerror] = computeTrigonometricInterpolationError(
   YGD_C(5:10),x_int,x(1:end-1),YGD_C);
[pxD, pyD, pyDerror] = computeTrigonometricInterpolationError(
   YGD_D(5:10),x_int,x(1:end-1),YGD_D);
[pxEW, pyEW, pyEWerror] =
   computeTrigonometricInterpolationError(YGD_EW(5:10),x_int,x
   (1:end-1),YGD_EW);

ErrorMatrix(1,:)= [pyAerror, pyBerror, pyCerror, pyDerror,
   pyEWerror];

% plot(pxA,pyA, ‘r.’,pxB,pyB, ‘b:’,pxC,pyC, ‘g--’,pxD,pyD, ‘c
   -.’,‘LineWidth’,2)
plot(pxA,pyA, ‘r.’,pxB,pyB, ‘b:’,pxC,pyC, ‘g--’,pxD,pyD, ‘c
   -.’,pxEW,pyEW, ‘m-.’,‘LineWidth’,2)
% %
% % ***********************************************************
   ******************
% % MAT 135 SPRING MODELING
% % ***********************************************************
   ******************
```

```
% %
YGD_A = getYearGradeData(data,'Spring', 'MAT135',
   statisticsFallStartTimes,'A',Years);
YGD_B = getYearGradeData(data,'Spring', 'MAT135',
   statisticsFallStartTimes,'B',Years);
YGD_C = getYearGradeData(data,'Spring', 'MAT135',
   statisticsFallStartTimes,'C',Years);
YGD_D = getYearGradeData(data,'Spring', 'MAT135',
   statisticsFallStartTimes,'D',Years);
YGD_EW = getYearGradeData(data,'Spring', 'MAT135',
   statisticsFallStartTimes,'EW',Years);

figure

% plot(x(1:end-1),YGD_A,'ro',x(1:end-1),YGD_B,'b+',x(1:end-1),
   YGD_C,'gd',x(1:end-1),YGD_D,'cs','LineWidth',2)
plot(x(1:end-1),YGD_A,'ro',x(1:end-1),YGD_B,'b+',x(1:end-1),
   YGD_C,'gd',x(1:end-1),YGD_D,'cs',x(1:end-1),YGD_EW,'m+','
   LineWidth',2)

ylabel('Proportion of Students')
xlabel('T years after 2003')
title('MAT135 Spring Semester - cycle assumed from T = 4 to
   10')
legend('A','B','C','D','EW')
 xticks([-4*pi/3 -2*pi/3 0 2*pi/3 4*pi/3 6*pi/3  8*pi/3 10*pi/
    3 12*pi/3])
xticklabels({'0', '2', '4', '6', '8', '10', '12', '14'})
hold on
%
% compute the interpolating polynomial
%

[pxA, pyA, pyAerror] = computeTrigonometricInterpolationError(
   YGD_A(5:10),x_int,x(1:end-1),YGD_A);
[pxB, pyB, pyBerror] = computeTrigonometricInterpolationError(
   YGD_B(5:10),x_int,x(1:end-1),YGD_B);
[pxC, pyC, pyCerror] = computeTrigonometricInterpolationError(
   YGD_C(5:10),x_int,x(1:end-1),YGD_C);
[pxD, pyD, pyDerror] = computeTrigonometricInterpolationError(
   YGD_D(5:10),x_int,x(1:end-1),YGD_D);
[pxEW, pyEW, pyEWerror] =
   computeTrigonometricInterpolationError(YGD_EW(5:10),x_int,x
   (1:end-1),YGD_EW);

ErrorMatrix(2,:)= [pyAerror, pyBerror, pyCerror, pyDerror,
   pyEWerror];
```

```
% plot(pxA,pyA, 'r.',pxB,pyB, 'b:',pxC,pyC, 'g--',pxD,pyD, 'c
   -.','LineWidth',2)
plot(pxA,pyA, 'r.',pxB,pyB, 'b:',pxC,pyC, 'g--',pxD,pyD, 'c
   -.',pxEW,pyEW, 'm-.','LineWidth',2)

%
% ***********************************************************
   ***************
% MAT 250 SPRING MODELING
% ***********************************************************
   ***************
%
YGD_A = getYearGradeData(data,'Spring', 'MAT250',
   statisticsFallStartTimes,'A',Years);
YGD_B = getYearGradeData(data,'Spring', 'MAT250',
   statisticsFallStartTimes,'B',Years);
YGD_C = getYearGradeData(data,'Spring', 'MAT250',
   statisticsFallStartTimes,'C',Years);
YGD_D = getYearGradeData(data,'Spring', 'MAT250',
   statisticsFallStartTimes,'D',Years);
YGD_EW = getYearGradeData(data,'Spring', 'MAT250',
   statisticsFallStartTimes,'EW',Years);

figure

% plot(x(1:end-1),YGD_A,'ro',x(1:end-1),YGD_B,'b+',x(1:end-1),
   YGD_C,'gd',x(1:end-1),YGD_D,'cs','LineWidth',2)
plot(x(1:end-1),YGD_A,'ro',x(1:end-1),YGD_B,'b+',x(1:end-1),
   YGD_C,'gd',x(1:end-1),YGD_D,'cs',x(1:end-1),YGD_EW,'m+','
   LineWidth',2)

ylabel('Proportion of Students')
xlabel('T years after 2003')
title('MAT250 Spring Semester - cycle assumed from T = 4 to
   10')
legend('A','B','C','D','EW')
 xticks([-4*pi/3 -2*pi/3 0 2*pi/3 4*pi/3 6*pi/3  8*pi/3 10*pi/
    3 12*pi/3])
xticklabels({'0', '2', '4', '6', '8', '10', '12', '14'})
hold on

% compute the interpolating polynomial
%

[pxA, pyA, pyAerror] = computeTrigonometricInterpolationError(
   YGD_A(5:10),x_int,x(1:end-1),YGD_A);
[pxB, pyB, pyBerror] = computeTrigonometricInterpolationError(
   YGD_B(5:10),x_int,x(1:end-1),YGD_B);
[pxC, pyC, pyCerror] = computeTrigonometricInterpolationError(
   YGD_C(5:10),x_int,x(1:end-1),YGD_C);
[pxD, pyD, pyDerror] = computeTrigonometricInterpolationError(
   YGD_D(5:10),x_int,x(1:end-1),YGD_D);
[pxEW, pyEW, pyEWerror] =
   computeTrigonometricInterpolationError(YGD_EW(5:10),x_int,x
   (1:end-1),YGD_EW);

ErrorMatrix(4,:)= [pyAerror, pyBerror, pyCerror, pyDerror,
   pyEWerror];

% plot(pxA,pyA, ‘r.’,pxB,pyB, ‘b:’,pxC,pyC, ‘g--’,pxD,pyD, ‘c
   -.’,‘LineWidth’,2)
plot(pxA,pyA, ‘r.’,pxB,pyB, ‘b:’,pxC,pyC, ‘g--’,pxD,pyD, ‘c
   -.’,pxEW,pyEW, ‘m-.’,‘LineWidth’,2)
%
% ***********************************************************
   ****************
% MAT 250 FALL MODELING
% ***********************************************************
   ****************
%
YGD_A = getYearGradeData(data,‘Fall’, ‘MAT250’,
   statisticsFallStartTimes,‘A’,Years);
YGD_B = getYearGradeData(data,‘Fall’, ‘MAT250’,
   statisticsFallStartTimes,‘B’,Years);
YGD_C = getYearGradeData(data,‘Fall’, ‘MAT250’,
   statisticsFallStartTimes,‘C’,Years);
YGD_D = getYearGradeData(data,‘Fall’, ‘MAT250’,
   statisticsFallStartTimes,‘D’,Years);
YGD_EW = getYearGradeData(data,‘Fall’, ‘MAT250’,
   statisticsFallStartTimes,‘EW’,Years);

figure

% plot(x(1:end-1),YGD_A,‘ro’,x(1:end-1),YGD_B,‘b+’,x(1:end-1),
   YGD_C,‘gd’,x(1:end-1),YGD_D,‘cs’,‘LineWidth’,2)
plot(x(1:end-1),YGD_A,‘ro’,x(1:end-1),YGD_B,‘b+’,x(1:end-1),
   YGD_C,‘gd’,x(1:end-1),YGD_D,‘cs’,x(1:end-1),YGD_EW,‘m+’,‘
   LineWidth’,2)

ylabel(‘Proportion of Students’)
xlabel(‘T years after 2003’)
title(‘MAT250 Fall Semester - cycle assumed from T = 4 to 10’)
```

```
legend('A','B','C','D','EW')
 xticks([-4*pi/3 -2*pi/3 0 2*pi/3 4*pi/3 6*pi/3  8*pi/3 10*pi/
    3 12*pi/3])
xticklabels({'0', '2', '4', '6', '8', '10', '12', '14'})

hold on
%
% compute the interpolating polynomial
%

[pxA, pyA, pyAerror] = computeTrigonometricInterpolationError(
   YGD_A(5:10),x_int,x(1:end-1),YGD_A);
[pxB, pyB, pyBerror] = computeTrigonometricInterpolationError(
   YGD_B(5:10),x_int,x(1:end-1),YGD_B);
[pxC, pyC, pyCerror] = computeTrigonometricInterpolationError(
   YGD_C(5:10),x_int,x(1:end-1),YGD_C);
[pxD, pyD, pyDerror] = computeTrigonometricInterpolationError(
   YGD_D(5:10),x_int,x(1:end-1),YGD_D);
[pxEW, pyEW, pyEWerror] =
   computeTrigonometricInterpolationError(YGD_EW(5:10),x_int,x
   (1:end-1),YGD_EW);

ErrorMatrix(3,:)= [pyAerror, pyBerror, pyCerror, pyDerror,
   pyEWerror];

% plot(pxA,pyA, 'r.',pxB,pyB, 'b:',pxC,pyC, 'g--',pxD,pyD, 'c
   -.','LineWidth',2)
plot(pxA,pyA, 'r.',pxB,pyB, 'b:',pxC,pyC, 'g--',pxD,pyD, 'c
   -.',pxEW,pyEW, 'm-.','LineWidth',2)
```

References

[1] Breiman, L. (2001). Statistical modeling: the two cultures. *Statistical Science*, 16(3), 199-231. Retrieved from https://projecteuclid.org/download/pdf_1/euclid.ss/1009213726

[2] Fox, J., & Weisberg, S. (2011). *An R companion to applied regression: Second edition*. Thousand Oaks, CA: SAGE.

[3] Mishra, A. (2018, February 24). Metrics to evaluate your machine learning algorithm. Retrieved from https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

[4] Loss Functions (n.d.). Retrieved from https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

[5] Collier, A. B. (2015, December 14). Making sense of logarithmic loss. Retrieved from https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/

[6] Zammito, F. (2019). What's considered a good log loss in maching learning? Retrieved from https://medium.com/@fzammito/whats-considered-a-good-log-loss-in-machine-learning-a529d400632d

[7] Irizarry, R. A. (2020, February 24). *Introduction to data science: Data analysis and prediction algorithms with R*. CRC Press

[8] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An introduction to statistical learning: With application in R*. Springer

[9] Efron, B. & Hastie, T (2016). Computer age statistical inference. *Cambridge University Press*. Retrieved from https://we.stanford.edu/~hastie/CAST/

[10] Murphy, K. P. (2012). *Machine learning a probabilistic perspective*. Cambridge, MA: MIT Press.

[11] Kuhn, M., Contributions from Wing J., Weston S., Williams A., Keefer C., Engelhardt A., Cooper T., Mayer Z., Kenkel B., Benesty M., Lescarbeau R., Ziem A., Scrucca L., Tang Y., Candan C., Hunt T., & the R Core Team (2019). caret:

Classification and Regression Training. R package version 6.0-84. Retrieved from `https://CRAN.R-project.org/package=caret`

[12] Therneau T., & Atkinson B. (2019). rpart: Recursive Partitioning and Regression Trees. R package version 4.1-15. Retrieved from `https://CRAN.R-project.org/package=rpart`

[13] Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction: Second Edition*. Springer.

[14] Fox, J. (2008). *Applied regression analysis and generalized linear models: Second edition*. Thousand Oaks, CA: SAGE

[15] Yee, T., W. (2010). The VGAM Package for Categorical Data Analysis. Journal of Statistical Software, 32(10), 1-34. Retrieved from `http://www.jstatsoft.org/v32/i10/`.

[16] Hornung, R. (2019). ordinalForest: Ordinal Forests: Prediction and Variable Ranking with Ordinal Target Variables. R package version 2.3-1. Retrieved from `https://CRAN.R-project.org/package=ordinalForest`

[17] *Ordinal logistic regression* (n.d). UCLA. Retrieved March 8, 2020, from `https://stats.idre.ucla.edu/r/dae/ordinal-logistic-regression/`

[18] *How do I interprete the coefficients in an ordinal logistic regression in R?* (n.d). UCLA. Retrieved March 8, 2020, from `https://stats.idre.ucla.edu/r/faq/ologit-coefficients/`

[19] Venables, W. N., & Ripley, B. D. (2002). MASS: Modern Applied Statistics with S: Fourth Edition. New York, Spring. ISBN 0-387-95457-0 Retrieved from `http://www.stats.ox.ac.uk/pub/MASS4`

[20] Hunt, T. (2020). ModelMetrics: Rapid Calculation of Model Metrics. R package version 1.2.2.1. Retrieved from `https://CRAN.R-project.org/package=ModelMetrics`

[21] Yee, T. W. (2010). The VGAM Package for Categorical Data Analysis. *Journal of Statistical Software*, 32(10), 1-34. Retrieved from `https://www.jstatsoft.org/article/view/v032i10`

[22] Yee, T. W., & Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling: An international journal*, 3(1), 15-41.

[23] Wright, M. N., & Ziegler, A. (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software*, 77(1), 1-17. `doi:10.18637/jss.v077.i01`

[24] Faires, J. D., & Burden, R. (2003). Numerical methods: Third edition. *Brooks/-Cole.*