Applications of Machine Learning in Imputation

Vinayak Anand-Kumar

2019

Contents

Chapter 1

Introduction

Editing and imputation are both methods of data processing. Editing refers to the detection and correction of errors in the data, whilst imputation is a method of correcting errors in a dataset. This document presents findings from work carried out on the use of machine learning in imputation. The chapters address the following questions:

1) What is imputation?

2) What is machine learning?

3) Why use machine learning?

4) How XGBoost works?

5) Methods used for the investigation

6) Results of the investigation

7) Conclusions and future direction

Chapter 2

What is Imputation?

Editing and imputing are both methods of data processing. Editing refers to the detection and correction of errors in the data. Whilst imputation is a method of correcting errors and estimating and filling in missing values in a data set. Where there are errors in the data set, and when these are considered to add no value in the correction process, these values are set to missing and are imputed with a plausible estimate. Alternatively, missing values may already exist in the data, and imputation may be carried out to produce a complete data set for analysis.

This research project evaluated the use of machine learning methods for imputation. In order to provide a context for using machine learning in the imputation process, the reader is presented with:

- A rationale for carrying out imputation
- An introduction to the methods of imputation

2.1 Why is imputation carried out?

Missingness and erroneous values can impact the quality of data. A large volume of incorrect and/or missing values increase the risk of the product failing to capture the target concept or target population. That is, omissions (introduced in collection or processing) may result in certain sub-groups of the target population from being excluded in the analysis data set, and in turn increasing the risk of biased estimates, reducing the power of inferential statistics and increasing the uncertainty of estimates and inferences derived from the data. Similarly, errors in a data set may impact the degree to which the final estimate or output represents the reality it was designed to capture.

Correcting erroneous responses and filling in missing values helps manage the quality of data. A complete data set can improve the accuracy and reliability of estimates, and help maintain the consistency of counts across published tables. Moreover, a data set with fewer errors and more units may more accurately capture the underlying distribution of the variable of interest. Selecting a method for estimating values in a data set is generally advised by the nature of errors or missingness in the data, and the output desired from the analysis data set.

2.2 Methods

An imputation process of a data set can be broken down into the following three phases:

1) Review, whereby data is examined for potential problems; identifying instances of missingness and erroneous values

2) Select, whereby data is identified for further treatment. Of the potential problems identified in the review phase, a method is applied to determine which of these erroneous or missing cases need to be treated
3) Amend, whereby changes are applied to the data identified in the select phase by either correcting errors/ filling in missing values

The focus of this project was in applying Machine Learning methods to amend values in a data set. That is, it was of interest to compare existing approaches, of treating missing or erroneous values by estimating replacement figures, to machine learning methods. Methods of variable amendment can be grouped into one of the following categories:

- interactive treatment
- deductive imputation
- model based imputation
- donor based imputation

The mechanisms for a given imputation method could be deterministic or stochastic. The former refers to instances where repeated trials of the same method yield identical output. Whereas the latter refers to instances where there is element of randomness; repeated iterations will produce different results.

2.2.1 Interactive treatment

Interactive treatment refers to a class of methods whereby the data are adjusted by a human editor by either re-contacting the respondent/ data provider, replacing values from another variable/ data source, or creating a value based on subject matter expertise.

2.2.2 Deductive imputation

Deductive imputation uses logic or an understanding about the relationship between variables and units to fill in missing values. Examples include deriving a value as a function of other values, adopting a value from a related unit, and adopting a value from an earlier time point. Generally, this method is used when the true value can be derived with certainty or with a very high probability.

2.2.3 Model based imputation

Model based imputation refers to a class of methods that estimate missing values using assumptions about the distribution of the data, which include mean and median imputation. Or assumptions about the relationship between auxiliary variables (or x variables) and the target y variable to predict missing values.

2.2.4 Donor based imputation

Donor based imputation adopt values from an observed unit, which are then used for the missing unit. For each recipient with a missing value for variable y, a donor is identified that is similar to the recipient with respect to certain background characteristics (often referred to as matching variables) that are related to the target variable y. Such methods are relatively easy to apply when there are several related missing values in one record, and if the intention is to preserve the relationship between variables.

Chapter 3

What is Machine Learning?

Machine learning is the field of study that enables a program to learn from its experience of iterating through a task multiple times. A performance measure is generally specified by the programmer, which is used to evaluate how well the machine has carried out the task at each iteration. Learning of the task is evidenced by its improvement against the performance measure.
The different types of machine learning systems can be categorised with respect to:

- Whether or not they are trained with human supervision
- Whether or not they can learn incrementally or on the fly
- Whether they work by comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model

3.1 Supervision

Machine learning systems can vary with regards to the degree of supervision. The major types of supervision:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

3.1.1 Supervised learning

Supervised learning is the specification of the desired output. That is, the data used to train the model includes the solutions (which are referred to as labels), which the machine learning system attempts to estimate. The desired solutions specified in the machine learning algorithm are referred to as labels.

3.1.2 Unsupervised learning

Unsupervised learning uses training data that is unlabelled. In this class of machine learning systems, the outcome/ desired solutions are not specified in the machine learning algorithm.

3.1.3 Semi-supervised learning

Machine learning systems that use partially labelled data are categorised as utilising semi-supervised learning.

3.1.4 Reinforcement learning

Reinforcement learning involves the use of rewards or penalties to train the machine in identifying the appropriate action for a given situation. The learning system, which is referred to as an agent, observes the environment, selects and performs actions, and gets a response in the form of a reward or penalty. After multiple iterations, it identifies the best strategy, referred to as a policy, that results in the most reward over time.

3.2 Batch and Online learning

Another criterion for classifying machine learning systems is the way in which the algorithm learns. That is, whether the learning takes place at once or if it happens in increments.

3.2.1 Batch learning

Batch learning uses all the available data to train the machine learning system. This is generally time consuming and computationally expensive, and as a result is carried out offline. Whilst in production, the system is no longer learning, and is simply applying what it has learnt from the full set of training data.

Any changes to the data generating mechanism (GDM) will mean that a new system would need to be trained, from scratch on the full set of data (that includes data points before and after changes to the GDM).

3.2.2 Online learning

Online learning trains the system incrementally through sequential input of data. Data can be delivered individually or in small groups, referred to as mini-batches. As each learning step is relatively fast and cheap, the system can learn about new data whilst in production, as it arrives. It is an ideal approach for when the velocity of new data is high, and when there is a need to adapt to changes rapidly or autonomously.

3.3 Approaches to generalisation

Machine learning systems can also be categorised with regards to how the systems generalise. That is, there are different approaches to using the training data to develop a system that can then be generalised to new cases. The two main approaches are instance-based learning and model-based learning.

3.3.1 Instance-based learning

Instance-based learning identifies all instances of a given feature in the training data and uses a similarity measure to generalise to new cases.

3.3.2 Model-based learning

Model-based learning uses features in the training data to predict the outcome/ variable of interest; the model used to specify the relationship between the predictor(s) and outcome(s) are then generalised on new cases.

Chapter 4

Why use Machine Learning?

It was of interest to explore the utility of Machine Learning to directly impute for missing values in data sets. More specifically, the analysts were interested in examining whether Machine Learning models can improve the timeliness, reliability and accuracy of the imputation process in social survey data. Figure 1 presents the imputation pipeline for social survey data. Prior to imputation, units and values are reviewed, and those that are missing and should be routed to the item in question, are selected (i.e. flagged) for imputation. Data is then further processed before imputed and observed data are compiled in an analysis data set, used for publishing estimates and carrying out analyses.

The intention was to use a machine learning system to impute flagged missing values. This model based approach for imputation may reduce the data processing time and improve the precision and reduce the variance of estimates. The current approach, which utilises nearest neighbour donor imputation involves the following:

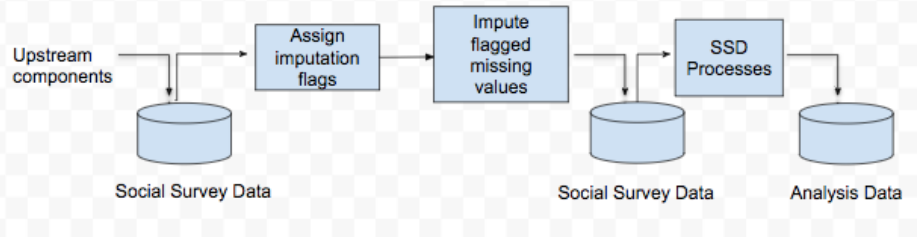

Figure 4.1: Figure 1. Imputation pipeline in social survey data.

- Setting up specification files for each variable and imputation group combination
- Iterating through weights for matching variables so that all missing values are imputed

Designing the selection criteria for donors can be time consuming as it requires analysts to identify matching variables (MV), along with weights for each MV. Teams currently use subject matter expertise in designing the donor imputation strategy for each variable. As this process is not data driven, it introduces an element of subjectivity and does not guarantee that matching variables selected are the best predictors of the variable of interest. In contrast, a data driven approach would be reproducible and identify the best predictors, in the data set, to estimate missing values. Moreover, applying the machine learning system may offer a more parsimonious approach as fewer input parameters and files would be required in executing imputation.

Analysts were interested in whether a Machine Learning System would perform better compared to the current imputation process with regards to:

- Timeliness: Would the ML system reduce processing time and by how much?
- Accuracy & Reliability: How do the two methods compare with respect to the bias and variance of estimates?
- Interpretability: What advantages and challenges do the ML system present with regards to making the imputation methods transparent?

At present, the following Machine Learning library was used in the investigation:

- XGBoost

Chapter 5

XGBoost

XGBoost is a set of open source functions and steps, referred to as a library, that use supervised ML where analysts specify an outcome to be estimated/ predicted. The XGBoost library uses multiple decision trees to predict an outcome.

The ML system is trained using batch learning and generalised through a model based approach. It uses all available data to construct a model that specifies the relationship between the predictor and outcome variables, which are then generalised to the test data set.

XGBoost stands for eXtreme Gradient Boosting. The word "extreme" reflects its goal to push the limit of computational resources. Whereas gradient boosting is a machine learning technique for regression and classification problems that optimises a collection of weak prediction models in an attempt to build an accurate and reliable predictor.

In order to build a better understanding of how XGBoost works, the documentation will briefly review:

- Decision trees: How decision trees play a role in XGBoost?
- Boosting: What is it?

The final section of this chapter provides a step by step guide on building models using XGBoost; the reader is encouraged to use this code to predict an outcome variable using available auxiliary variables.

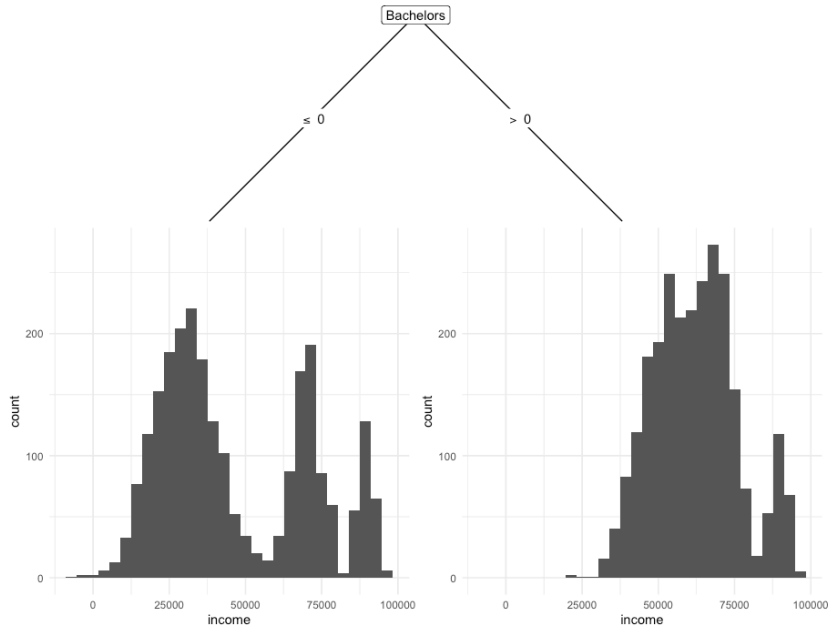

Figure 5.1: Figure 2. Decision tree that splits units in a dataset based on whether individual has a Bachelor's degree or not, in order to predict Income. The tree shows that those with a Bachelors degree (> 0) on average earn more than than those wihtout a Bachelor's degree (< 0).

5.1 Decision trees

Decision trees can be used as a method for grouping units in a data set by asking questions, such as "Does an individual have a Bachelor's degree?". In this example, two groups would be created; one for those with a Bachelor's degree, and one for those without. Figure 2 provides a visual depiction of this grouping in an attempt to explain Income.

Each subsequent question in a decision tree will produce a smaller group of units. This grouping is carried out to identify units with similar characteristics with respect to an outcome variable. The model in Figure 3 attempts to use University qualifications to predict Income.

The following characteristics are true of decision trees:

- A single question is asked at each decision node, and there are only two possible choices. With the example in Figure 3, the questions include 1) Does the individual have a PhD, 2) Does the individual have a Master's

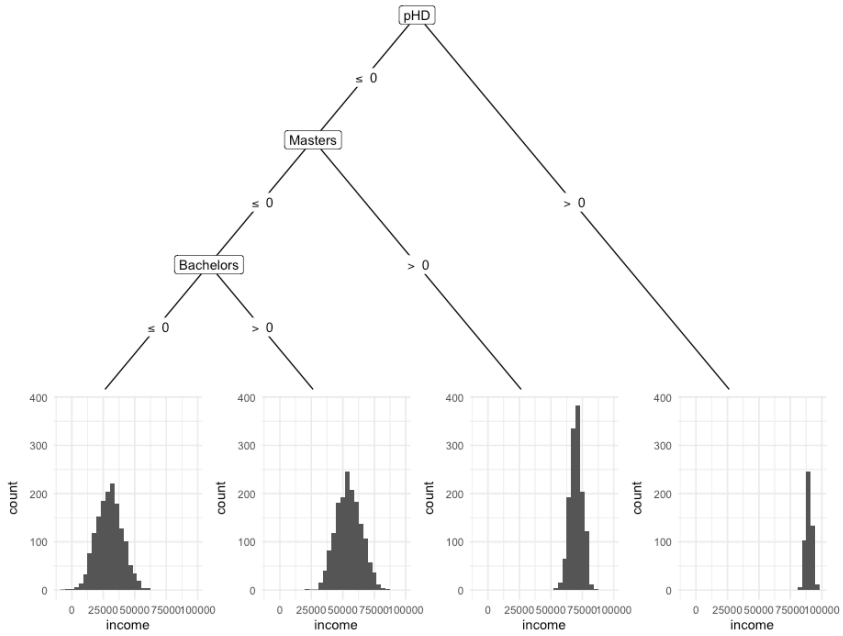

Figure 5.2: Figure 3. Decision tree that splits units in a dataset based on whether individual has a Bachelor's degree (yes/no), a Master's degree and PhD (yes/no), in order to predict Income. The tree shows that those with a higher qualification tend to earn more.

and 3) Does the individual have a Bachelor's degree.

- At the bottom of every decision tree, there is a single possible decision. Every possible decision will eventually lead to a choice. Some decisions will lead to a choice sooner. The model in Figure 3 attempts to predict Income using University Qualifications. Each node presents a question about whether an individual possesses a given qualification. The end nodes present the distribution of income for individuals with the specified qualifications. As a result, the choices would be the expected value of Income for an individual, given the qualifications obtained.

Decision trees are a learning method that involve a tree like graph to model either continuous or categorical choice given some data. It is composed of a series of binary questions, which when answered in succession yield a prediction about data at hand. XGBoost uses Classification and Regression Trees (CART), which are presented in the above examples, to predict the outcome variable.

5.1.1 Boosting

A single decision tree is considered a weak/ base learner as it is only slightly better than chance at predicting the outcome variable. Whereas strong learners are any algorithm that can be tuned to achieve peak performance for supervised learning. XGBoost uses decision trees as base learners; combining many weak learners to form a strong learner. As a result it is referred to as an ensemble learning method; using the output of many models (i.e. trees) in the final prediction.

The concept of combining many weak learners to produce a strong learner is referred as boosting. XGBoost will iteratively build a set of weak models on subsets of the data; weighting each weak prediction according to the weak learner's performance. A prediction is derived by taking the weighted sum of all base learners.

5.1.2 Building models with XGBoost

In the training data, a target variable y_i is specified, whilst all other features x_i are used as predictors of the target variable. A collection of decision trees are used to predict values of y_i using x_i. Individually, each decision tree, would be a weak predictor of the outcome variable. However, as a collective, the decision trees may enable analysts to make accurate and reliable predictions of y_i. As a result, the method for predicting the target variable using x_i in XGBoost is referred to as decision tree ensembles. The steps below demonstrate how XGBoost was used to build a model, to predict income, using University Qualifications.

1) Load the following packages

```
library(caret)

library(xgboost)
```

2) Load the data set and remove the identifier

```
# Load data
load("data/Income_tree.RData")

# Remove identifier
Income <- Income[,-1]
```

3) Split the data set into training and test

```
# Split data into training and test
set.seed(5)

s <- createDataPartition(Income$income, p = 0.8, list=FALSE)

training <- Income[s,]

test <- Income[-s,]
```

4) Convert the data into DMatrix objects, which is the recommended input type for xgboost

```
# Convert the data to matrix and assign output variable
train.outcome <- training$income

train.predictors <- sparse.model.matrix(income ~ .,
                                        data = training
)[, -1]

test.outcome <- test$income

test.predictors <- model.matrix(income ~ .,
                                data = test
)[, -1]

# Convert the matrix objects to DMatrix objects
dtrain <- xgb.DMatrix(train.predictors, label=train.outcome)

dtest <- xgb.DMatrix(test.predictors)
```

5) Train the model

```
# Train the model
model <- xgboost(
  data = dtrain, max_depth = 2, eta = 1, nthread = 2, nrounds = 10,
  objective = "reg:linear")
```

6) Test the model

```
# Test the model
pred <- predict(model, dtest)

# Evaluate the performance of model
RMSE(pred,test.outcome)
```

7) Examine the importance of each feature in the model

```
# Examine feature importance
importance_matrix <- xgb.importance(model = model)

print(importance_matrix)

xgb.plot.importance(importance_matrix = importance_matrix)
```

8) Plot the individual trees in the model

```
# Plot the trees
# Tree 1
xgb.plot.tree(model = model, tree=0)
# Tree 2
xgb.plot.tree(model = model, tree=1)
# Tree 3
xgb.plot.tree(model = model, tree=2)
```

Chapter 6

Methods

The project evaluated the machine learning methods using:

1) The Census Teaching File, an open data set containing 1% of the person records from the 2011 Census in England & Wales.

2) Survey Data

6.1 Census Teaching File

The Census Teaching File was downloaded from the ONS website as a CSV file named "CensusTeachingFile", and was read into R using the following line of code. The data set consisted of 569,741 individuals and 18 categorical variables from the 2011 Census population.

```
# Read CSV into R
CensusRaw <- read.csv(
  file = "data/source/CensusTeachingFile.csv", skip = 1,
  header = TRUE, sep = ","
)
```

The code below specifies the packages used in the preparation, study and build of machine learning systems using the Census Teaching File.

```
library(tidyverse)
library(mice)
library(reshape2)
library(GGally)
```

```
library(Matrix)
library(xgboost)
library(caret)
library(DiagrammeR)
library(MLmetrics)
library(rpart)
library(scales)
library(knitr)
library(kableExtra)
library(DescTools)
library(dummies)
```

The steps below present the methods used to compare the performance of model based imputation (using XGBoost) with that of donor based imputation (using CANCEIS). Code chunks are provided to demonstrate how each step was carried out; with the imputable variable, economic activity, as the example.

1) Variables in the data set were renamed and recoded so that:

- Variable names were consistent with Google's R style guide
- The response categories for all variables were numeric

```
# Rename variables
Census <- plyr::rename(CensusRaw, c(
  "Person.ID" = "person.id",
  "Region" = "region",
  "Residence.Type" = "residence.type",
  "Family.Composition" = "fam.comp",
  "Population.Base" = "resident.type",
  "Sex" = "sex",
  "Age" = "age",
  "Marital.Status" = "marital.status",
  "Student" = "student",
  "Country.of.Birth" = "birth.country",
  "Health" = "health",
  "Ethnic.Group" = "ethnicity",
  "Religion" = "religion",
  "Economic.Activity" = "econ.act",
  "Occupation" = "occupation",
  "Industry" = "industry",
  "Hours.worked.per.week" = "hours.worked",
  "Approximated.Social.Grade" = "social.grade"
))
```

```
# Recode variables (dataset is mutated in order to recode variables)
Census <- Census %>% mutate_if(is.factor, as.character)

# Recode the Region variable so that it is numeric
Census$region[Census$region == "E12000001"] <- 1
Census$region[Census$region == "E12000002"] <- 2
Census$region[Census$region == "E12000003"] <- 3
Census$region[Census$region == "E12000004"] <- 4
Census$region[Census$region == "E12000005"] <- 5
Census$region[Census$region == "E12000006"] <- 6
Census$region[Census$region == "E12000007"] <- 7
Census$region[Census$region == "E12000008"] <- 8
Census$region[Census$region == "E12000009"] <- 9
Census$region[Census$region == "W92000004"] <- 10

Census$residence.type[Census$residence.type == "C"] <- 0
Census$residence.type[Census$residence.type == "H"] <- 1

Census$student[Census$student == 1] <- 0
Census$student[Census$student == 2] <- 1

Census$sex[Census$sex == 1] <- 0
Census$sex[Census$sex == 2] <- 1

Census$birth.country[Census$birth.country==1] <- 0
Census$birth.country[Census$birth.country==2] <- 1

Census <- Census %>% mutate_if(is.character, as.numeric)

Census$person.id <- as.character(Census$person.id)

Ht <- table(Census$hours.worked)
Census$hours.cont <- ifelse(Census$hours.worked == 1, runif(
  1:Ht[names(Ht) == 1],
  1, 15
),
ifelse(Census$hours.worked == 2, runif(1:Ht[names(Ht) == 2], 16, 30),
  ifelse(Census$hours.worked == 3, runif(1:Ht[names(Ht) == 3], 31, 48),
    ifelse(Census$hours.worked == 4, runif(1:Ht[names(Ht) == 4], 49, 60),
      Census$hours.worked
    )
  )
)
)
```

```r
save(Census, file = "data/core/Census.Rda")

Census.pre.ohe <- Census

# Prepare Census data for one hot encoding
Census.pre.ohe$fam.comp[Census.pre.ohe$fam.comp == -9] <- "NCR"

Census.pre.ohe$health[Census.pre.ohe$health == -9] <- "NCR"

Census.pre.ohe$ethnicity[Census.pre.ohe$ethnicity == -9] <- "NCR"

Census.pre.ohe$religion[Census.pre.ohe$religion == -9] <- "NCR"

Census.pre.ohe$econ.act[Census.pre.ohe$econ.act == -9] <- "NCR"

Census.pre.ohe$occupation[Census.pre.ohe$occupation == -9] <- "NCR"

Census.pre.ohe$industry[Census.pre.ohe$industry == -9] <- "NCR"

Census.pre.ohe$social.grade[Census.pre.ohe$social.grade == -9] <- "NCR"
```

A preview of the data set is provided below.

person.id

region

residence.type

fam.comp

resident.type

sex

age

marital.status

student

birth.country

health

ethnicity

religion

econ.act

occupation

industry

hours.worked

social.grade

hours.cont

7394816

1

1

2

1

1

6

2

1

0

2

1

2

5

8

2

-9

4

-9.00000

7394745

1

1

5

1

0

4

1

1
0
1
1
2
1
8
6
4
3
52.97915
7395066
1
1
3
1
1
4
1
1
0
1
1
1
1
1
6
11
3
4
37.80685
7395329

1
1
3
1
1
2
1
1
0
2
1
2
1
7
7
3
2
39.52916
7394712
1
1
3
1
0
5
4
1
0
1
1
2

1

1

4

3

2

31.29265

7394750

1

1

2

1

0

6

2

1

0

2

1

1

1

9

2

3

3

31.04491

2) For the purposes of training and testing a machine learning system, the data was divided into training and test data sets using the following code.

```
# Randomly select 80% of Census units and split into Train and Test data
set.seed(5)
Census80 <- sample(1:nrow(Census_ohe), 0.8 * nrow(Census_ohe), replace = FALSE)
Census20 <- setdiff(1:nrow(Census_ohe), Census80)
```

```
Census.train.ohe <- Census_ohe[Census80, !(names(Census_ohe)=="hours.worked")]
Census.test.ohe <- Census_ohe[Census20, !(names(Census_ohe)=="hours.worked")]

Census.train.label <- Census[Census80, !(names(Census)=="hours.worked")]
Census.test.label <- Census[Census20, !(names(Census)=="hours.worked")]

save(Census.train.ohe, file = "data/ohe/Census.train.ohe.Rda")
save(Census.test.ohe, file = "data/ohe/Census.test.ohe.Rda")

save(Census.train.label, file = "data/label/Census.train.label.Rda")
save(Census.test.label, file = "data/label/Census.test.label.Rda")
```

The intention was to build models to predict a selection of variable using training data, which had no missingness. This model would then be evaluated with respect to its accuracy and generalisability using a test data set, which would have missingness. The Census Teaching File was a complete data set. As a result, missingness was simulated in the test data set, and the imputation models (derived for each variable) were evaluated with regards to how well they predicted the true values.

Models were tested for the following variables:
- Economic activity (a multi-class variable) - Hours worked (a derived continuous variable)
- Social Grade (a multi-class variable)
- Student status (a binary variable) A more detailed description of all the variables can be found here.

3) The distribution of the imputable variable was studied

```
# Study the variable: How many units in each category?
EAt <- table(Census$econ.act)

EAt

# What is the distribution of the variable: Remove NCR and plot to look at distribution
g <- ggplot(Census[!Census$econ.act == -9, ], aes(econ.act))

g + geom_bar() + scale_x_discrete(
  name = "Economic Activity",
  breaks = pretty_breaks()
)
```

4) The data set was cleaned for model training; the personal identifier and the categorical hours worked variable were removed. Moreover, units that

were classified as no code required for the imputable variable were removed from the training and test data sets

5) Missingness was simulated in the test data set

```
# Purpose: Simulate missingness in one hot encoded dataset (with respect to imputable
# Input: Dataset without missingness
# Output: Dataset with missingness
# Parameters for the function are included below
## dat: Name of input dataset
## ident: Name of unique identifier in the dataset
## var.aux: List of auxiliary variables in dataset
## ohe.vars: List of variables that have been one hot encoded
## imp.vars: List of imputable variables
## var: Name of imputable variable to have increased missingness for
## ohe: An indicator of whether imputable variable (in var) is one hot encoded
## exc: Exclude rows that are not eligible to answer imputable item
## crop: Exclude columns not eligible to answer imputable item (only eligible if imputa
SimMissV <- function(dat,ident,var.aux,ohe.vars,imp.vars,var,ohe,exc,drop){

  # Order dataset by identifier
  dat <- dat[order(dat[,as.character(ident)]),]

  # Remove all instances where imputable variable is NCR
  if(ohe==1){
    dat.tidy <- subset(dat, !(grepl(exc, dat[,drop])))
    dat.tidy <- dat.tidy[,!names(dat.tidy)==drop]
  } else{
    dat.tidy <- subset(dat, !(grepl(exc, dat[,var])))
  }

  # Remove the personal identifier in order to stimulate missingness
  person.id <- dat.tidy[,as.character(ident)]
  dat.tidy <- dat.tidy[, -1]

  # Subset dataset to auxiliary variables
  dat.aux <- sub.dat.ohe(dat=dat.tidy,
                                varlist=c(var.aux))

  #Setting up two missingness patterns for the auxiliary variables
  #One pattern for core auxiliary variables
  #One pattern for other auxiliary variables
  pattern_core <- c(rep(0,28), rep(1,46))
  pattern_oth <- c(rep(1,28), rep(0,46))
  pattern_aux <- as.matrix(rbind(pattern_core, pattern_oth))
```

```
  dat.aux.amp <- ampute(dat.aux, prop = 0.4, mech = "MAR",
                                set.seed(5), pattern=pattern_aux, freq=c(0.2,0.8))

  dat.aux.miss <- dat.aux.amp$amp

  instance_sub <- setdiff(imp.vars,var)

  # Subset data to mputable variables
  dat.imp <- sub.dat.ohe(dat=dat.tidy,
                                 varlist=c(
                                   var,
                                   instance_sub))

  #Setting up two missingness patterns for the imputable variables
  #One pattern for the imputable variable of interest
  #One pattern for other imputable variables
  iteration <- select_vars(names(dat.imp), starts_with(var, ignore.case = TRUE))

  pattern_pr <- c(rep(0,length(iteration)), rep(1,eval(ncol(dat.imp)-length(iteration))))
  pattern_sc <- c(rep(1,length(iteration)), rep(0,eval(ncol(dat.imp)-length(iteration))))
  pattern_imp <- as.matrix(rbind(pattern_pr, pattern_sc))

  dat.imp.amp <- ampute(dat.imp, prop = 0.5, mech = "MAR",
                                set.seed(5), pattern=pattern_imp, freq=c(0.8,0.2))

  dat.imp.miss <- dat.imp.amp$amp

  dat.tidy.miss <- cbind(dat.imp.miss, dat.aux.miss)

  # Convert missing cases to -999
  dat.tidy.miss[is.na(dat.tidy.miss)] <- -999

  instance_miss <- setdiff(ohe.vars,var)

  dat.tidy.miss <- ohe.miss(dat=dat.tidy.miss,
                              varlist=c(instance_miss))

  dat.miss <- cbind(person.id, dat.tidy.miss)

  # Return dataset with missingenss
  return(dat.miss)


}
```

5) A model was built using the training data and used to predict values in the test data

```
# Purpose: Train XGBoost model to impute variable
# Input: Dataset with multiple auxiliary variables and single imputable variable
# Output: Model for using auxiliary variables to predict imputable variable
# Parameters for the function are included below
## train: Name of training dataset
## test.orig: Name of test dataset without missingess
## test.miss: Name of test dataset with missingness simulated
## ident: Name of unique identifier in dataset
## cat: An indicator if imputable variable is categorical. cat = 1 if categorical and
## var: Name of imputable variable
## exc: Values of imputable variable to exclude from model training
## col_order: The order in which to arrange the features in the training and test matri
## objective: The objective function to use in the XGBoost model
## max_depth: Maximum depth of trees for the XGBoost model
## eta: Learning rate for the XGBoost model
## nrounds: Number of rounds (i.e. trees) to use in the XGBoost model
## num_class: Number of classes in categorical variable + 1. Only use if imputable vari
trainDT <- function(train, test.orig, test.miss, ident, cat, var, exc, col_order,
                    objective, max_depth, eta, nrounds, num_class){
  ## ---- tidy-data
  # Sort all test datasets by identifier to ensure final comparison is valid
  test.full <- test.orig[order(test.orig[,as.character(ident)]),]

  test.miss <- test.miss[order(test.miss[,as.character(ident)]),]

  # Tidy/treat the training and test datasets
  # Remove units with NCR codes and remove the personal identifier
  train.tidy <- subset(train, !(grepl(exc, train[,var])), select = col_order)

  test.full.tidy <- subset(test.full, !(grepl(exc, test.full[,var])), select = col_orde

  test.miss.tidy <- subset(test.miss, select = col_order)

  train.tidy <- train.tidy[, col_order]

  test.full.tidy <- test.full.tidy[, col_order]

  test.miss.tidy <- test.miss.tidy[, col_order]

  ## ---- missingness-examine
  # Study the test dataset with missingness: How much missingness per variable?
  NumberMissing <- sapply(test.miss.tidy, function(y) sum(length(
```

```
  which((y)==-999)
)))

TestNumberMissing <- data.frame(NumberMissing)

## ---- train-model
# Train the model
# Convert the data to matrix and assign output variable
train.outcome <- train.tidy[,var]

train.ex <- train.tidy[,!(names(train.tidy)==var)]

train.predictors <- sparse.model.matrix(train.tidy[,var] ~ .,
                                        data = train.ex
)[, -1]

test.outcome <- test.miss.tidy[,var]

test.ex <- test.miss.tidy[,!(names(test.miss.tidy)==var)]

test.predictors <- model.matrix(test.miss.tidy[,var] ~ .,
                                data = test.ex
)[, -1]

# Convert the matrix objects to DMatrix objects
dtrain <- xgb.DMatrix(train.predictors, label = train.outcome)

dtest <- xgb.DMatrix(test.predictors, missing = -999)

# Train a model using training set
if(objective == "multi:softmax"){
  model <- xgboost(
    data = dtrain, max_depth = max_depth, eta = eta, nthread = 2, nrounds = nrounds,
    objective = "multi:softmax", num_class = num_class, missing = -999
  )
} else if (objective == "reg:linear"){
  model <- xgboost(
    data = dtrain, max_depth = max_depth, eta = eta, nthread = 2, nrounds = nrounds,
    objective = "reg:linear", missing=-999
  )
} else {
  model <- xgboost(
    data = dtrain, max_depth = max_depth, eta = eta, nthread = 2, nrounds = nrounds,
    objective = "reg:logistic", missing=-999
  )
```

```
}

# Examine feature importance
importance_matrix <- xgb.importance(model = model)

print(importance_matrix)

plot.matrix <- xgb.plot.importance(importance_matrix = importance_matrix)

## ---- test-model
# Test the model
if(objective=="reg:logistic"){
predicted <- ifelse(
  predict(model, dtest, missing = -999, na.action = na.pass) > 0.5, 1, 0)
} else {
predicted <- predict(model, dtest, missing = -999, na.action = na.pass)
}

## ---- eval-model
# Evaluate performance of XGBoost model
# Compare versions of the outcome variable (Actual, Predicted, Missing)

if(cat==1){
  actuals <- test.full.tidy[,var]

  missing <- test.miss.tidy[,var]

  compareVar <- tibble(
    Actuals = actuals, Predictions = predicted,
    Missing = missing
  )

  compareVar$PI <- ifelse(compareVar$Missing==-999,
                          compareVar$Predictions,
                          compareVar$Actuals)

  compareMissing <- compareVar[compareVar$Missing == -999, ]

  compareMissing$indicator <- ifelse(compareMissing$Actuals ==
                                       compareMissing$Predictions,"Correct", "Wrong"

  output <- list(TestNumberMissing,
                 model,
                 importance_matrix,
                 plot.matrix,
```

```
                  predicted,
                  compareVar,
                  compareMissing)

  } else{
    actuals <- test.full.tidy[,var]

    missing <- test.miss.tidy[,var]

    compareVar <- tibble(
      Actuals = actuals, Predictions = predicted,
      Missing = missing)

    compareVar$PI <- ifelse(compareVar$Missing==-999,
                            compareVar$Predictions,
                            compareVar$Actuals)

    compareMissing <- compareVar[compareVar$Missing == -999, ]

    # Using Mean Absolute Error and Root Mean Square error to evaluate predictions
    MAE_XG <- MAE(compareMissing$Predictions,compareMissing$Actuals)

    RMSE_XG <- RMSE(compareMissing$Predictions,compareMissing$Actuals)

    output <- list(TestNumberMissing,
                  model,
                  importance_matrix,
                  plot.matrix,
                  predicted,
                  compareVar,
                  compareMissing,
                  MAE_XG,
                  RMSE_XG)

  }

  return(output)

}
```

6) Predictions from the test data were evaluated using graphs and quality metrics

```
# Save feature importance plot
png(filename="images/featuresXG_econAct.png")
xgb.plot.importance(importance_matrix = econAct[[3]])
dev.off()

# Examine quality metrics
compareMissing <- econAct[[7]]

counts <- table(compareMissing$indicator)

counts.melt <- melt(counts)

counts.melt$Outcome <- counts.melt$Var1

accuracy_plot <- ggplot(data=counts.melt, aes(x=Outcome, y=value)) +
  geom_bar(stat="identity") + ggtitle("Accuracy of Predictions: XGBoost")

accuracy_plot

ggsave("images/accuracyXG_econAct.png")

confusion_XGBoost <- confusionMatrix(
  factor(compareMissing$Actuals, levels = 1:9),
  factor(compareMissing$Predictions, levels = 1:9)
)

qplot(Actuals, Predictions,
      data = compareMissing,
      geom = c("jitter"), main = "XGBoost: predicted vs. observed in test data",
      xlab = "Observed Class", ylab = "Predicted Class"
) + scale_x_discrete(limits=c("1","2","3","4","5","6","7","8","9")
) + scale_y_discrete(limits=c("1","2","3","4","5","6","7","8","9"))

ggsave("images/EAXGBoostqplot.png")

# What is the distribution of the variable before and after imputation
Compare_val <- econAct[[6]]

actuals <- ggplot(econAct[[6]], aes(Actuals))

actuals + geom_bar() + ggtitle("True Distribution") +
  scale_x_discrete(
    name = "Economic Activity",
    breaks = pretty_breaks()
  )
```

```
ggsave("images/actuals_socialGrade.png")

actuals <- table(Compare_val$Actuals)

actuals <- as.data.frame(actuals)

actuals <- plyr::rename(actuals, c(
  "Var1" = "econ.act",
  "Freq" = "actuals"))

imputed <- table(Compare_val$PI)

imputed <- as.data.frame(imputed)

imputed <- plyr::rename(imputed, c(
  "Var1" = "econ.act",
  "Freq" = "imputed"))

plot.dat <- merge(actuals, imputed, by = "econ.act")

plot.dat.melt <- melt(plot.dat, id="econ.act")

plot.dat.melt <- plyr::rename(plot.dat.melt, c(
  "variable" = "output"))

predictions <- ggplot(plot.dat.melt, aes(x=econ.act,y=value,fill=output)) +
  geom_bar(stat="identity", position = "identity", alpha=.5)

predictions + ggtitle("Post Imputation: XGBoost") +
  scale_x_discrete(
    name = "Economic Activity",
    breaks = pretty_breaks()
  )

ggsave("images/PIXG_econAct.png")

# Save model
trainEA_v1 <- econAct[[2]]

xgb.save(trainEA_v1, "models/XGBoost/xgboost.econAct")

# Save predicted values
predicted.econAct <- econAct[[5]]

save(predicted.econAct, file = "data/predicted/XGBoost/predicted.econAct.RData")
```

7) Donor based imputation was carried out on the test data (with missingness):

   i) CANCEIS: One round of CANCEIS selected matching variables based on a correlation matrix. Variables that had a correlation coefficient of |0.4| or greater were included as matching variables in the CANCEIS imputation specification. All variables were given the same weight.

   ii) Mixed Methods: One round of CANCEIS selected matching variables based on the feature importance figures from the XGBoost model. The six most important variables from the feature importance output were selected as matching variables; with more important variables assigned a larger weight.

```
# Purpose: Create a CANCEIS input dataset with matching variables for a singe imputabl
# Input: Dataset with multiple auxiliary variables and single imputable variable
# Output: Dataset with most highly correlated auxiliary variables and single imputable
# Parameters for the function are included below
## dat.cor: Name of dataset to use to derive correlation coefficients between imputable
## dat.miss: Name of dataset with missingness simulated
## var: Name of imputable variable
## exc: Exclusion condition for variables
## varlist: List of auxiliary variables to consider for selecting matching variables
CANCEIS.in <- function(dat.cor, dat.miss, var, exc, varlist) {

  dat.cor[dat.cor==exc] <- NA

  dat.tidy <- na.omit(dat.cor)

  variable <- NULL

  correlation <- NULL

  if(var=='hours.cont'){

    inputvar <- setdiff(varlist, 'hours.worked')

    for (i in 1:length(inputvar)){

      variable[i] <- inputvar[i]

      correlation[i] <- Lambda(dat.tidy[,'hours.worked'], dat.tidy[,inputvar[i]])
      }
```

```
    } else{

      inputvar <- setdiff(varlist, var)

      for (i in 1:length(inputvar)){

        variable[i] <- inputvar[i]

        correlation[i] <- Lambda(dat.tidy[,var], dat.tidy[,inputvar[i]])
        }
      }

  corlist <- data.frame(variable, correlation)

  corlist_high <- corlist[order(corlist$correlation, decreasing = TRUE),]

  corlist_high <- corlist_high[1:4,]

  CANCEIS.var <- corlist_high$variable

  var_imp <- dat.miss[,var]

  canceis.id <- 1:nrow(dat.miss)

  CANCEIS.phase1 <- cbind(canceis.id, var_imp)

  matching <- dat.miss[, names(dat.miss) %in% CANCEIS.var]

  CANCEIS.input <- cbind(CANCEIS.phase1, matching)

  CANCEIS.input <- plyr::rename(CANCEIS.input, c(
    "var_imp" = var))

  output <- list(corlist_high,
                 CANCEIS.input)

  return(output)

}
```

```
# Purpose: Create a CANCEIS input dataset with matching variables for a singe imputable variable,
# Input: Dataset with multiple auxiliary variables and single imputable variable, along with feat
# Output: Dataset with auxiliary variables (from highest scores feature importance list) and sing
# Parameters for the function are included below
## dat.miss: Name of dataset with missingness simulated
```

```
## var: Name of imputable variable
## featureList: List of features used in XGBoost model with corresponding scores
CANCEISXG <- function(dat.miss, var, featureList){

  featureList <- featureList[order(featureList$Cover, decreasing = TRUE),]

  featureImp <- featureList[1:5,]

  featureSel <- featureImp$Feature

  canceis.id <- 1:nrow(dat.miss)

  var_imp <- dat.miss[,var]

  CANCESXG.phase1 <- cbind(canceis.id, var_imp)

  CANCEISXG.matching <- dat.miss[, featureSel]

  CANCEISXG.input <- cbind(CANCESXG.phase1, CANCEISXG.matching)

  CANCEISXG.input <- plyr::rename(CANCEISXG.input, c(
    "var_imp" = var))

  return(CANCEISXG.input)

}
```

```
# Purpose: Evaluate performance of CANCEIS imputation
# Input: Dataset with CANCEIS imputation
# Output: Metrics to evaluate quality of imputation
# Parameters for the function are included below
## cat: An indicator of whether imputable variable was categorical (=1 if Yes and =0 i
## var: Name of imputable variable
## exc: Values of imputable variable to exclude from from evaluating quality of predict
## actuals: Name of dataset without missingness
## missing: Name of dataset with missingness simulated
## predicted: Name of dataset with imputed values
evalCANCEIS <- function(cat, var, exc, actuals, missing, predicted){

  actuals.CANCEIS <- subset(actuals, !(grepl(exc, actuals[,var])), select = var)

  colnames(actuals.CANCEIS) <- NULL

  missing.CANCEIS <- missing[, var]
```

```
  predicted.CANCEIS <- predicted[, var]

  compare_var_CANCEIS <- tibble(
    Actuals = actuals.CANCEIS, Predictions =
      predicted.CANCEIS, Missing = missing.CANCEIS
  )

  compare_missing_CANCEIS <- compare_var_CANCEIS[
    compare_var_CANCEIS$Missing == -999, ]

  if(cat==1){
  compare_missing_CANCEIS$indicator <- ifelse(
    compare_missing_CANCEIS$Actuals[[1]] ==
      compare_missing_CANCEIS$Predictions,
    "Correct", "Wrong"
  )

  counts_CANCEIS <- table(compare_missing_CANCEIS$indicator)

  output <- list(
    compare_var_CANCEIS,
    compare_missing_CANCEIS,
    counts_CANCEIS
  )
  } else {

    MAE <- MAE(compare_missing_CANCEIS$Predictions,compare_missing_CANCEIS$Actuals[[1]])

    RMSE <- RMSE(compare_missing_CANCEIS$Predictions,compare_missing_CANCEIS$Actuals[[1]])

    output <- list(
      compare_var_CANCEIS,
      compare_missing_CANCEIS,
      MAE,
      RMSE
    )

  }
  return(output)

}
```

8) The two rounds of donor based imputation (using CANCEIS) and the model based imputation (XGBoost) were compared using either root mean squared error, absolute error (for continuous variables) and the confusion

matrix (for categorical variables).

i) First, information and data from the previous steps was loaded into working memory

ii) Next, Each of the methods was evaluated with respect to the performance measure, and compared to either median (for continuous imputation) or mode (for categorical imputation) imputation

```
XGBoost <- round(confusion_XGBoost$overall[c('Accuracy','Kappa')],2)

CANCEIS <- round(confusion_CANCEIS$overall[c('Accuracy','Kappa')],2)

MixedMethods <- round(confusion_CANCEISXG$overall[c('Accuracy','Kappa')],2)

Mode <- c(round(econAct.modmed[[3]][['Correct']]/(econAct.modmed[[3]][['Correct']]+eco

CompareEconAct <- cbind(XGBoost, CANCEIS, MixedMethods, Mode)

save(CompareEconAct, file = "data/output/CompareEconAct.RData")
```

Chapter 7

Results

7.1 Summary statistics

Summary statistics were produced to review the pattern of responses of individuals included in the data set. Please note, that all comments that refer to respondents reflect only the respondents included in the 2011 Census Teaching File, and not descriptive statistics for the Census Population as a whole. Summary statistics produced using the data set show that:

- The majority of respondents resided in the South East and London. A relatively small proportion reside in North East and Wales.
- Almost all respondents resided in a non-communal establishment
- The majority of respondents were living in a family that was composed of a married or same-sex civil partnership couple
- Almost all respondents were usual residents at the collection address during time of collection
- There were a similar number of male and female respondents in the data set
- The majority of respondents were aged 0 to 15
- The majority of respondents were single and had never married or registered for a same-sex civil partnership
- The majority of respondents were not school children nor were they in full time study

- The majority of respondents were born in the United Kingdom
- The majority of respondents reported as being in very good health at time of collection
- With respect to Ethnicity, the majority of respondents identified as White
- With respect to religion, the majority of respondents identified as Christian. The second largest group were those that stated they had no religion.
- The two most prevalent categories for economic activity were employee and retired
- Of those that were eligible to answer the Occupation item, the majority were either in a Professional or Elementary occupation
- Of those that were eligible to answer the Industry item, the majority were employed in the Wholesale and retail trade industry
- Of those that were eligible to answer the hours worked item, the majority worked between 31 and 38 hours per week
- Of those eligible to answer the Social Grade item, the majority would be classed into the Supervisory, Clerical, and Junior Managerial social group

```
# Study the data: 1) How many units, 2) How many attributes? and
# 3) How many missing units?
str(Census)

summary(Census)

sapply(apply(Census[, c(-1, -19)], 2, table), function(x) x / sum(x))

missing_values <- sapply(Census, function(y) sum(length(which(is.na(y)))))

missing_values <- data.frame(missing_values)

# Look for relationship between variables
ggcorr(Census[, -1],
       nbreaks = 8, palette = "RdGy",
       label = TRUE, label_size = 3, label_color = "white"
)

ggsave("images/cor_all.png")
```

Bar charts were used to review the distribution of responses for each categorical variable in the complete, training and test data sets (see Figures 2 to 4). As

expected, there was a similar response pattern for a each variable between the complete, training and test data sets.

```
# Compare the test and training datasets
str(Census.train.label)

str(Census.test.label)

summary(Census.train.label)

summary(Census.test.label)

sapply(apply(Census.train.label[, c(-1, -19)], 2, table), function(x) x / sum(x))

sapply(apply(Census.test.label[, c(-1, -19)], 2, table), function(x) x / sum(x))

# Plot distribution of variables
Census.graph <- subset(Census, select=-hours.cont)

melt.Census <- melt(Census.graph)

melt.Census <- melt.Census[!melt.Census$value==-9,]

head(melt.Census)

ggplot(data = melt.Census, aes(x = value)) +
  geom_bar() +
  facet_wrap(~variable, scales = "free")

ggsave("images/dist_all.png")

# Plot distribution of variables
Census.train.graph <- subset(Census.train.label, select=-hours.cont)

melt.Census.train.graph <- melt(Census.train.graph)

melt.Census.train.graph <- melt.Census.train.graph[!melt.Census.train.graph$value==-9,]

head(melt.Census.train.graph)

ggplot(data = melt.Census.train.graph, aes(x = value)) +
  geom_bar() +
  facet_wrap(~variable, scales = "free")

ggsave("images/dist_train.png")
```

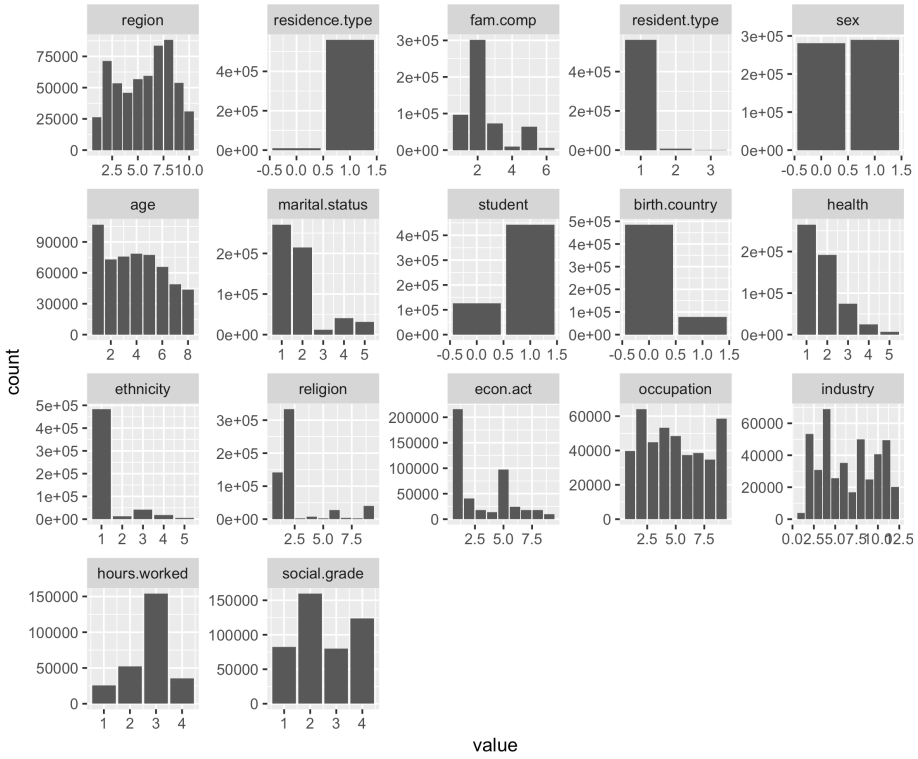

Figure 7.1: Figure 2. Distribution of responses for variables in complete Census Teaching File

```
Census.test.graph <- subset(Census.test.label, select=-hours.cont)

melt.Census.test.graph <- melt(Census.test.graph)

melt.Census.test.graph <- melt.Census.test.graph[!melt.Census.test.graph$value==-9,]

head(melt.Census.test.graph)

ggplot(data = melt.Census.test.graph, aes(x = value)) +
  geom_bar() +
  facet_wrap(~variable, scales = "free")

ggsave("images/dist_test.png")
```

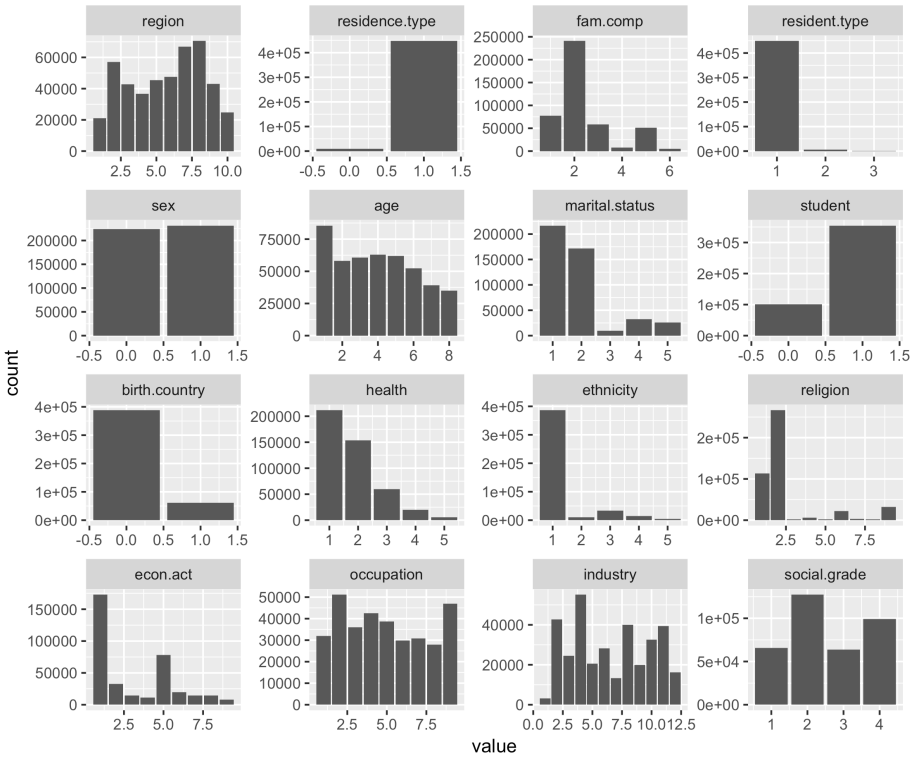

Figure 7.2: Figure 3. Distribution of responses for variables in training dataset

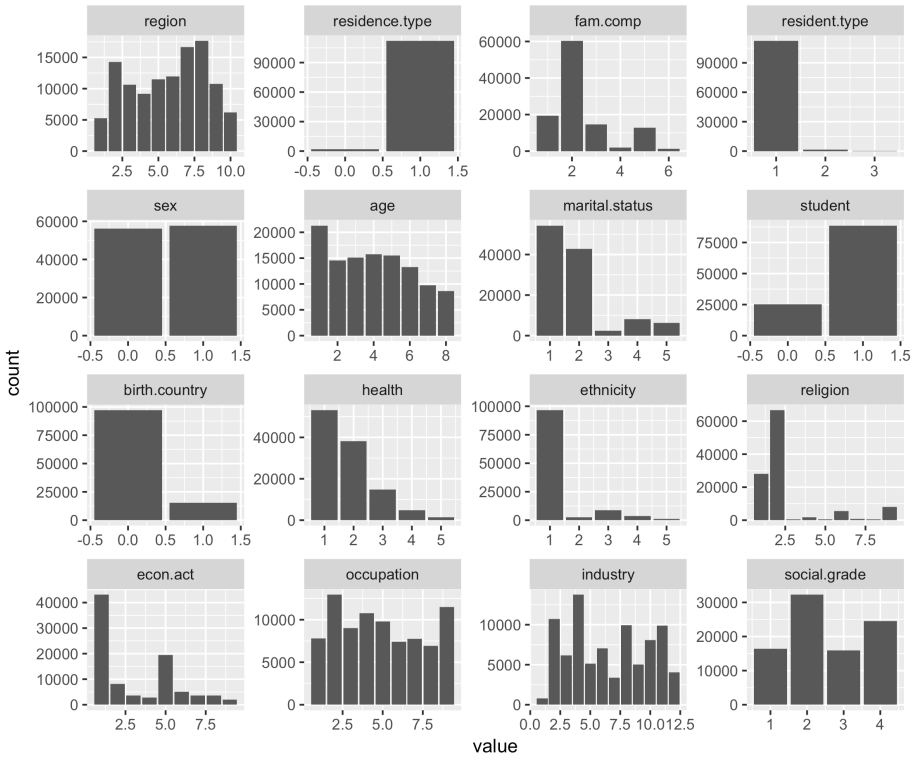

Figure 7.3: Figure 4. Distribution of responses for variables in test dataset

7.2 Comparison of imputation methods

The results for the different imputation methods are presented for each imputable variable. Performance measures were selected based on the type of imputable variable used (i.e. categorical or continuous). Please refer to the links below for guidance on interpreting the performance measures:

- Root mean squared error and mean absolute error (for continuous variables)
- Confusion matrix (for categorical variables)

For the categorical imputable variables, plots of the Observed vs Predicted are provided to give an indication of which categories each method predicted relatively well.

7.2.1 Economic Activity

The results shows that:

- XGBoost predicted economic activity with greater accuracy relative to donor and mode imputation
- Compared to donor based methods, the XGBoost model appeared to have greater sensitivity for the different classes of economic activity. That is, for any given class of economic activity, the model based approach was more likely to predict the correct response relative to donor based methods.
- The Mixed Methods model was the least accurate imputation method for the multi-class variable, economic activity.

XGBoost

CANCEIS

MixedMethods

Mode

Accuracy

0.78

0.30

0.2

0.47

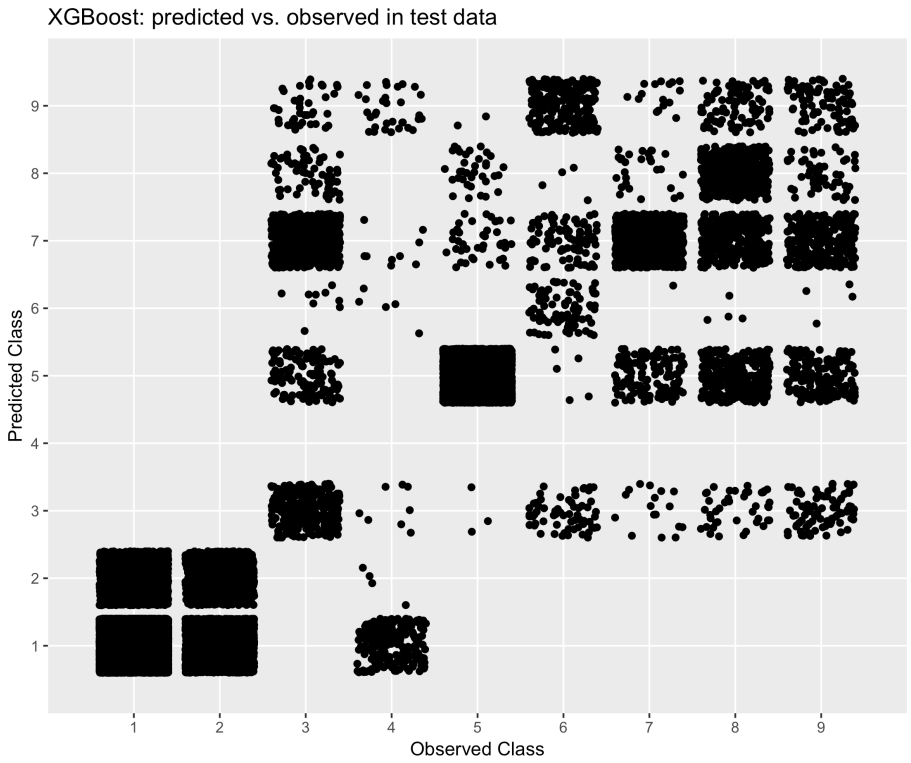

Figure 7.4: Performance of XGBoost in predicting economic activity

Kappa

0.61

0.01

0.0

NA

7.2.2 Hours worked

The results shows that:

- XGBoost predicted hours worked with greater accuracy relative to donor and mode imputation
- Median imputation and the Mixed Methods approach had a similar level of accuracy

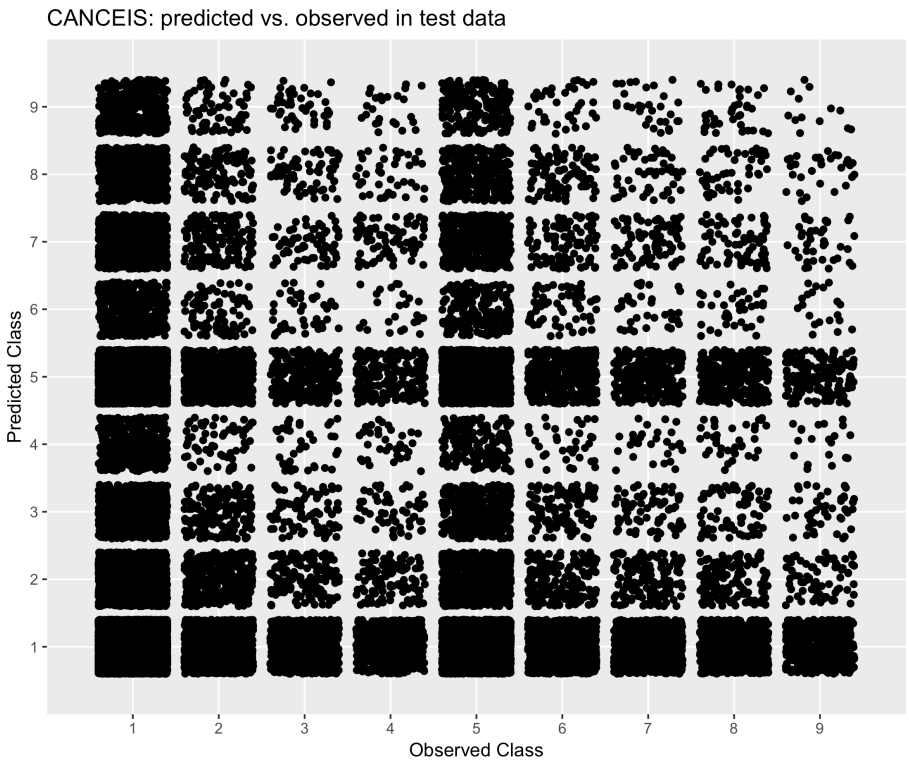

Figure 7.5: Performance of CANCEIS in predicting economic activity

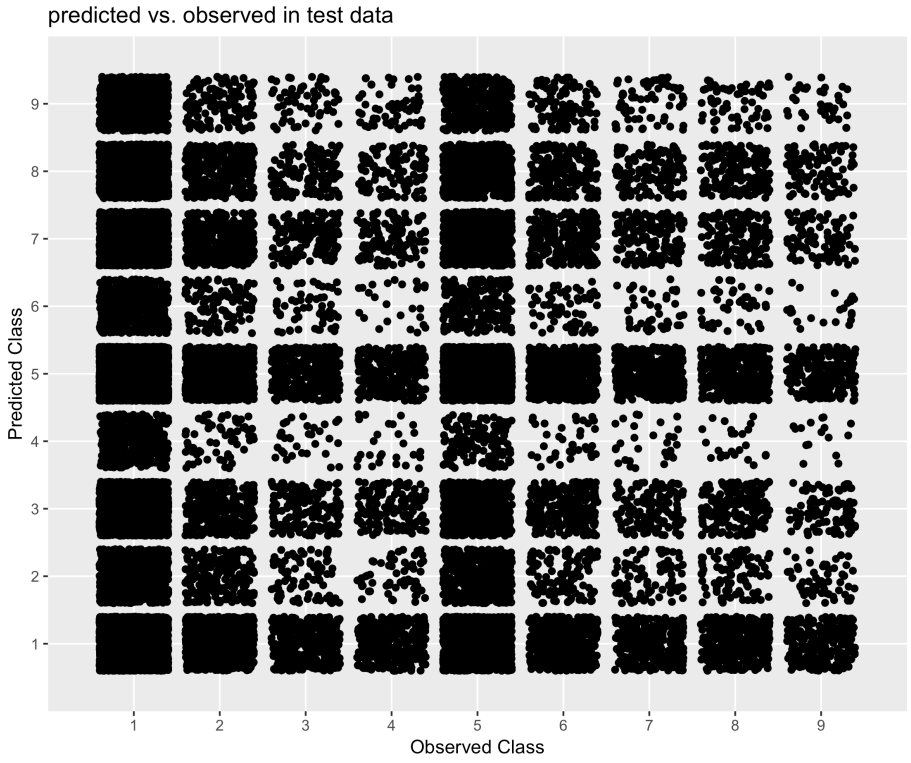

Figure 7.6: Performance of Mixed Methods approach in predicting economic activity

- Donor based imputation had the lowest level of accuracy

XGBoost

CANCEIS

MixedMethods

Median

MAE

9.71

14.64

14.62

10.51

RMSE

12.31

18.53

18.52

13.5

7.2.3 Social Grade

The results shows that:

- All three approaches performed with similar degree of accuracy, whilst out-performing mode imputation
- Compared to donor based methods, the XGBoost model appeared to have greater sensitivity for the different classes of social grade. That is, for any given class of social grade, the model based approach was more likely to predict the correct response relative to donor based methods.

XGBoost

CANCEIS

MixedMethods

Mode

Accuracy

0.62

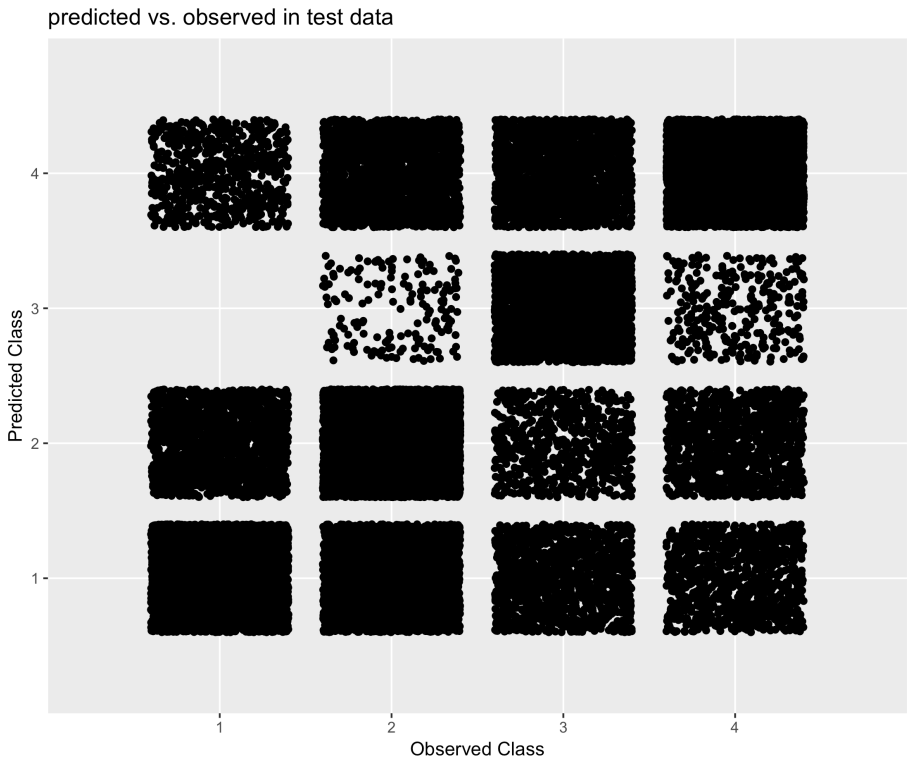

Figure 7.7: Performance of XGBoost in predicting social grade

0.28

0.28

0.36

Kappa

0.49

0.01

0.01

NA

7.2.4 Student

The results shows that:

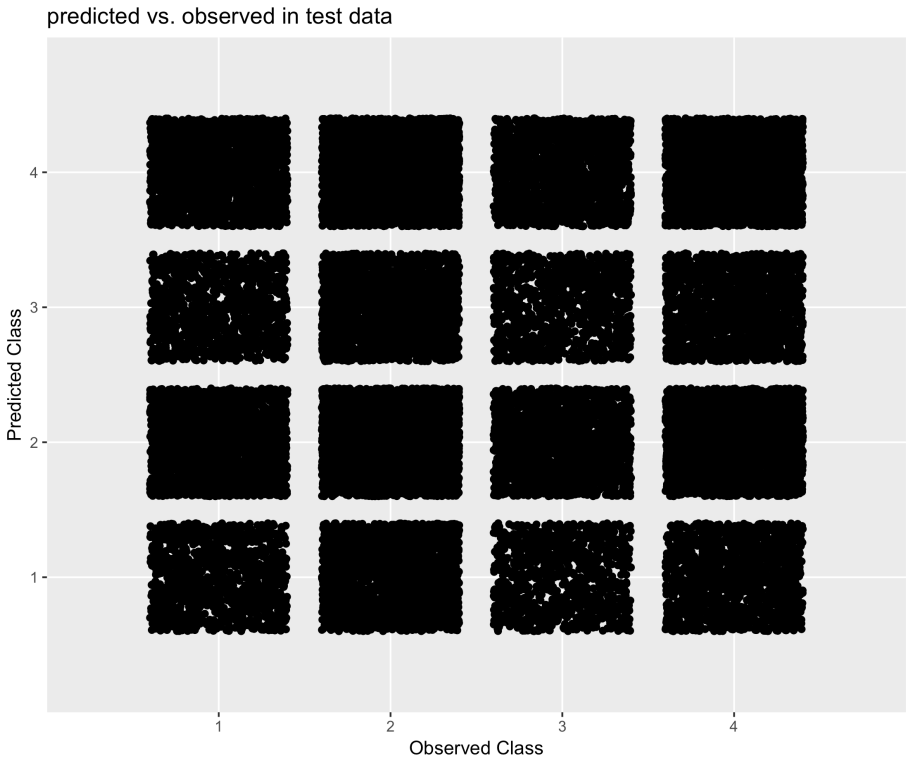

Figure 7.8: Performance of CANCEIS in predicting social grade

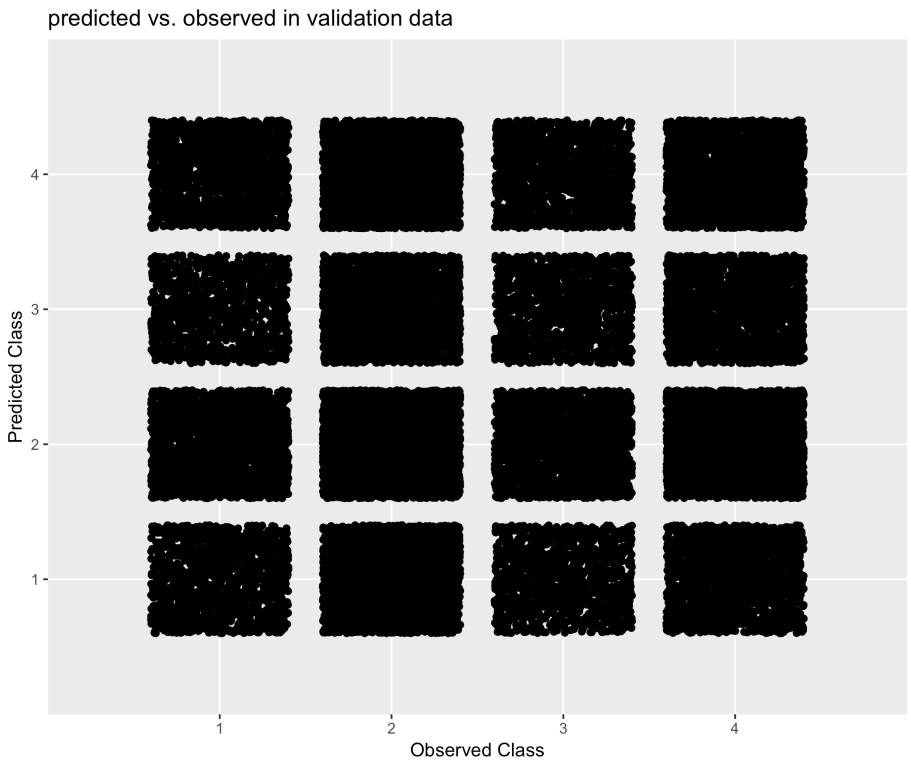

Figure 7.9: Performance of Mixed Methods approach in predicting social grade

- All three approaches performed with similar degree of accuracy, whilst out-performing mode imputation

- Compared to donor based methods, the XGBoost model appeared to have greater sensitivity for the different classes of student status. That is, for both students and non-students, the model based approach was more likely to predict the correct response relative to donor based methods.

XGBoost

CANCEIS

MixedMethods

Mode

Accuracy

0.9747038

0.6984867

0.6819554

0.7763281

Kappa

0.9192150

-0.0015287

-0.0000217

NA

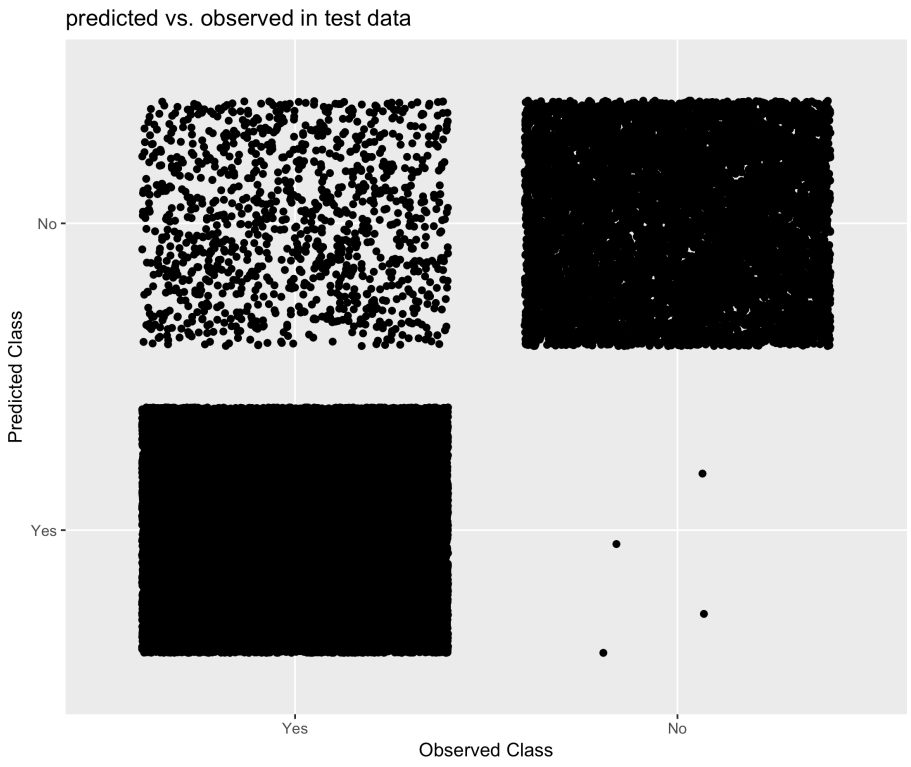

Figure 7.10: Performance of XGBoost in predicting student status

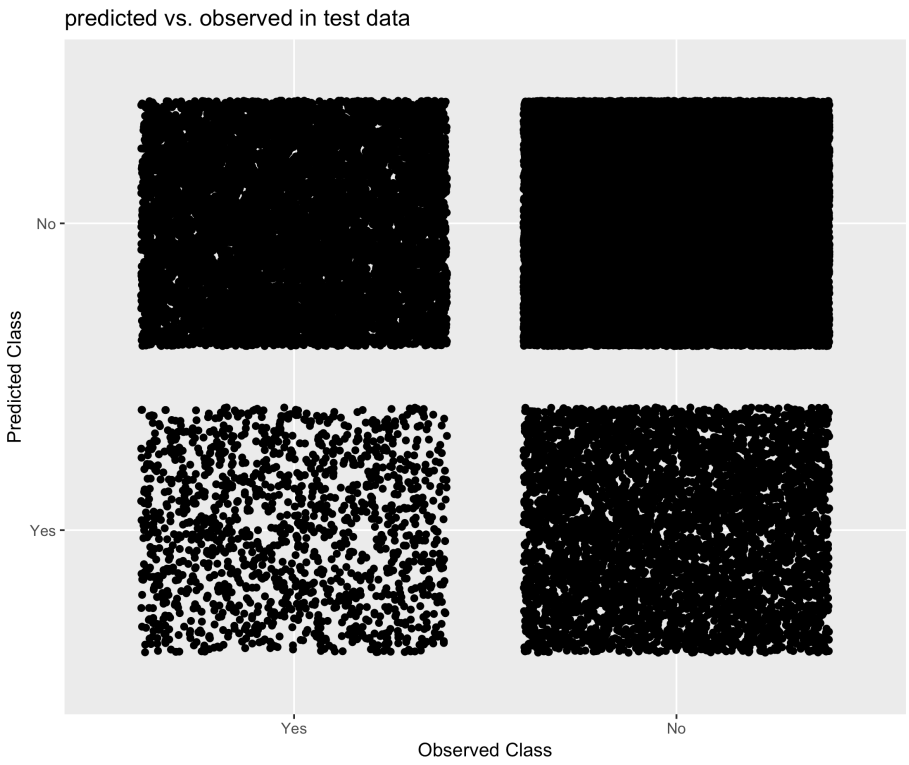

Figure 7.11: Performance of CANCEIS in predicting student status

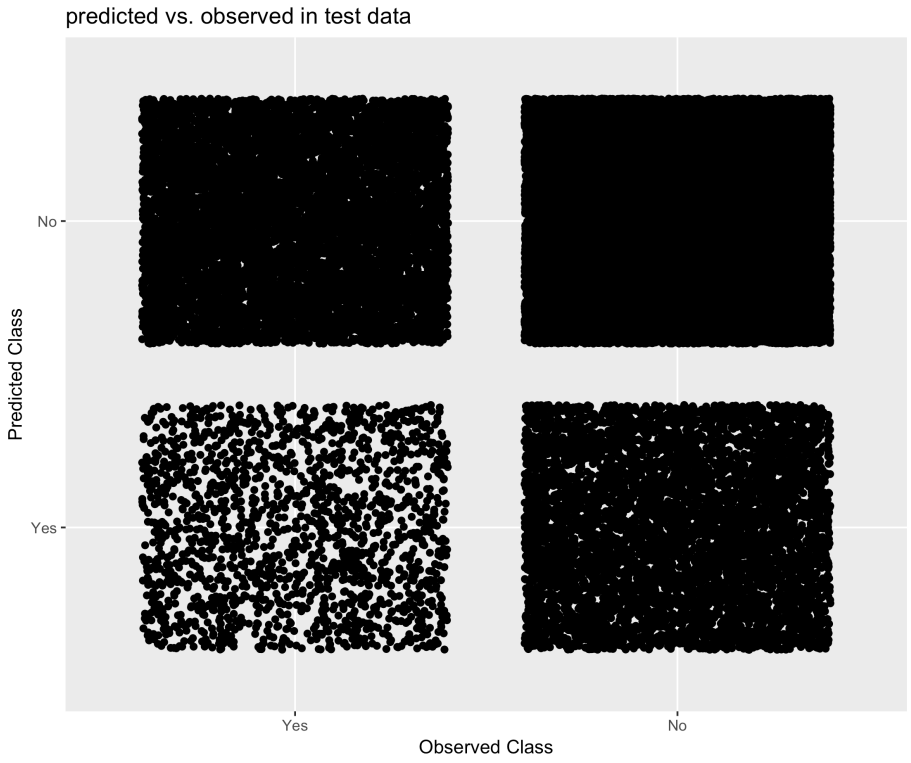

Figure 7.12: Performance of Mixed Methods approach in predicting student status

Chapter 8

Next steps

Preliminary results indicate that XGBoost performs well, relative to donor based methods, in univariate imputation. The models were especially effective at predicting the different classes of multi-class and binary variables. The ability to program an end to end imputation process is an added advantage of XGBoost; it reduces the time taken to implement an imputation method, and presents clients with the option of automating the imputation process. Current donor based methods utilise either closed, or proprietary code, which cannot easily be integrated into open source platforms.

The intention is to build on this work, and examine the efficacy of machine learning systems on household survey imputation by carrying out the following:

- Compare XGBoost, as well as Deep Learning methods (such as Autoencoders) to current donor based methods on ONS household survey data
- Study how changes to XGBoost hyper-parameters can influence the performance of the imputation model. As the current investigation, using Census data, did not iterate through different combinations of hyper-parameters, it would be of interest to see if this improves the performance of this ML system in household survey imputation.
- If time permits, it would be of interest to explore the use of XGBoost as a method to advise donor selection. That is, how the selection and importance of features could be used to identify matching variables and weights respectively.

Chapter 9

Resources

9.1 R scripts

All R scripts pertaining to this investigation are saved in the folder "R". If you are intending to run any of the code, please load all the packages and functions first, which can be found in the script "WF1_package_load.R" and "WF2_function_load.R".

The program "WF3_data_prep.R" cleans and edits the data for the investigation, whilst the scripts "WF4_missingness.R" and "WF5_data_load.R" simulate missigness and produce descriptive statistics of the data sets respectively. Scripts with the prefix "WFI" refer to programs that carry out the imputation for each variable.

9.2 Data

The folder named "data" includes:

- full Census Teaching File
- test and training data
- the predicted values from each imputation method and variable

9.3 XGBoost

The models produced for each imputable variable can be found in the following folder (located in the main directory):

- XGBoost

9.4 Donor imputation

The CANCEIS specifications used for the two rounds of donor based imputation can be found in the following folders (located in the main directory):

- CANCEIS
- MixedMethods