STAT 222: Project 4
Countable Care: Modeling Women's Health Care Decisions

Yuan He, Lindsey Lee, Jin Rou New, Tianyi Zhu
University of California, Berkeley

April 12, 2015

Abstract

In DrivenData.org's Countable Care: Modeling Women's Health Care Decisions competition, the challenge is to predict probabilities that female survey respondents used 14 different health care services in the 12 months preceding the survey based on survey data from a nationally representative sample of women in the United States. Our best solution using a basic missing value imputation method and a gradient boosting machine gave a log loss score of 0.2613, putting us at rank 30 on the leaderboard out of 483 competitors on April 12, 2015, two days before the end of the competition. This report details our entire process including data exploration, data processing and modeling with a variety of methods.

Keywords: countable care, Driven Data competition, women's health, machine learning, gradient boosting machine, random forest

1 Introduction

The competition we have chosen to work on is DrivenData.org's Countable Care: Modeling Women's Health Care Decisions. The link to the competition can be found at: http://www.drivendata.org/competitions/6/. DrivenData is effectively Kaggle for social issues, and this competition is a partnership with Planned Parenthood Federation of America.

2 Data

2.1 Data description

The data consists of responses by women in United States to the National Survey of Family Growth carried out by the United States Center for Disease Control and Prevention. Questions in the survey span topics such as demographics, marriage and reproductive health. There are a total of 14,644 observations in the training set and 3,661 observations in the test set, with each observation representing an individuals responses for one release of the survey, and a total of 1378 features, including the survey round and 116 numeric (rescaled to the interval [0, 1]), 211 ordinal and 1050 categorical features that are responses to survey questions. The data has been obfuscated, so it is not given what each feature corresponds to in real terms.

Data were given for 3 survey releases. Dates were not given for these survey releases (while the years in which this survey was conducted are easy to check, the competition prohibits any use of external information) and the chronological order of the survey releases is also unknown (i.e. the chronological order of the survey releases is not necessarily a, b, c). The number of women for which we have data in the training and test sets for each survey release is shown in Table 1.

	release_a	release_b	release_c	Total
Training set	6982	4477	3185	14644
Test set	1736	1141	784	3661

Table 1: **Number of women in training and test sets for each survey release.**

For each woman in the training set, a value of 1 or 0 was given for each of 14 health care services depending on whether she used that health care service in the 12 months preceding the survey, with 1 corresponding to yes. Again, this data is also obfuscated so it is unknown what any of the health care services are. The prediction objective is to predict the probability for each women in the test set of going to a healthcare provider for each of the 14 services. In other words, 3,661 (women) x 14 predicted probabilities are required.

The distribution of the number of health care services used by each woman in the past year is shown in Fig 1. This is a long-tailed distribution with more than half the women using 3 or less services and very few using more than 5 services.

The proportion of women using each health care service is visualized in Fig 2 across all survey releases and by survey release. The relative usage of the health care services is relatively consistent over time/across different survey releases, except for services d and n, for which the proportion is 0 in survey release b. Since the chronological order of the survey releases are not given, there are two possible hypotheses for this. The first is that survey release b is the earliest or latest and services d and n were not included as options in the survey question about health care services used, e.g. because these services were not available at that point in time. The second is that these services are simply so rarely used that the women selected to be in training set do not happen to have used them. In the first case, women in the test set in survey release b would then have a probability of 0 of having used services d and n, but not in the second case.

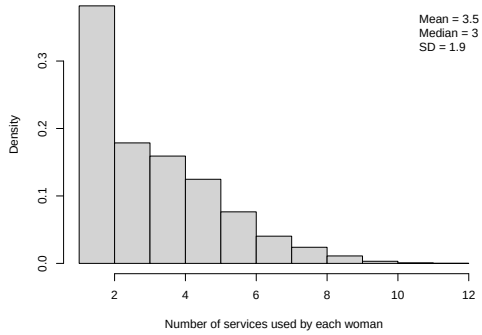

Figure 1: **Distribution of number of health care services used by each woman.**

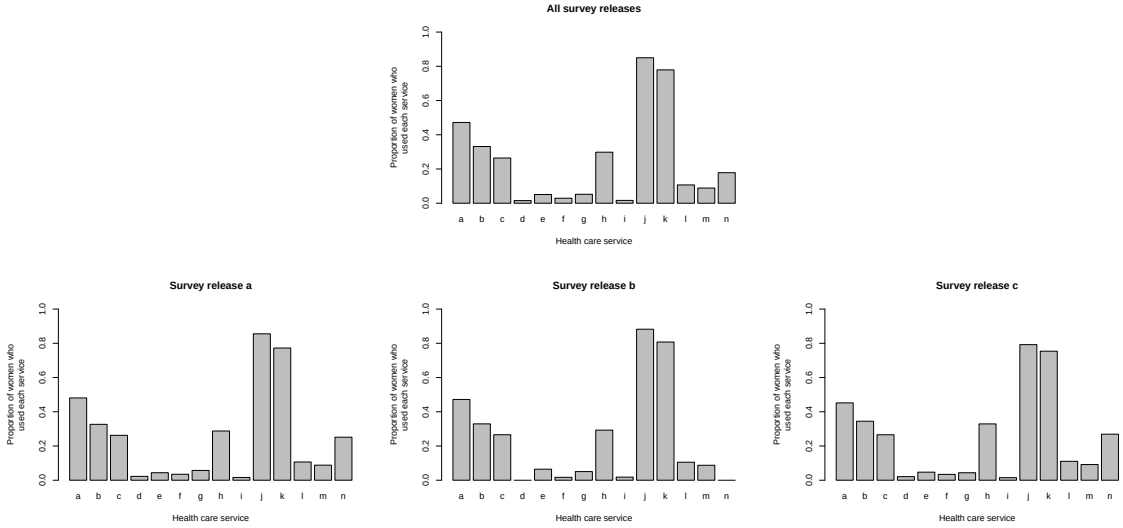

Figure 2: **Proportion of women who used each health care service in the 12 months before the survey across all survey releases (top) and in each survey release (bottom).**

2.2 Data problems

The data is very sparse, with 82.6% of the data missing in the train set. There are many features with a high proportion of missing values as shown in Fig 3. This is because many survey questions depend on the response to previous survey questions and may be skipped. Based on the description on the competition website, this seems to be the only reason for missing values (no mention is made of questions that may be left blank for other reasons or any cases of invalid survey responses). Clearly, these data is not missing at random. Given these, case deletion was not a feasible option.

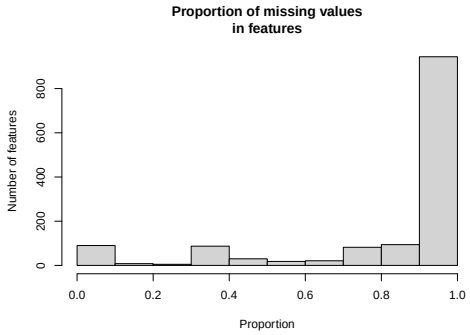

Figure 3: **Proportion of missing values in features.**

Secondly, the 14 health care services are not mutually exclusive; a woman can go to more than one service, so this is not a multiclass problem. Also, whether a women goes for different services may be dependent on each other. Fitting separate independent models for each health care service may result in poorer predicted probabilities than if the correlation structure between the different services are taken into account.

3 Methods

3.1 Data processing

3.1.1 Feature engineering

3 features were engineered and added to the data set. These are:

- Number of numeric features with missing values for each woman
- Number of ordinal features with missing values for each woman
- Number of categorical features with missing values for each woman

The distribution of each feature across all women is given in Fig 4. Given the high proportion of missing values in the data set, the number of missing values for each women could be predictive. Moreover, these features capture some of the data that is lost in the next feature pruning step.

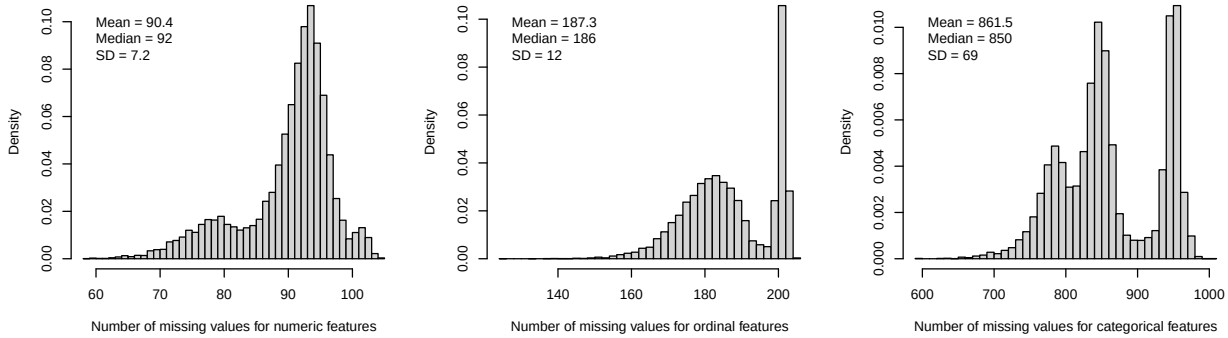

Figure 4: **Distributions of missing values for different feature types.**

3.1.2 Feature pruning

Given that there are 1378 features in the data set, it is unlikely that all of them would be strongly predictive of the dependent variables. In the first data processing step, features that would not be useful in modeling were dropped from both the training and test sets.

First of all, features that contain only missing values or one unique value (i.e. constant value for all women) were dropped from the data. There were 14 and 20 of such features respectively.

Secondly, features with a proportion of missing values exceeding a certain cut-off in the training set would be dropped, since missing value imputation is hardly meaningful for such features. If this cut-off was set to be 50%, i.e. at least half of women in the training set answered a particular question, then 1159 out of 1378 features would be dropped. In comparison, with cut-offs of 80%, 90% and 95%, 1038, 944 and 770 features would be dropped respectively. Different cut-offs were tested.

3.1.3 Missing value imputation

Missing values left were handled using different approaches for different feature types: numeric, ordinal or categorical. As there were a large number of features, features that were not purely numeric, and a lack of information on what each feature actually means, many of the missing value imputation methods, such as mean imputation or regression imputation could not be used. Instead, we did a very basic imputation as follows:

- For numeric features, missing values were set to 0.
- For ordinal features, missing values were set to -1, as the lowest category for each ordinal feature is coded as either 0 or 1 in the training set.
- For categorical features, a new category "missing" was introduced and missing values were set to this category instead.

3.1.4 Coding dummy variables

Certain algorithms (e.g. ridge regression) required that categorical features be explicitly coded into dummy variables. Firstly, values in test set were set to the most frequently-occurring category in the training set for

those with categories in the test but not the training set and those with missing values in the test but not the training set. This is because it is not possible to do prediction for an observation that has a category that is not present in the training set. For categories that appeared in the training set but not in the test set, placeholder columns (columns with all 0's) were introduced in the test set to take the position of that category. This was necessary because the training set and test set need to have the same number of columns even after introducing dummy variables.

3.2 Models

We fitted a variety of models, including:

- Benchmark: set the predicted probability for each service in the test set to be the proportion of women who used each service in the training set
- Gradient boosting machine
- Random forest
- k-nearest neigbors
- Ridge regression
- Ordinary Least Squares (OLS)

These models were fitted independently for each of the 14 health care services, ignoring any correlation among them.

3.3 Miscellaneous

As discussed in the Data section, it is possible that women in the test set in survey release b would have a probability of 0 of having used services d and n. To test our hypothesis, we produced a version of predictions with the predicted probabilities for these 2 services for women in survey release b set to 0. If our hypothesis is incorrect, our log loss score would be much worse than in the original predictions since the log loss penalizes wrong but confident predictions heavily. Otherwise, it is likely to improve our score.

3.4 Computation

All calculations were carried out in the `R` programming language, with additional functions from the `caret` package and the `glmnet` package. Annotated code is given in the Appendix in Section 5.3.

4 Results

4.1 Evaluation metric

The evaluation metric here is the logarithmic loss, defined by $-\frac{1}{n}\sum_{i=1}^{10}[y_i log(\hat{y}_i) + (1 - y_i log(1 - \hat{y}_i))]$, where y_i is the binary outcome of 0 or 1 of whether the woman used the health care service and $\hat{y}_i$ is the predicted

probability that y_i is 1. The goal is to minimize the log loss, so the top solution on the leaderboard will be the one that has the minimum log loss score on the public test set, while the top solution in the competition will be the one that has the minimum log score on the private test set. The log loss penalizes predictions that are both confident and wrong very heavily.

4.2 Prediction results

A summary of the performance of our models on the public test set is given in Table 2. The best solution on the leaderboard is bolded. The gradient boosting method and the random forest performed well as both algorithms are suited to dealing with a large number of features. k-nearest neighbors is not optimal for high dimensions, hence the poor performance. Ridge regression also performed poorly, likely also due to the high dimension of the data.

The processed data with the missing value cut-off of 90% gave the best performance, probably because it gives the right balance of not discarding too many features, of which some may be informative, and features that yield almost no information after having their (large proportion of) missing values imputed with our basic imputation method.

Data set	Model	Log loss
Original	Benchmark; proportion of women using each service in train set	0.3847
Missing value cut-off of 50%	Gradient boosting method	0.2796
Missing value cut-off of 80%	Gradient boosting method	0.2812
Missing value cut-off of 90%	**Gradient boosting method**	**0.2613**
Missing value cut-off of 95%	Gradient boosting method	0.2622
Missing value cut-off of 50%	Random forest	0.3037
Missing value cut-off of 80%	Random forest	0.3088
Missing value cut-off of 90%	Random forest	0.2868
Missing value cut-off of 50%	k-nearest neighbors	1.0622
Missing value cut-off of 80%	k-nearest neighbors	1.1002
Missing value cut-off of 50%	Ridge regression	0.5044
Missing value cut-off of 80%	Ridge regression	0.5274
Missing value cut-off of 50%	Ordinary Least Squares	0.4972

Table 2: **Summary of model performance on test set.** The best method is bolded.

4.3 Miscellaneous

We tested our hypothesis of whether the predicted probabilities for services d and n for women in survey release b should be all 0. The original version of predicted probabilities gave a log loss score of 0.2613, while the version with the predicted probabilities for services d and n for women in survey release b set to 0 gave a score of 0.2619. The difference appears to be marginal, so under these circumstances, no conclusion can be reached. It is possible that predicted probabilities in the original version were already small to begin with, as seen in Fig /reffig-svcsdn, so setting them to 0 produced little effect.

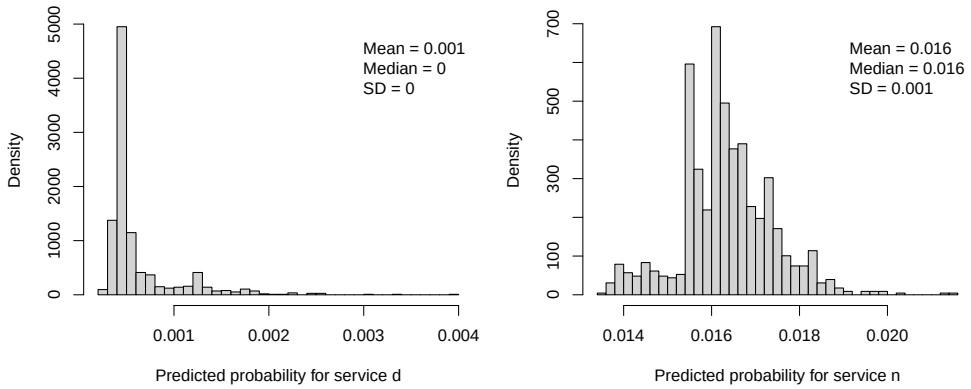

Figure 5: **Distributions of predicted probabilities for services d and n for survey release b.**

5 Discussion

5.1 Summary

As of 12 April 2015, two days before the conclusion of the competition, our best model of the gradient boosting machine reached rank 30 out of 483 competitors on the leaderboard with a log loss score of 0.2613. The competition has been ongoing for more than a month and we entered late into it, so this performance is not unacceptable.

5.2 Methods and feature pruning

Classification methods (e.g. Gradient Boosting, random forest) generally performed better than regression methods (e.g. OLS, ridge regression). This could attribute to the binary feature (i.e. being either 0 or 1) of the Y responses. Not only did regression methods give a prediction about the probabilities instead of 0's and 1's, but some predicted probabilities could be outside of the range of 0 to 1. Dealing with these over-bound predicted values could be very tricky, as discussed in the next subsection.

As for the missing value cut-offs, although a 50% cut-off generally had lower loss comparing to an 80% cut-off, the best result (with lowest log loss) was generated using a 90% cut-off. Therefore, despite saving computer time, dropping features with mostly missing values could not help with increasing prediction accuracy. In other words, dropping (seemingly) useless data had the risk of undermining completeness and validity of dataset.

5.3 Limitations and future work

Our basic missing value imputation method in which we replaced missing values with a global value for each feature type was a compromise due to the large number of missing values and the obfuscation of the data set. Given the actual meaning of the features, we would likely be able to use more sophisticated missing value imputation methods that would improve our final predictions. The same can be said of the method used to replace categories found in the training but not in the test set for categorical variables; a better method would be to use the next most similar category found in the training set, but this could not be determined based on the obfuscated data.

Another significant feature about this data set was that it had not one, but 14 dependent variables that are not mutually exclusive. Moreover, these dependent variables were likely correlated with each other. Ignoring the correlation among these variables may have caused loss of prediction accuracy. The use of multivariate methods that take into account this correlation structure might be able to increase prediction accuracy.

Finally, since ridge regression is not well-suited for classification problems, it yielded predicted probabilities outside of the range of 0 to 1. Converting these values to 0 or 1 would dramatically increase the score since log loss was used as the evaluation metric. To avoid this problem, such values were replaced with the proportions of women who used that service in the training set. However, a finer way should be adopted to accommodate such values.

Appendix

```r
#=====================================================================
# Data exploration
#=====================================================================
data_dir <- "data"
fig_dir <- "fig"
dir.create(fig_dir, showWarnings = FALSE)
prop_missing_cutoff <- 0.9

# Read in data
train_readin <- read.csv(file.path(data_dir, "train_values.csv"),
                         stringsAsFactors = FALSE, na.strings = "")
ytrain <- read.csv(file.path(data_dir, "train_labels.csv"))
test_readin <- read.csv(file.path(data_dir, "test_values.csv"),
                        stringsAsFactors = FALSE, na.strings = "")

# Column types
colnames_all <- names(train_readin)
colnames_type <- sapply(colnames_all, function(x) strsplit(x, "_")[[1]][1])
table(colnames_type[-c(1, 2)])

# Check proportion of missing values in features
prop_missing <- sapply(train_readin, function(x) mean(is.na(x)))
par(mar = c(4.5, 4.5, 4.5, 2))
pdf(file.path(fig_dir, "prop-missing-in-columns.pdf"), width = 7, height = 5)
hist(prop_missing, main = "Proportion of missing values\nin features",
     xlab = "Proportion", ylab = "Number of features", col = "lightgrey")
dev.off()

# Check original number of features, not including id
ncol(train_readin[, -1]) # 1378

# Check for features with only constant values
cols_constant <- sapply(train_readin, function(x) length(unique(x)) == 1)
sum(cols_constant) # 20

# Feature engineering
# Column types
colnames_all <- names(train_readin)
colnames_type <- sapply(colnames_all, function(x) strsplit(x, "_")[[1]][1])
table(colnames_type[-c(1, 2)])
cols_numeric <- colnames_type == "n"
cols_ordinal <- colnames_type == "o"
cols_categorical <- colnames_type == "c"

# Number of missing numeric/ordinal/categorical features
num_missing_numeric <- apply(train_readin[, cols_numeric], 1, function(x) sum(is.na(x)))
num_missing_ordinal <- apply(train_readin[, cols_ordinal], 1, function(x) sum(is.na(x)))
num_missing_categorical <- apply(train_readin[, cols_categorical], 1, function(x) sum(is.na(x)))
pdf(file.path(fig_dir, "dist-missing-values-across-women.pdf"), width = 12/1.2, 3.5/1.2)
par(mar = c(4.5, 4.5, 1, 1), mfrow = c(1, 3))
hist(num_missing_numeric, freq = FALSE, breaks = 50,
```

```
     main = "", col = "lightgrey",
     xlab = "Number of missing values for numeric features")
legend("topleft", legend = c(paste0("Mean = ", round(mean(num_missing_numeric), digits = 1),
                                    "\nMedian = ", median(num_missing_numeric),
                                    "\nSD = ", round(sd(num_missing_numeric), digits = 1))),
       bty = "n")
hist(num_missing_ordinal, freq = FALSE, breaks = 50,
     main = "", col = "lightgrey",
     xlab = "Number of missing values for ordinal features")
legend("topleft", legend = c(paste0("Mean = ", round(mean(num_missing_ordinal), digits = 1),
                                    "\nMedian = ", median(num_missing_ordinal),
                                    "\nSD = ", round(sd(num_missing_ordinal), digits = 1))),
       bty = "n")
hist(num_missing_categorical, freq = FALSE, breaks = 50,
     main = "", col = "lightgrey",
     xlab = "Number of missing values for categorical features")
legend("topleft", legend = c(paste0("Mean = ", round(mean(num_missing_categorical), digits = 1),
                                    "\nMedian = ", median(num_missing_categorical),
                                    "\nSD = ", round(sd(num_missing_categorical), digits = 1))),
       bty = "n")
dev.off()

# Check number of observations per survey release
table(train_readin$release)

# Check number of combinations of ytrain
nrow(ytrain) # 14644
nrow(unique(ytrain[, -1])) # 932

# Check number of services used across all women
numy <- apply(ytrain[, -1], 1, sum)
summary(numy)
pdf(file.path(fig_dir, "number-of-svcs-used.pdf"), width = 7, height = 5)
par(mar = c(4.5, 4.5, 1, 1))
hist(numy, freq = FALSE, main = "",
     xlab = "Number of services used by each woman", col = "lightgrey")
legend("topright", legend = c(paste0("Mean = ", round(mean(numy), digits = 1),
                                     "\nMedian = ", median(numy),
                                     "\nSD = ", round(sd(numy), digits = 1))),
       bty = "n")
dev.off()

# Check proportion of women using each service
pdf(file.path(fig_dir, "prop-women-using-each-service.pdf"),
    width = 7, height = 5)
par(mar = c(4.5, 5.5, 4.5, 1))
barplot(apply(ytrain[, -1], 2, mean), ylim = c(0, 1),
        names.arg = gsub("service_", "", colnames(ytrain[, -1])),
        main = "All survey releases", xlab = "Health care service",
        ylab = "Proportion of women who\nused each service")
dev.off()

releases <- sort(unique(train_readin$release))
```

```r
for (i in seq_along(releases)) {
  pdf(file.path(fig_dir,
                paste0("prop-women-using-each-service-survey-release-", releases[i], ".pdf")),
      width = 7, height = 5)
  par(mar = c(4.5, 5.5, 4.5, 1))
  barplot(apply(ytrain[train_readin$release == releases[i], -1], 2, mean), ylim = c(0, 1),
          names.arg = gsub("service_", "", colnames(ytrain[, -1])),
          main = paste0("Survey release ", releases[i]), xlab = "Health care service",
          ylab = "Proportion of women who\nused each service")
  dev.off()
}
```

```r
#=========================================================================
# Data processing
#=========================================================================
# rm(list = ls())
# gc()
# setwd("~/Copy/Berkeley/stat222-spring-2015/stat222sp15/projects/countable-care")
# data_dir <- "data"
# prop_missing_cutoff <- 0.9

# Read in data
train_readin <- read.csv(file.path(data_dir, "train_values.csv"),
                         stringsAsFactors = FALSE, na.strings = "")
ytrain <- read.csv(file.path(data_dir, "train_labels.csv"))
test_readin <- read.csv(file.path(data_dir, "test_values.csv"),
                        stringsAsFactors = FALSE, na.strings = "")

# Check for features with only constant values
cols_constant <- sapply(train_readin, function(x) length(unique(x)) == 1)
sum(cols_constant) # 20

# Feature engineering
# Column types
colnames_all <- names(train_readin)
colnames_type <- sapply(colnames_all, function(x) strsplit(x, "_")[[1]][1])
table(colnames_type[-c(1, 2)])
cols_numeric <- colnames_type == "n"
cols_ordinal <- colnames_type == "o"
cols_categorical <- colnames_type == "c"

# Number of missing numeric/ordinal/categorical features
num_missing_numeric <- apply(train_readin[, cols_numeric], 1, function(x) sum(is.na(x)))
num_missing_ordinal <- apply(train_readin[, cols_ordinal], 1, function(x) sum(is.na(x)))
num_missing_categorical <- apply(train_readin[, cols_categorical], 1, function(x) sum(is.na(x)))
num_missing_numeric_test <- apply(test_readin[, cols_numeric], 1, function(x) sum(is.na(x)))
num_missing_ordinal_test <- apply(test_readin[, cols_ordinal], 1, function(x) sum(is.na(x)))
num_missing_categorical_test <- apply(test_readin[, cols_categorical], 1, function(x) sum(is.na(x)))

# Check for features with proportion of missing values > prop_missing_cutoff
prop_missing <- sapply(train_readin, function(x) mean(is.na(x)))
cols_missing <- prop_missing > prop_missing_cutoff
sum(cols_missing)
```

```r
# Remaining number of features, not including id
sum(!cols_constant & !cols_missing) - 1

# Remove above features and id
train <- train_readin[, names(train_readin) != "id" & !cols_constant & !cols_missing]
test <- test_readin[, names(train_readin) != "id" & !cols_constant & !cols_missing]

# Column types
colnames_all <- names(train)
colnames_type <- sapply(colnames_all, function(x) strsplit(x, "_")[[1]][1])
table(colnames_type[-c(1, 2)])
cols_numeric <- colnames_type == "n"
cols_ordinal <- colnames_type == "o"
cols_categorical <- colnames_type == "c"

# Missing value imputation for remaining features
# Numeric features: Set as 0
for (i in 1:sum(cols_numeric)) {
  train[, cols_numeric][, i] <- ifelse(is.na(train[, cols_numeric][, i]),
                                       0, train[, cols_numeric][, i])
  test[, cols_numeric][, i] <- ifelse(is.na(test[, cols_numeric][, i]),
                                      0, test[, cols_numeric][, i])
}
# Ordinal features: Set as -1
# table(sapply(train[, cols_ordinal], min, na.rm = TRUE)) # min is 0 or 1
for (i in 1:sum(cols_ordinal)) {
  train[, cols_ordinal][, i] <- as.integer(ifelse(is.na(train[, cols_ordinal][, i]),
                                                  -1, train[, cols_ordinal][, i]))
  test[, cols_ordinal][, i] <- as.integer(ifelse(is.na(test[, cols_ordinal][, i]),
                                                 -1, test[, cols_ordinal][, i]))
}

# Categorical features
# Check for: i) features with categories in test but not train set.
# ii) features with missing values in test but not train set
cols_unknownlevels <- NULL
cols_nomissingintrain <- NULL
for (i in 1:sum(cols_categorical)) {
  if (any(is.na(test[, cols_categorical][, i])) & all(!is.na(train[, cols_categorical][, i]))) {
    print(i)
    cols_nomissingintrain <- c(cols_nomissingintrain, i)
  }
  if (any(!is.na(test[, cols_categorical][, i]) &
            !(test[, cols_categorical][, i] %in% unique(train[, cols_categorical][, i])))) {
    cols_unknownlevels <- c(cols_unknownlevels, i)
    levels_train <- unique(train[, cols_categorical][, i])
    levels_train <- ifelse(is.na(levels_train), "missing", as.character(levels_train))
    levels_test <- unique(test[, cols_categorical][, i])
    levels_test <- ifelse(is.na(levels_test), "missing", as.character(levels_test))
    if (!("missing" %in% levels_train)) {
      print(i)
      print(levels_test[!(levels_test %in% levels_train)])
      print("---")
```

```
      cols_nomissingintrain <- c(cols_nomissingintrain,
                                 rep(i, length(levels_test[!(levels_test %in% levels_train)])))
    }
  }
}

# Categorical features: Set as new category missing
for (i in 1:sum(cols_categorical)) {
  train[, cols_categorical][, i] <- as.factor(ifelse(is.na(train[, cols_categorical][, i]),
                                                     "missing", train[, cols_categorical][, i]))
  test[, cols_categorical][, i] <- as.factor(ifelse(is.na(test[, cols_categorical][, i]),
                                                    "missing",
                                                    ifelse(!(test[, cols_categorical][, i] %in%
                                                      unique(train[, cols_categorical][, i])),
                                                      "missing", test[, cols_categorical][, i])))
}

# Set values in test set to most frequently-occurring category in train set for:
# i) features with categories in test but not train set.
# ii) features with missing values in test but not train set
for (c in seq_along(cols_nomissingintrain)) {
  i <- cols_nomissingintrain[c]
  value_new <- names(which.max(table(test[, cols_categorical][, i])))
  test[, cols_categorical][, i] <- as.factor(ifelse(as.character(test[, cols_categorical][, i]) ==
                                                      "missing",
                                                    value_new,
                                                    as.character(test[, cols_categorical][, i])))
}

# For random forest, ensure that categorical features have same
# levels in train and test sets
for (i in 1:sum(cols_categorical)) {
  test[, cols_categorical][, i] <- factor(test[, cols_categorical][, i],
                                          levels = levels(train[, cols_categorical][, i]))

}

# Add engineered features to data
train$num_missing_numeric <- num_missing_numeric
train$num_missing_ordinal <- num_missing_ordinal
train$num_missing_categorical <- num_missing_categorical
test$num_missing_numeric <- num_missing_numeric_test
test$num_missing_ordinal <- num_missing_ordinal_test
test$num_missing_categorical <- num_missing_categorical_test

# Convert release variable to factor, else it throws error
train$release <- as.factor(train$release)
test$release <- as.factor(test$release)

# Convert prediction label to alphabetical factor,
# else it throws error in caret::predict
for (i in 1:ncol(ytrain))
  ytrain[, i] <- factor(ifelse(ytrain[, i] == 1, "yes", "no"))
```

```
# Save processed data to file
data <- list(train = train,
             ytrain = ytrain[, -1], # Drop id column
             test = test,
             prop_missing_cutoff = prop_missing_cutoff)
save(data, file = file.path(data_dir, paste0("data_cutoff", prop_missing_cutoff, ".rda")))

### Coding dummy variables (for Ridge, OLS and logit):
makeDummy <- function(train, test){#Make dummies for categorical, and feature 'release':
  dum_release <- model.matrix(~train[,1])
  # Convert the feature 'release' to dummy variable
  train <- cbind(dum_release[,-1], train[,-1])
  first_cat <- grep("c_", colnames(train))[1]
  train1 <- train[,c(1:(first_cat-1), (ncol(train)-2):ncol(train))]

  dum_release <- model.matrix(~test[,1])
  test <- cbind(dum_release[,-1], test[,-1])
  test1 <- test[,c(1:(first_cat-1), (ncol(test)-2):ncol(test))]

  for (i in first_cat:(ncol(train)-3)){
    if (length(levels(train[,i])) >= length(levels(test[,i]))){
      dummy <- model.matrix(~train[,i])
      train1 <- cbind(train1, dummy[,-1])
    }
    else{#Generate placeholder columns for missing levels
      extra <- length(levels(test[,i]))-length(levels(train[,i]))
      mat0 <- matrix(rep(0,nrow(train)*extra), ncol=extra)
      dummy <- model.matrix(~train[,i])
      train1 <- cbind(train1, dummy[,-1], mat0)
    }
  }

  for (i in first_cat:(ncol(train)-3)){#Repeat for test data
    if (length(levels(test[,i])) >= length(levels(train[,i]))){
      dummy <- model.matrix(~test[,i])
      test1 <- cbind(test1, dummy[,-1])
    }
    else{
      extra <- length(levels(train[,i]))-length(levels(test[,i]))
      mat0 <- matrix(rep(0,nrow(test)*extra), ncol=extra)
      dummy <- model.matrix(~test[,i])
      test1 <- cbind(test1, dummy[,-1], mat0)
    }
  }

  return(list(train1, test1))
}

# setwd("~/Copy/Berkeley/stat222-spring-2015/stat222sp15/projects/countable-care")
load("data/data.rda")
train <- data$train
test <- data$test
```

```
dummy <- makeDummy(train,test)
data_dummy_0.5 <- list(dummy[[1]], dummy[[2]], data$ytrain, 0.5)
names(data_dummy_0.5) <- c("train", "test", "ytrain", "na_cutoff")
save(data_dummy_0.5, file="data_dummy_0.5.rda")
```

```
#=========================================================================
# Modeling
#=========================================================================
rm(list = ls())
gc()
run_on_server <- TRUE ###
if (!run_on_server)
  setwd("~/Copy/Berkeley/stat222-spring-2015/stat222sp15/projects/countable-care")
data_dir <- "data"
fig_dir <- "fig"
results_dir <- "results"
dir.create(fig_dir, showWarnings = FALSE)
dir.create(results_dir, showWarnings = FALSE)
dir.create("submit", showWarnings = FALSE)
get_notifications <- ifelse(run_on_server, TRUE, FALSE)
if (get_notifications) {
  library(RPushbullet)
  # options(error = function() { # Be notified when there is an error
  #   pbPost("note", "Error!", geterrmessage(), recipients = c(1, 2))
  # })
}

write_submission <- function(probs, model_name) {
  file_path <- file.path("submit", paste0(model_name, ".csv"))
  submit <- read.csv("data/SubmissionFormat.csv")
  submit[, 2:ncol(submit)] <- probs
  if (file.exists(file_path))
    stop(paste0(file_path, " already exists!"))
  write.csv(submit, file.path(file_path), row.names = FALSE)
  message(paste0("Results written to ", file_path))
}

seed <- 12345
set.seed(seed)
library(caret)
library(e1071)
# List all models in caret
# names(getModelInfo())

# Load data
prop_missing_cutoff <- 0.9
load(file = file.path(data_dir, paste0("data_cutoff", prop_missing_cutoff, ".rda")))
train <- data$train
test <- data$test
ytrain <- data$ytrain

# Create data partitions of 80% and 20%
ntrain <- nrow(train)
```

```
train_indices <- sample(1:ntrain)[1:floor(ntrain*0.8)]
train_val <- train[-train_indices, ]

# Set up caret models
train_control <- trainControl(method = "cv", number = 10, returnResamp = "none")

mod_types <- c("gbm", "rf")
if (FALSE) {
mod <- list()
probs <- matrix(NA, nrow(test), ncol(ytrain))
for (mod_type in mod_types) {
  for (svc_index in 1:ncol(ytrain)) {
    # Train all the models with train data
    mod[[svc_index]] <- train(train[train_indices, ], ytrain[train_indices, svc_index],
                              method = mod_type, trControl = train_control)

    # Predict on test data
    probs[, svc_index] <- predict(object = mod[[svc_index]], newdata = test,
                                  type = "prob")$yes
  }
  write_submission(probs, paste0(mod_type, "_cutoff", prop_missing_cutoff))
  save(mod, file = file.path(results_dir,
                             paste0("mod_", mod_type, "_cutoff", prop_missing_cutoff, ".rda")))
  if (get_notifications)
    pbPost(type = "note",
           title = "stat222",
           body = paste0(mod_type, " done!"),
           recipients = c(1, 2))
}
}

# Ensemble the models
mod_ensemble_types <- c("glm")
mod_ensemble <- list()
probs_ensemble <- matrix(NA, nrow(test), ncol(ytrain))
for (mod_ensemble_type in mod_ensemble_types) {
  for (svc_index in 1:ncol(ytrain)) {
    for (mod_type in mod_types) {
      load(file.path(results_dir,
                     paste0("mod_", mod_type, "_cutoff", prop_missing_cutoff, ".rda")))

      # Get predictions for each model and add them back to themselves
      train_val[[paste0(mod_type, "_PROB")]] <- predict(object = mod[[svc_index]],
                                                        newdata = train_val,
                                                        type = "prob")$yes
      test[[paste0(mod_type, "_PROB")]] <- predict(object = mod[[svc_index]],
                                                   newdata = test,
                                                   type = "prob")$yes
    }
    # Run an ensemble model to blend all the predicted probabilities
    mod_ensemble[[svc_index]] <- train(train_val[, grepl("_PROB", names(train_val))],
                                       ytrain[-train_indices, svc_index],
                                       method = mod_ensemble_type, trControl = train_control)
```

```
    # Predict on test data
    probs_ensemble[, svc_index] <- predict(object = mod_ensemble[[svc_index]],
                                           newdata = test,
                                           type = "prob")$yes
  }
  write_submission(probs_ensemble, paste0(mod_ensemble_type, "_cutoff", prop_missing_cutoff))
  save(mod_ensemble, file = file.path(results_dir,
                                      paste0("mod_ensemble_", mod_ensemble_type, "_cutoff",
                                             prop_missing_cutoff, ".rda")))
  if (get_notifications)
    pbPost(type = "note",
           title = "stat222",
           body = paste0(mod_ensemble_type, " done!"),
           recipients = c(1, 2))
}

# Set predicted prob to 0 for services d and n in survey release b
if (FALSE) {
prop_missing_cutoff <- 0.9 # does not matter which one is used here
load(file = file.path(data_dir, paste0("data_cutoff", prop_missing_cutoff, ".rda")))
test <- data$test
# submit_files <- list.files("submit")
# for (file in submit_files) {
  file <- "gbm_cutoff0.9.csv"
  preds <- read.csv(file.path("submit", file))
  pdf(file.path(fig_dir, "preds-svcsdn.pdf"), width = 10, 4)
  par(mar = c(4.5, 4.5, 1, 1), mfrow = c(1, 2))
  preds_svc_d <- preds$service_d[test$release == "b"]
  hist(preds_svc_d, freq = FALSE, breaks = 50,
       main = "", col = "lightgrey",
       xlab = "Predicted probability for service d")
  legend("topright", legend = c(paste0("Mean = ", round(mean(preds_svc_d), digits = 3),
                                       "\nMedian = ", round(median(preds_svc_d), digits = 3),
                                       "\nSD = ", round(sd(preds_svc_d), digits = 3))),
       bty = "n")
  preds_svc_n <- preds$service_n[test$release == "b"]
  hist(preds_svc_n, freq = FALSE, breaks = 50,
       main = "", col = "lightgrey",
       xlab = "Predicted probability for service n")
  legend("topright", legend = c(paste0("Mean = ", round(mean(preds_svc_n), digits = 3),
                                       "\nMedian = ", round(median(preds_svc_n), digits = 3),
                                       "\nSD = ", round(sd(preds_svc_n), digits = 3))),
         bty = "n")
  dev.off()
  preds$service_d[test$release == "b"] <- 0
  preds$service_n[test$release == "b"] <- 0
  write.csv(preds, file.path("submit", gsub("\\.csv", "_releaseb_svcsdn0.csv", file)),
            row.names = FALSE)
# }
}

# Ridge Regression (Sample with 50% cutoff)
# load("data/data_dummy_0.5.rda")
```

```
train <- data_dummy_0.5$train
test <- data_dummy_0.5$test
ytrain <- data_dummy_0.5$ytrain
train <- as.matrix(train)
class(train) <- "numeric"
test <- as.matrix(test)
class(test) <- "numeric"
ytrain <- as.matrix(ytrain)
ytrain[ytrain=="yes"] <- 1
ytrain[ytrain=="no"] <- 0
class(ytrain) <- "numeric"
# Convert train, test and ytrain to numeric matrices, to accomodate glmnet.

library(glmnet)

# Use parallel to speed up
nCores <- parallel::detectCores()
doParallel::registerDoParallel(nCores)

# Keep track of the lambdas that give the smallest MSE
lambdas.min <- rep(0,14)
result <- matrix(rep(0,nrow(test)*ncol(ytrain)), ncol=ncol(ytrain))

for (i in 1:14){
  print(i)
  cv_mod_ridge <- cv.glmnet(train, ytrain[,i], alpha = 0, parallel=TRUE)
  #Use 10-fold cross validation on 100 default lambda values.
  lambdas.min[i] <- cv_mod_ridge$lambda.min
  pred_ridge <- predict(cv_mod_ridge, s = cv_mod_ridge$lambda.min, newx = test)
  result[,i] <- pred_ridge
}

# Visualize the lambdas
plot(x=1:14, y=lambdas.min, xlab="Column# in Y", ylab="Lambda.min", type="h",
     main="Lambdas with Smallest MSE Using Ridge_CV")

# Deal with values outside of 0 and 1:
colmean = colMeans(ytrain)
for (i in 1:14){
  result[which(result[,i] < 0),i] <- colmean[i]
  result[which(result[,i] > 1),i] <- colmean[i]
}

# Write Submission
write_submission(result, "Ridge0.5")

# Benchmark models
# Random probability drawn from U(0, 1)
probs <- matrix(runif(nrow(test)*(ncol(ytrain))), nrow(test), ncol(ytrain))
write_submission(probs, paste0("unifseed", seed))
# Constant probability of 0.5
probs <- matrix(0.5, nrow(test), ncol(ytrain))
write_submission(probs, "constant0.5")
```

```r
# Constant probability of proportion for each service
ytrain_props <- sapply(ytrain, mean)
probs <- matrix(rep(ytrain_props, each = nrow(test)), nrow(test), ncol(ytrain))
write_submission(probs, "constantprop")
```