Supervised Learning Classification

Statistical Learning on Self-Assessed Health Status

Preprocessing

Importing training and testing dataset

Removing the variables with more than 80% missing values

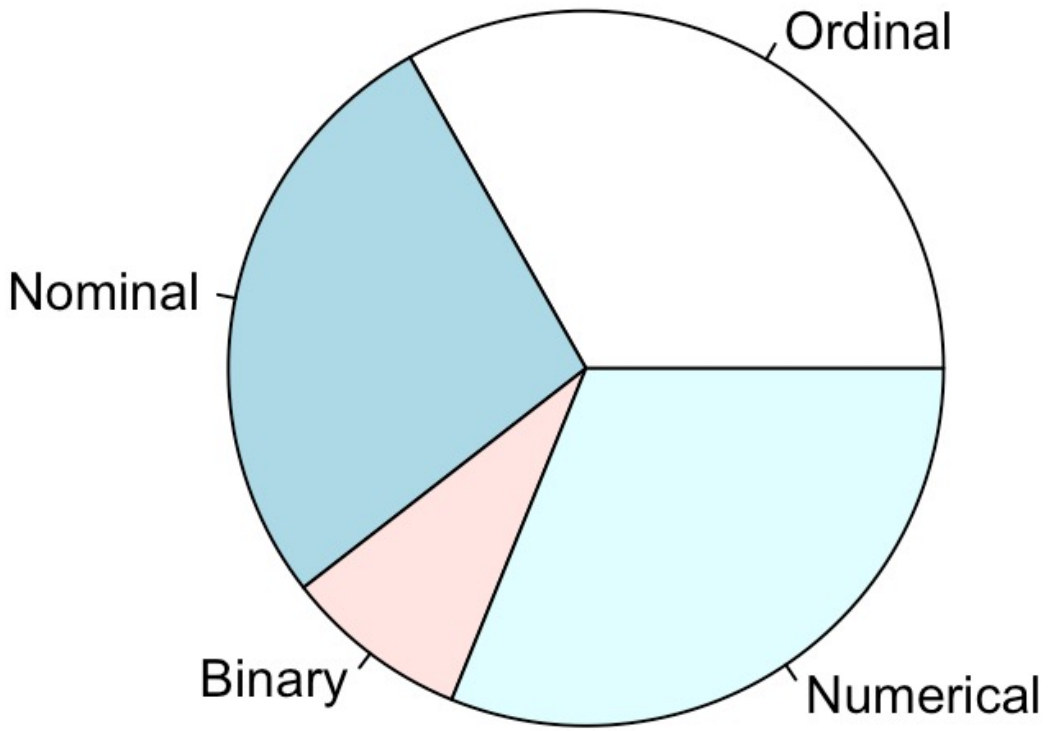
Removing Person Id, Interviewer Id, and Unique Id

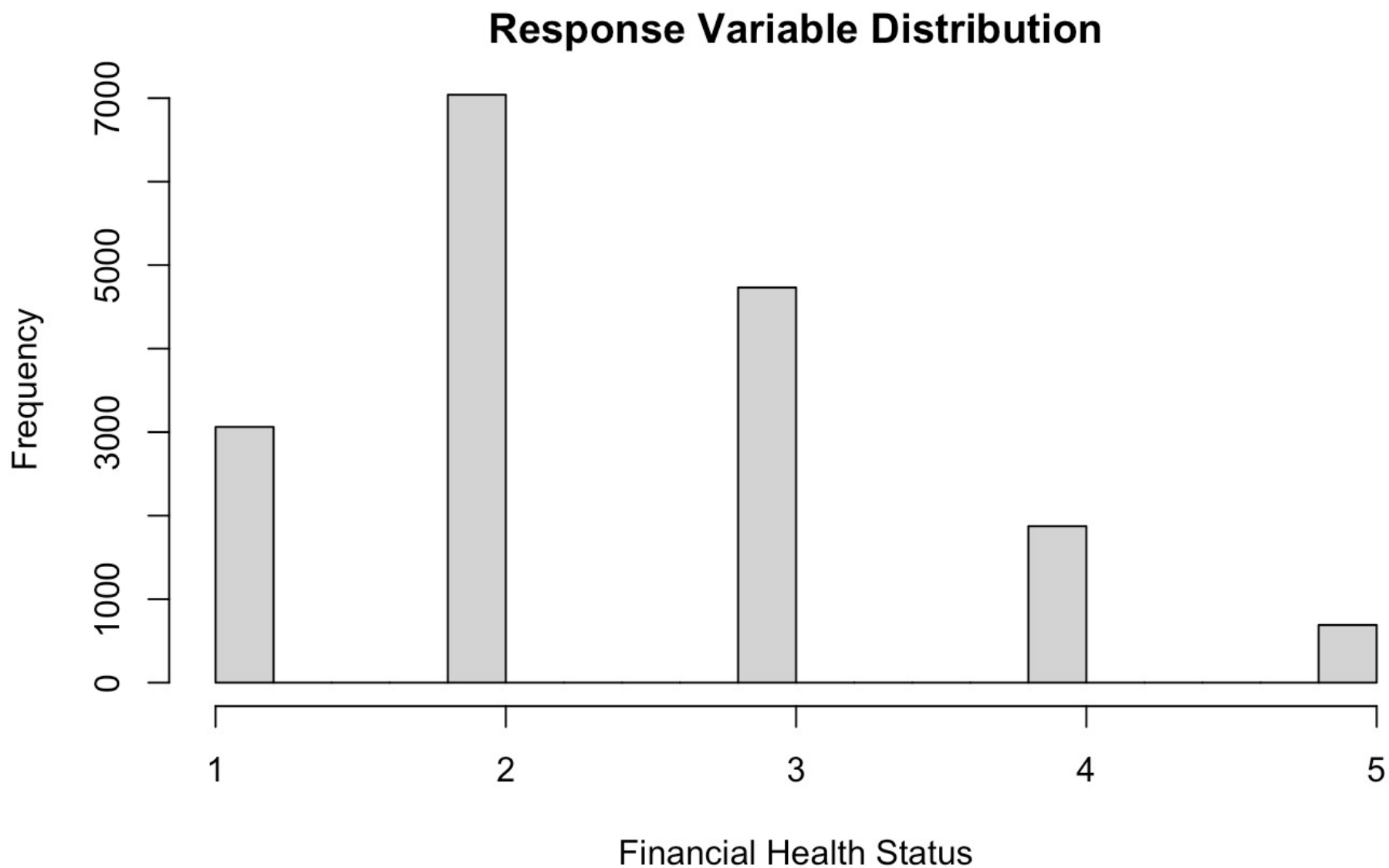
Highlights From Exploratory Data Analysis

We manual divide all the variables into four subsets

Numerical <chr>	Binary <chr>	Ordinal <chr>	Nominal <chr>
year	x15	x1	x7
personid	x18	x2	x8
x22	x19	x3	x9
x23	x73	x4	x10
x24	x175	x5	x11
x25	x227	x6	x12

Type of Variables in the dataset



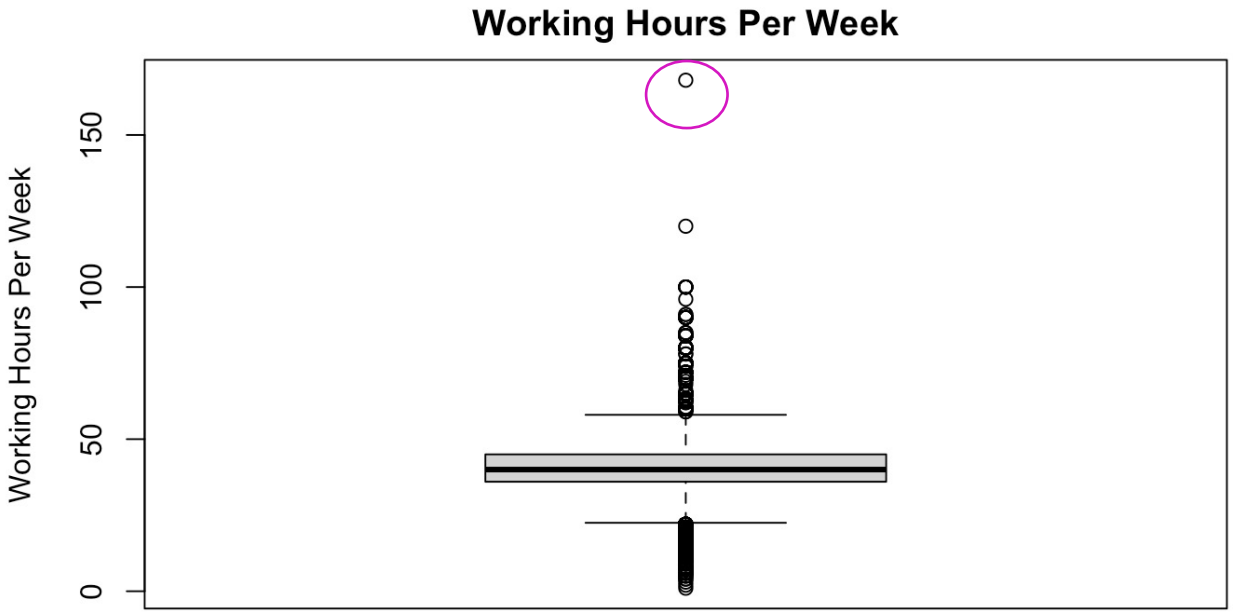

Response Variable Distribution
Frequency
0
1000
3000
5000
7000
1
2
3
4
5
Financial Health Status

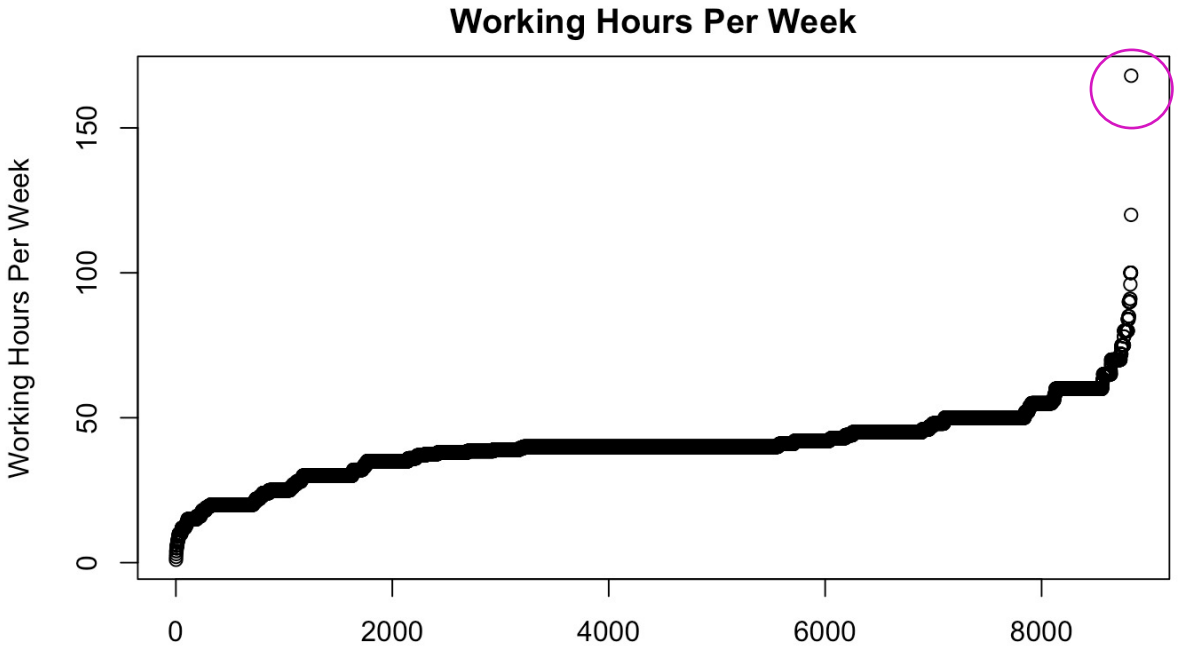
Unusual Circumstances

During data analysis, we used quantile plot and found two variables with unusual values

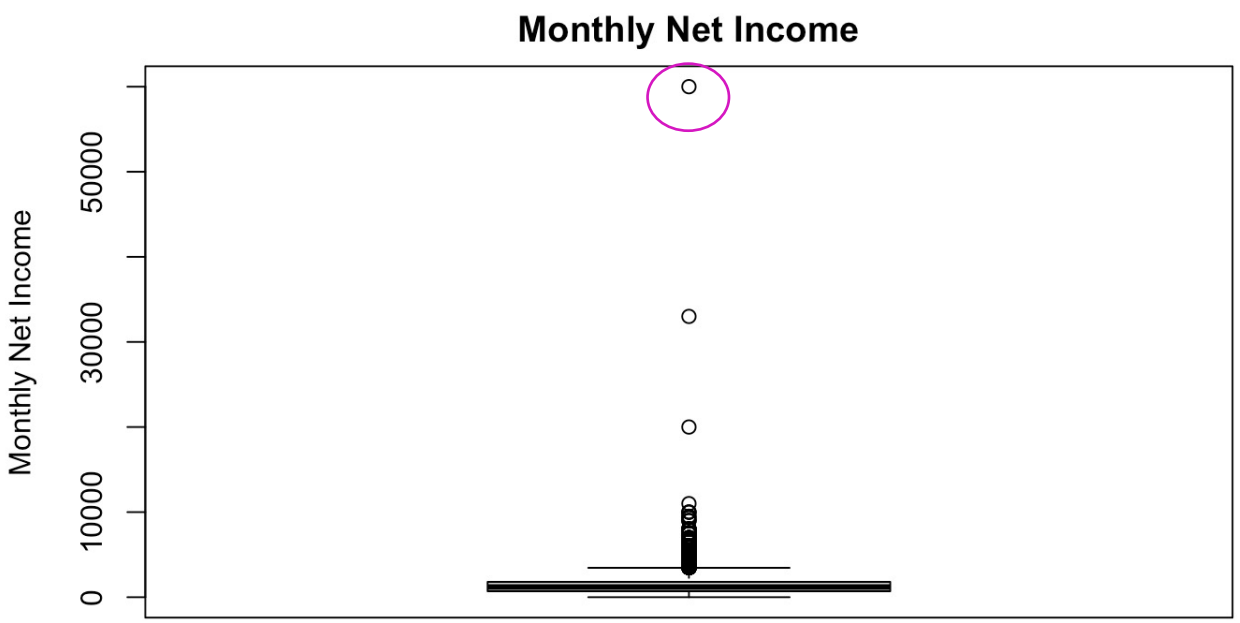
```
plot(sort(train[,"x715"]), ylab = "Working Hours Per Week",
     main = "Working Hours Per Week")
plot(sort(train[,"x723"]), ylab = "Monthly Net Income",
     main = "Monthly Net Income")
```

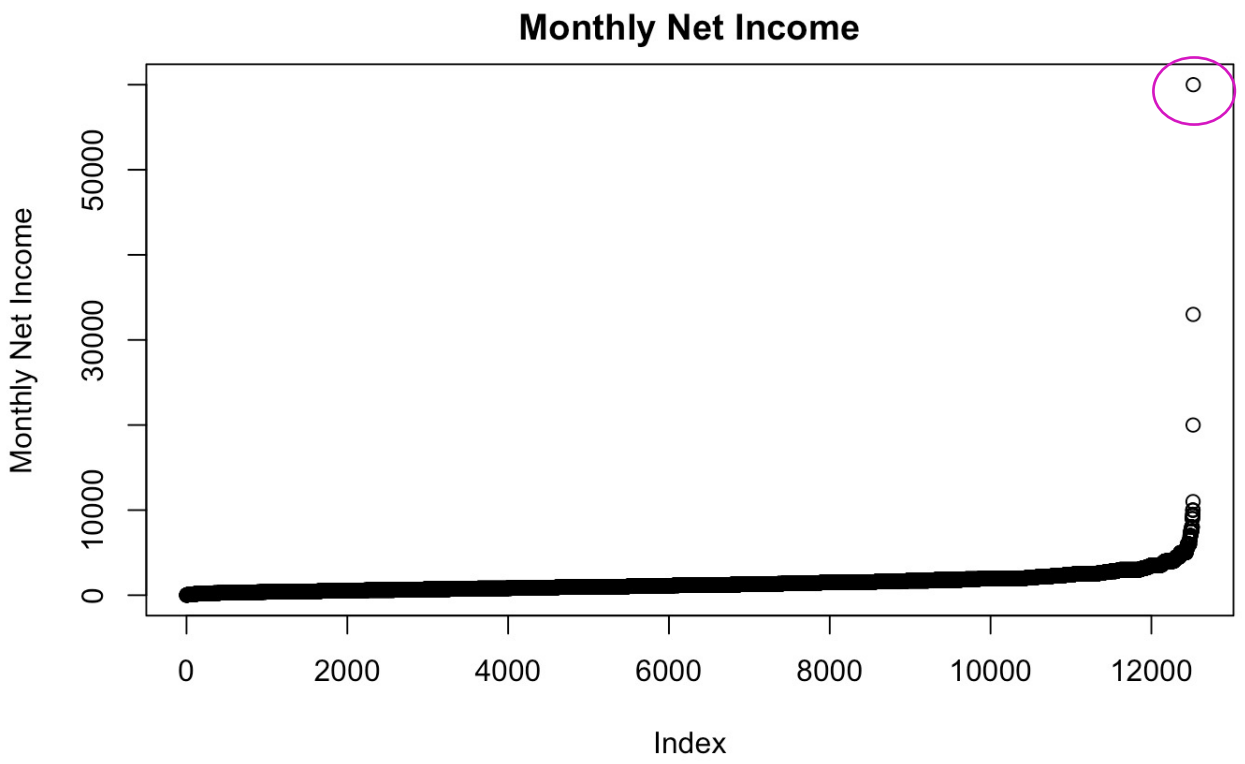
X715: Working Hours Per Week





X723: Monthly Net Income





Handling Categorical Variables

Before converting all nominal and ordinal variables into factors, we merge the training and testing dataset to ensure they are encoded with the same levels

```
# Making sure the testing dataset has the same columns as the training dataset
test <-test[,(colnames(test) %in% colnames(train))]
# Ensuring both datasets has same dimension for merging
fake_health = rep(NA,length(test[,1]))
test$health = fake_health
# Merging
total = rbind(test,train)
```

Factoring Ordinal and Nominal variables

- We used ”factor” function in R to assign levels for categorical variables.
- For ordinal variable, using ”order = TRUE” to ensure the encoding keeps the difference in order.

```
for (names in order){
  if (names %in% colnames(total)){
total[,names] = factor(total[,names],order=TRUE)
}
}
```

Imputing Missing Values

- For categorical variables, we imputed the missing values using mode.
- For numerical variables, we imputed the missing values using median.

Using ”na.roughfix” function in “randomForest” library

```
library(randomForest)
train_imputed = na.roughfix(train_factorized)
test_imputed = na.roughfix(test_factorized)
```

Feature Engineering

- During analysis, based on our domain knowledge, we derived a new x-variable: **the average living space in m^2 per person in the household.**
- This variable is derived by the ratio of living space in m^2 "x1134" and number of person in the household "x963".

```
train_imputed$average_space =
  train_imputed[,"x1134"]/train_imputed[,"x963"]
test_imputed$average_space =
  test_imputed[,"x1134"]/test_imputed[,"x963"]
```

Modeling

RANDOM FOREST

BOOSTING

NEURAL NETWORK

Training vs. Validation Set

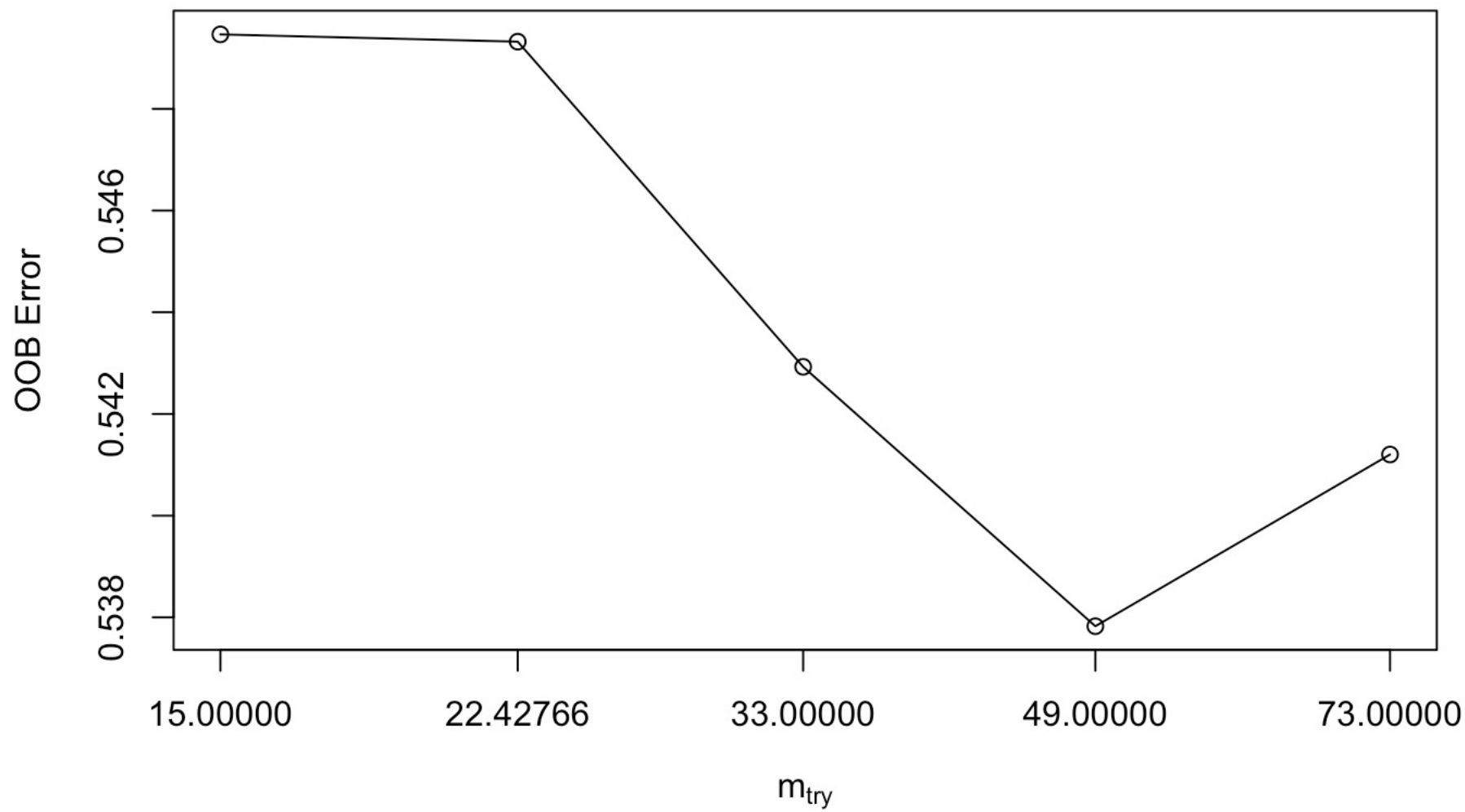
- Splitting the training set into training data and validation data
- 80% Training data for model training
- 20% Validation data to evaluate the performances of models and prepare for stacking
- The “createDataPartition” function in “caret” library ensures the response variable of validation data and training data have the similar distribution

Random Forest

Tuning Number of Variables to Choose

- Removed nominal variables with more than 53 categories.
  - There will be more than $2^{53} - 2$ choices for the Random Forest algorithm in corresponding splits (Infeasible Computational Cost)
- Tuning the number of variables to choose from at each split (mtry)
  - Lower number will make resulted tree unreliable.
  - Higher number will cause overfitting

```
tuneRF(x_var,y_var, mtryStart = sqrt(num_of_col),stepFactor = 1.5,
       improve = 1e-5, ntree = 100)
```


OOB Error
0.538
0.542
0.546
15.00000
22.42766
33.00000
49.00000
73.00000
mtry

Tuning Total Number of Tree

- ”Do.trace = TRUE” in random forest function allows us to check Out of Bag Error of each number of trees during training.
  - Lower “ntree” might result in not all variables being selected in algorithm
  - Higher “ntree” has high computational cost

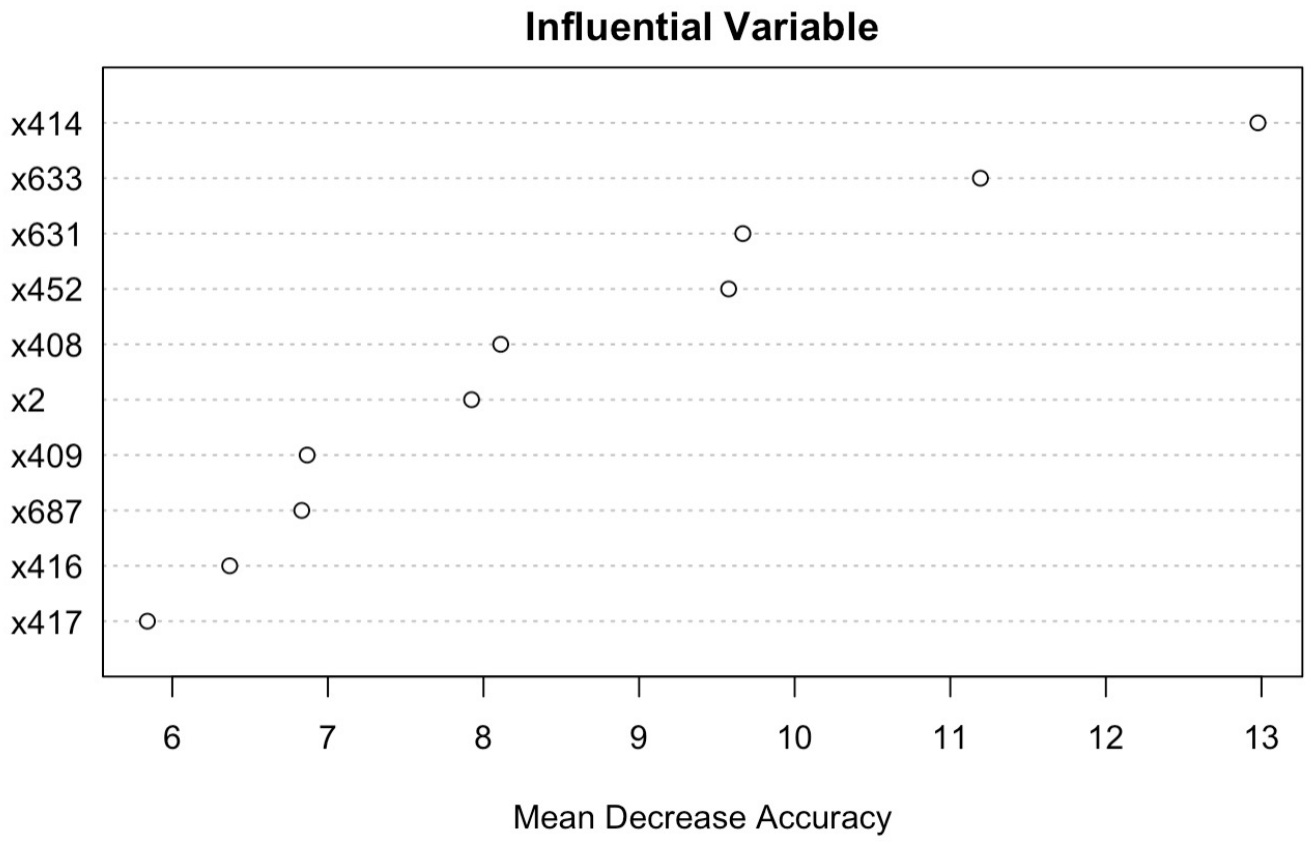
ntree	OOB	1	2	3	4	5
1:	66.85%	73.15%	58.89%	66.28%	79.67%	88.50%
2:	66.51%	70.92%	57.33%	67.05%	83.62%	88.38%
3:	66.26%	70.29%	56.30%	68.49%	82.59%	89.25%
4:	65.79%	70.36%	54.90%	68.68%	82.12%	90.75%
5:	65.32%	69.65%	54.11%	68.39%	83.80%	88.52%

Random Forest Training

- Using “mtry = 49” from pervious tuning and 1000 total number of trees , applying Random Forest algorithm to the training data.

```
rf_model<- randomForest(health~.,train_rf,mtry = 49, ntree = 1000, importance=TRUE,do.trace=TRUE)
```

- And using resulted model to predict the probability of categories for validation set



x414 LAST 4 WEEKS: HOW OFTEN PHYSICAL PAIN?

Influential Variables

- We also calculate the mean of decrease in accuracy when each variables are removed for the included tree
- One of the interest insight of data is the most influential variables is the “How often physical pain in last 4 weeks"

Model Performance

- One of the advantage of Random Forest is using OOB error to examine the performance for different parameters.
- However, since we split a validation set for stacking, we can also use it to calculate cross entropy for evaluation. (1.1823)

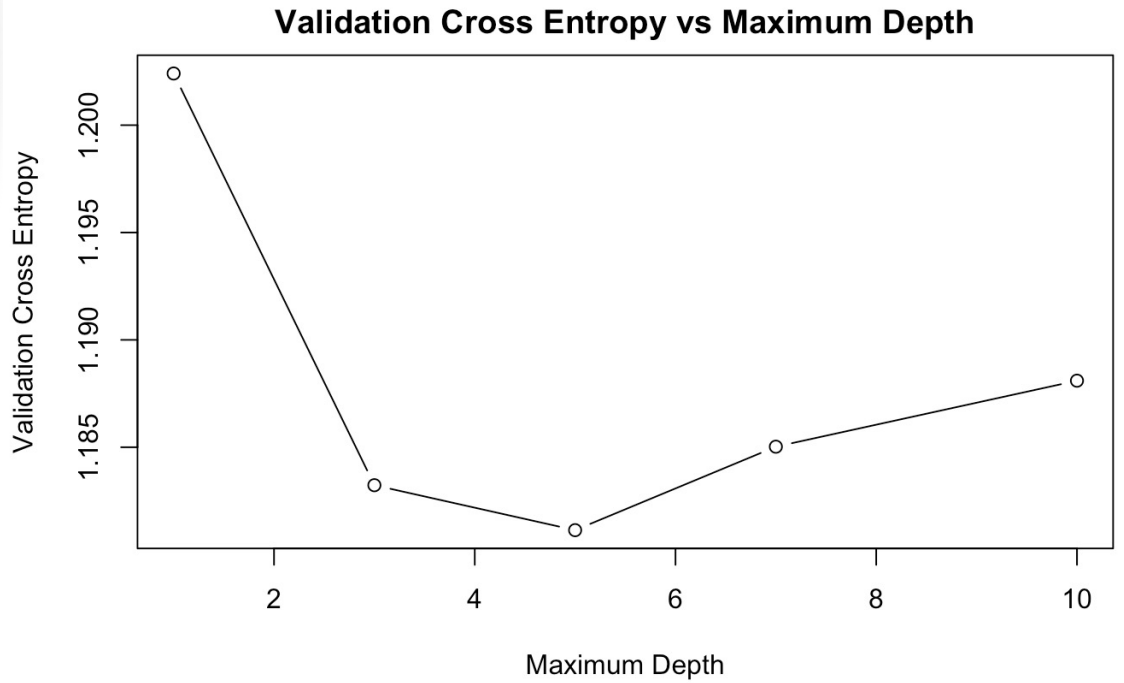
```
for (i in c(1:length(validation$health))){
result = c(result,log(pred_rf[i,as.integer(validation$health[i])]))
}
-mean(result)
```

Gradient Boosting

Tuning Maximum Depth

```
depth = c(1,3,5,7,10)
for (i in depth){
model_gbm <- gbm(health~., data = train_gbm, distribution = "multinomial",
                 n.trees = 200, shrinkage = 0.03, interaction.depth = i)
pred = predict(model_gbm,validation_gbm[,-c(length(validation_gbm)-1)],
               type = "response")
for (i in c(1:length(validation_gbm$health))){
result = c(result,log(pred[i,as.integer(validation_gbm$health[i])]))
}
error = c(error,-mean(result))
}
```

- Removed nominal variables with more than 53 categories.
- We tuned the maximum depth of each tree. The maximum depth of tree is selected to be 5.



Tuning Number of Iterations

Iter	TrainDeviance	ValidDeviance	StepSize	Improve
1	1.6094	nan	0.0300	0.0411
2	1.5847	nan	0.0300	0.0362
3	1.5621	nan	0.0300	0.0344
4	1.5410	nan	0.0300	0.0324
5	1.5213	nan	0.0300	0.0297
6	1.5033	nan	0.0300	0.0274
7	1.4865	nan	0.0300	0.0251
8	1.4705	nan	0.0300	0.0228
9	1.4561	nan	0.0300	0.0210
10	1.4426	nan	0.0300	0.0197
20	1.3397	nan	0.0300	0.0103
40	1.2360	nan	0.0300	0.0031
60	1.1823	nan	0.0300	0.0008
80	1.1465	nan	0.0300	0.0005
100	1.1185	nan	0.0300	0.0001
120	1.0944	nan	0.0300	0.0003
140	1.0738	nan	0.0300	-0.0000

- We can see that when number of iterations is above 140, improvement become negative.
- Therefore, we choose number of iterations to be 140.

Gradient Boosting Training

- Using Maximum Depth = 3 from previous tuning, 200 iterations, and learning rate of 0.03, applying gradient boosting algorithm to train the data.

```
model_gbm <- gbm(health~., data = train_gbm, distribution = "multinomial",
                 n.trees = 140 ,shrinkage = 0.03,interaction.depth = 5,verbose = TRUE)
```

- And using resulted model to predict the probability of categories for validation set

Model Performance

- The validation cross-entropy is calculated to be 1.1903.

```
pred_gbm = predict(model_gbm,validation_gbm,type = "response")
# Calculating Cross Entropy
for (i in c(1:length(validation_imputed$health))){
result = c(result,log(pred_gbm[i,as.integer(validation_imputed$health[i])]))
}
-mean(result)
```

Neural Network

Indicator variables

- We used "fastDummies" package to create indicator variables for all categorical variables.

```
library(fastDummies)
dummy_cols(train,select_columns = c(order,non_order,yes_no))
dummy_cols(validation,select_columns = c(order,non_order,yes_no))
dummy_cols(test,select_columns = c(order,non_order,yes_no)
```

Standarization

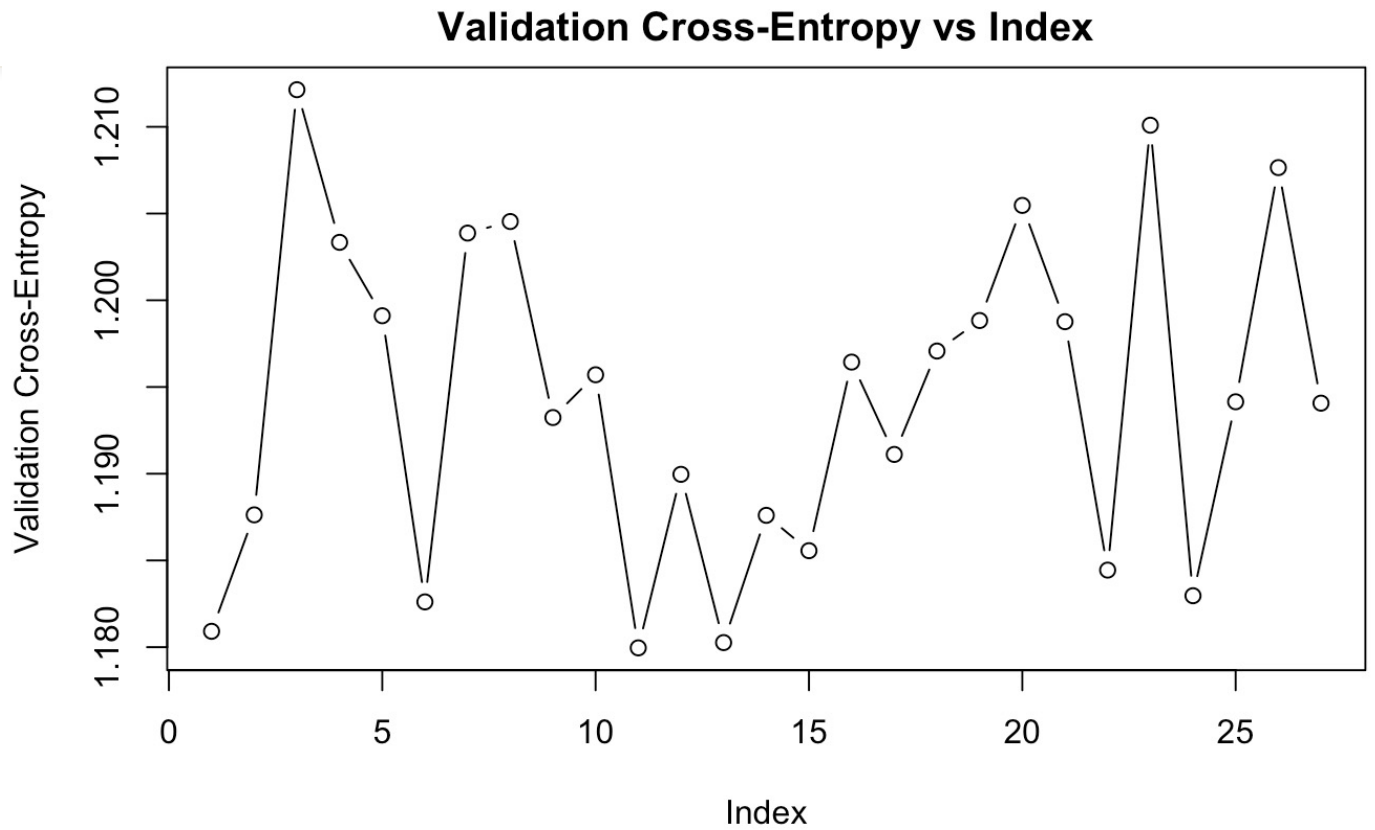
```
for (names in numerical){
  if (names %in% colnames(total)){
total[,names] = scale(total[,names]) } }
```

We standardized all numerical variables in the dataset using "scale" function to ensure that their expected values equal to 0 and variance equal to 1.

Tuning

- We tuned three parameters in total – Drop-out rate, number of neurals in the first layer, and number of neurals in the second layer.
- For Drop-out rate, we tried 0.1, 0.2, and 0.3.
- For number of neurals in each layer, we tried 64, 128, and 256.
- There are 27 combinations in total.

	Index	Drop Out Rate	First Layer Neural	Second Layer Neural	Error
1	1	0.1	64	64	1.180914
2	2	0.1	64	128	1.187628
3	3	0.1	64	256	1.212140
4	4	0.1	128	64	1.203340
5	5	0.1	128	128	1.199107
6	6	0.1	128	256	1.182610
7	7	0.1	256	64	1.203874
8	8	0.1	256	128	1.204540
9	9	0.1	256	256	1.193238
10	10	0.2	64	64	1.195705
11	11	0.2	64	128	1.179959
12	12	0.2	64	256	1.189964
13	13	0.2	128	64	1.180264
14	14	0.2	128	128	1.187601
15	15	0.2	128	256	1.185562
16	16	0.2	256	64	1.196436
17	17	0.2	256	128	1.191112
18	18	0.2	256	256	1.197074
19	19	0.3	64	64	1.198832
20	20	0.3	64	128	1.205469
21	21	0.3	64	256	1.198766
22	22	0.3	128	64	1.184443
23	23	0.3	128	128	1.210093
24	24	0.3	128	256	1.182966
25	25	0.3	256	64	1.194148
26	26	0.3	256	128	1.207649
27	27	0.3	256	256	1.194068



Neural Network Training

- We included two layers in our neural network model.
- Drop-out Rate: 0.2, Number of Neurals in First Layer: 64, and Number of Neurals in Second Layer: 128 as previously tuned.
- Activation Function “ReLU” provides nonlinearity to the model

```
model <-keras_model_sequential()%>%
  layer_dense(units = 64, activation = "relu")%>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 128, activation = "relu")%>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 5, activation = "softmax")%>%
  compile(optimizer = optimizer_adam(lr = 0.001), loss = "categorical_crossentropy")
```

Performance

- Neural Network has the lowest computational cost among the three algorithms that we tried.
- The Validation Cross-Entropy is calculated to be 1.1799.

```
for (i in c(1:length(validation_imputed$health))){
result = c(result,log(pred_nerual[i,as.integer(validation_imputed$health[i])]))
}
-mean(result)
```

Stacking

Stacking

- Combing the validation prediction results (probabilities) from above three models (Random Forest, Gradient Boosting, and Neural Network) as the new training set

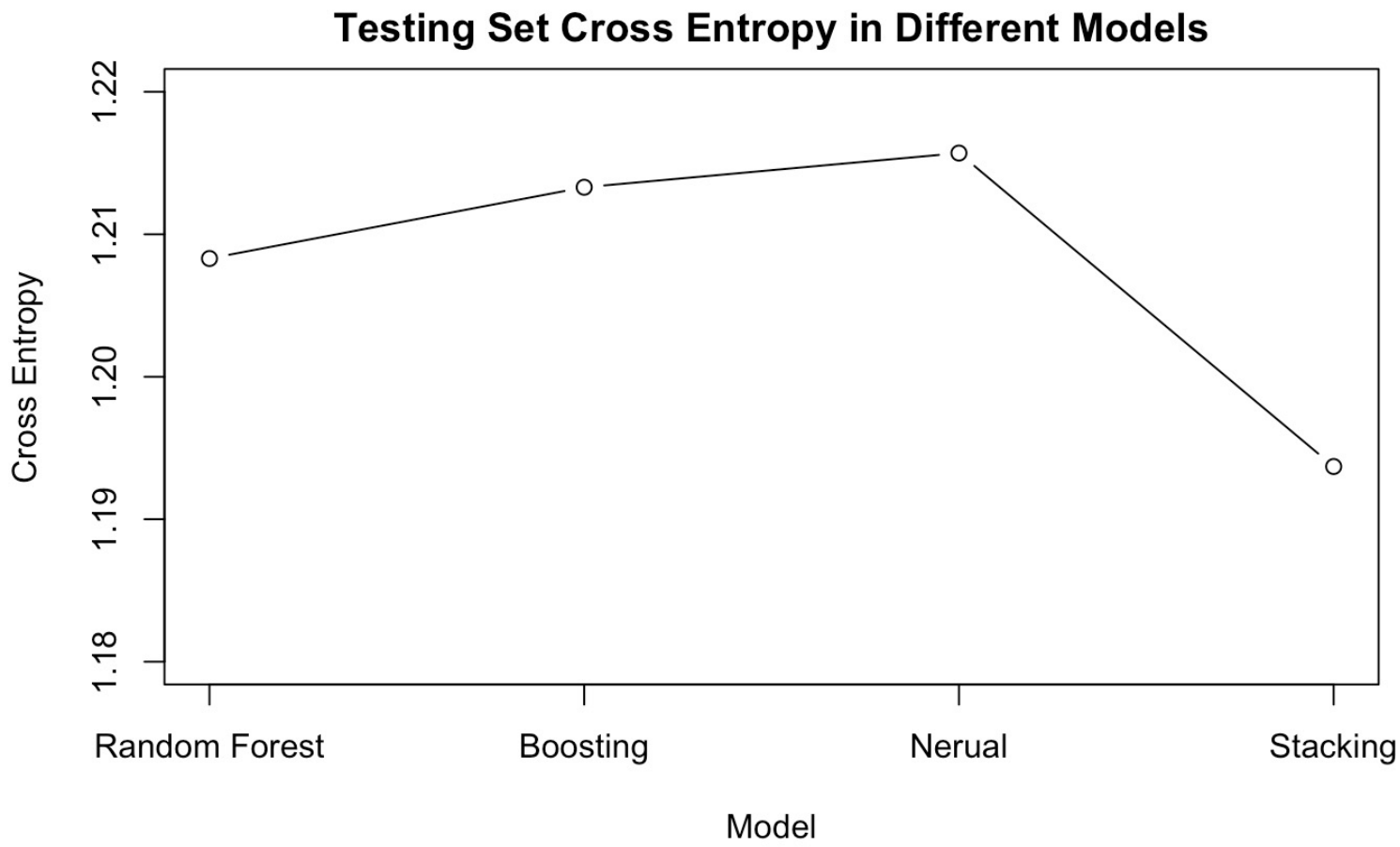
rf1	rf2	rf3	rf4	rf5	gbm1	gbm2	gbm3	gbm4	gbm5	n1	n2	n3	n4	n5	health
1.0e-13	6.0e-02	0.26	3.0e-01	3.8e-01	0.004642419	0.012354671	0.06486385	0.22243137	0.695707689	0.0005817707	0.022460317	0.13157688	0.3069919646	5.383891e-01	4
1.4e-01	7.6e-01	0.08	2.0e-02	1.0e-13	0.106506107	0.675526845	0.15603023	0.04661126	0.015325551	0.1780477613	0.651157618	0.14819561	0.0193621647	3.236833e-03	1
2.0e-02	3.2e-01	0.30	2.2e-01	1.4e-01	0.060776839	0.195243720	0.22486999	0.34036974	0.178739714	0.0078398101	0.214521974	0.56042171	0.1614444703	5.577207e-02	4
2.0e-02	2.6e-01	0.50	1.8e-01	4.0e-02	0.038662488	0.137017418	0.50236230	0.28890860	0.033049202	0.0221618023	0.310081005	0.49378464	0.1407002658	3.327233e-02	4
8.0e-02	3.8e-01	0.34	1.4e-01	6.0e-02	0.060110278	0.349950714	0.50037618	0.07068641	0.018876421	0.0599954650	0.304106146	0.34669903	0.1667679548	1.224314e-01	3
6.0e-02	2.0e-01	0.40	2.6e-01	8.0e-02	0.045280061	0.218013455	0.30572115	0.26315129	0.167834040	0.0194278751	0.179582030	0.26904681	0.2725459337	2.593973e-01	5

- Training a Multinomial logistic regression model with this new training set

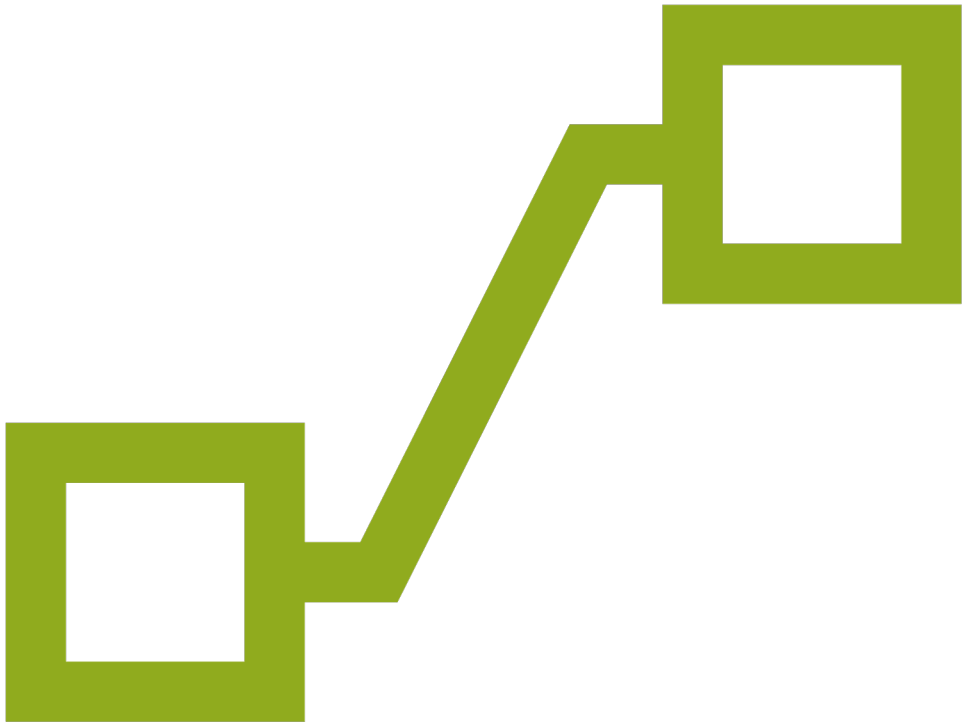
Testing Set Prediction

- Re-training the three models with same parameters using entire training dataset (training and validation data)
- Estimating the probabilities for testing dataset using each newly trained models
- Combining the estimated probabilities from each models to create a new testing set
- Estimating the probabilities of this new testing set using logistics regression

Stacking Improve the Performance



Conclusion



Performance on Different Models

Insight: validation sets might not represent the performance of testing sets