HINTS Analysis on EMR Patient Participation

IDS.506 - Health Info. Mgmt. & Analytics

Robert Duc Bui - mbui7@uic.edu - 660809303

Environment Setup

Our feature engineering and modeling process will primarily be based on the `tidyverse` ecosystem and its machine learning-focused sister package system `tidymodels`.

Data importing would be carried out using the `haven` package over the conventional `foreign` package thanks to `haven`'s better handling of named columns, of which the HINTS dataset has many.

For variables that we suspect share a common associative factor, we will be performing Principal Component Analysis by topic group. While we can use R's built-in `stats` to perform PCA on numeric or ordinal values, binary values require a specific Logistic PCA method, provided by the `logisticPCA` package. We will also be calling `factoextra` and `ggfortify` to perform cross-validated hyperparameter tuning for PCA.

Our modeling will be done using the `tidymodels` syntax, which is set to replace the outgoing `caret` convention. Specific backends such as `glmnet` and `kernlab` does not need to be explicitly called, but must be installed in order for the `tidymodels` workflow to execute.

Assessing the model will be done using the `DALEX` and `DALEXtra` family of packages, which allows us to visualise variable importance. This is ultimately the goal of this project, as the research question hinges upon finding out which factors influence patients' participation in EMR systems.

Miscellaneous beautification and presentation will be done using `patchwork`, `knitr`, and `kableExtra`. This notebook was created in R 4.1.2, RStudio version 2021.09.2, running on x64 Windows 11. Knitting to PDF is performed using `TinyTeX`, a lightweight distribution of `LaTeX` built for RStudio.

```
# R Classics
library(tidyverse)
library(tidymodels)

# SPSS data file importer
library(haven)

# Logistic PCA for binary variables + tools for examining PCA
library(logisticPCA)
library(factoextra)
library(ggfortify)

# Interface with tidymodels objects to derive variable importance
require(glmnet)
require(kernlab)
library(DALEX)
library(DALEXtra)

# Misc appearance
```

```
library(patchwork)
library(knitr)
library(kableExtra)

# setwd("G:/My Drive/academics/2122_uic_ms/2022_spring/ids506_health/HINTS")
```

Import data & Extract Variables

Data is imported from `hints5_cycle2_public.sav`, translated from SPSS .sav file into R data.frame with `haven`. We only select a subset of available variables, grouped by comments into certain topic clusters.

However, before any other processing steps, note that immediately importing we filter to keep only the data from patients that have actually been *offered* access to an EMR - as patients who have not been offered EMR access cannot participate no matter how willing they are to utilise EMRs.

```
df <- read_sav("hints5_cycle2_public.sav") %>%
  # Filtering for patients that are actually offered EMR access.
  filter(EverOfferedAccessRec == 1) %>%
  select(
    # Y VAR
    AccessOnlineRecord,
    # Technology Use
    UseInternet,
    Internet_Broadbnd,
    Internet_DialUp,
    Internet_Cell,
    Internet_WiFi,
    TabletHealthWellnessApps,
    OtherDevTrackHealth,
    IntRsn_VisitedSocNet,
    IntRsn_SharedSocNet,
    IntRsn_WroteBlog,
    IntRsn_SupportGroup,
    IntRsn_YouTube,
    WhereSeekHealthInfo,
    # Health-mindedness & Lifestyle
    SeekHealthInfo,
    MostRecentCheckup2,
    FreqGoProvider,
    GeneralHealth,
    HowLongModerateExerciseMinutes,
    TimesStrengthTraining,
    AverageTimeSitting,
    # Perceived Need/Availability of EMR
    ProviderMaintainEMR2,
    WhoOffered_HCP,
    WhoOffered_Insurer,
    ESent_AnotherHCP,
    ProbCare_BringTest,
    ProbCare_RedoTest,
    ProbCare_ProvideHist,
    # Security Concerns
```

```
    ConfidentInfoSafe,
    WithheldInfoPrivacy,
    # Demographics
    AgeGrpB,
    OccupationStatus,
    ActiveDutyArmedForces,
    MaritalStatus,
    EducA,
    BornInUSA,
    SpeakEnglish,
    RaceEthn5,
    SexualOrientation,
    SelfGender,
    RentOrOwn,
    IncomeRanges,
    MailHHAdults
  ) %>%
  sapply(as.numeric) %>%
  as.data.frame() %>%
  rownames_to_column("new_id")
```

Processing dependent target variable

Our dependent target variable for the models is here defined as `AccessOnlineRecord` in the dataset. This is defined as "How many times did you access your online medical record in the last 12 months?". Preprocessing includes converting all values equal to or greater than once to a 1 (or TRUE) factor variable, and all others to 0 (or FALSE) factor. Fundamentally this is a conscious decision to limit the scope of our analysis: we are only looking at whether a patient participate, or doesn't participate, in an EMR. We encode this as `y`, and remove the original, more granular vector.

```
df <- df %>%
  mutate(
    y = case_when(
      AccessOnlineRecord <= 0 ~ 0,
      AccessOnlineRecord >= 1 ~ 1
    ) %>% factor()
  ) %>%
  select(new_id,y,everything()) %>%
  select(-AccessOnlineRecord)
```

Feature Engineering:

Feature Engineering 1: Demographics

For demographics, we will be performing the following transformations:

- AgeGrpB: This is one of the few values that are pre-stratified by HINTS researchers into 18-34, 35-49, 50-64, 65-74 age groups. Missing values are imputed to be the US median/mean age of 38, falling into bucket 2.

- OccupationStatus: All missing data is coerced into the category for "Unemployed". The rest are kept as-is. This is also converted into a multiclass factor data type, representing different classes.
- ActiveDutyArmedForces: All personnel that is active or inactive military and National Guard are assigned to 1. Missing data and non-military are assigned to 0, effectively converting this to a dummy variable for military status.
- MaritalStatus: Missing data and single-never-married respondents are assigned to 0. Married and cohabiting as married are assigned to 1, divorced and separated is 2, and widowers are assigned to 3. This is subsequently converted into factor data type, representing different classes.
- EducA: Missing data, less than high school, and GED patients are assigned to 0. Some college is assigned to 2, and college graduate or more is assigned to 3. This is also converted into a multiclass factor data type, representing different classes.
- BornInUSA: Missing values and non-US-born patients are assigned to 0, and US-born values are assigned to 1. This effectively converts this into a dummy variable.
- SpeakEnglish: Missing data, bad responses, and "Not at all" speakers are assigned to 0. "Not well" is assigned to 1, "Well" is assigned to 2, and "Very Well" is assigned to 3. This effectively turns into an ascending grade for each increasing level of fluency.
- RaceEthn5: Missing data and "other" responses are assigned to "other". Specific ethnicities & nationalities are sorted into 4 buckets: "White", "Black", "Hispa", and "Asian". This and other multiclass factor variables will be converted into dummy variables by `tidymodels`.
- SexualOrientation: Straight patients are classified as "S" code, and all others are assigned to "NS" code. These will be converted into dummy variables by `tidymodels`.
- SelfGender: NAs are kept as "N", the rest are coded as M/F as a multiclass variable.
- RentOrOwn: Homeowners are kept as "Own", and all other categories are coerced to "Rent".

Income is handled more specifically with a multiple-step process:

- First, the household size is derived from MailHHAdults. Values are sorted into values from 1 to 6, with 6 representing all households with more than 5.
- Then, using household sizes, IncomeRanges is benchmarked against the 2018 Federal Poverty Line (set by the Department of Health and Human Services). The output of this transformation is encoded as a boolean {1,0} named IsPoverty - denoting whether a patient falls under the poverty line or not.

```
d.demo <- df %>%
  select(
    new_id,y,
    # Independent vars
    AgeGrpB,OccupationStatus,ActiveDutyArmedForces,MaritalStatus,
    EducA,BornInUSA,SpeakEnglish,RaceEthn5,SexualOrientation,
    SelfGender,RentOrOwn,IncomeRanges,MailHHAdults
  ) %>%
  mutate(
    # NAs: assigning US median age of 38, group 2 (US Census Bureau)
    AgeGrpB =                case_when(AgeGrpB == -9 ~ 2,
                                       T ~ AgeGrpB) %>% factor(),
    # NAs: assigning special 0 group for no information
    OccupationStatus =       case_when(OccupationStatus %in% c(-9,-5,91) ~ 0,
                                       T ~ OccupationStatus) %>% factor(),
```

```
  # NAs: missing data assigned to not military, active/past active & nat.guard
  # assigned to 1
  ActiveDutyArmedForces = case_when(ActiveDutyArmedForces %in% c(-9,5) ~ 0,
                                    ActiveDutyArmedForces %in% c(1,2,3,4) ~ 1),
  # NAs: missing + single assigned to 0, married is 1, divorced/separated is 2,
  # widowed is 3
  MaritalStatus         = case_when(MaritalStatus %in% c(-9,-5,6) ~ 0,
                                    MaritalStatus %in% c(1,2) ~ 1,
                                    MaritalStatus %in% c(3,5) ~ 2,
                                    MaritalStatus %in% c(4)   ~ 3) %>% factor(),
  # NAs: missing + less than GED + GED assigned to 1, college grad and above
  # to 3, college non-grad to 2
  EducA                 = case_when(EducA %in% c(-9,1,2) ~ 1,
                                    EducA == 3 ~ 2,
                                    EducA == 4 ~ 3) %>% factor(),
  # NAs: missing + non-US assigned to 0, US assigned to 1
  BornInUSA             = case_when(BornInUSA == 1 ~ 1,
                                    T ~ 0),
  # NAs: missing + non-English assigned to 0, steps reversed to ascending order
  SpeakEnglish          = case_when(SpeakEnglish %in% c(-9,-5,4) ~ 0,
                                    SpeakEnglish == 3 ~ 1,
                                    SpeakEnglish == 2 ~ 2,
                                    SpeakEnglish == 1 ~ 3) %>% factor(),
  # NAs: missing assigned to 'Other', values converted to str/factor.
  RaceEthn5             = case_when(RaceEthn5 %in% c(-9,5) ~ "Other",
                                    RaceEthn5 == 1 ~ "White",
                                    RaceEthn5 == 2 ~ "Black",
                                    RaceEthn5 == 3 ~ "Hispa",
                                    RaceEthn5 == 4 ~ "Asian"),
  # NAs: straight assigned to "S", others assigned to "NS"
  SexualOrientation     = case_when(SexualOrientation == 1 ~ "S",
                                    T ~ "NS"),
  # NAs: coerced to N. Numerics coerced to M/F
  SelfGender            = case_when(SelfGender == -9 ~ "N",
                                    SelfGender == 1 ~ "M",
                                    SelfGender == 2 ~ "F"),
  # NAs: homeowners assigned to 'own', non-homeowners assigned to 'rent'
  RentOrOwn             = case_when(RentOrOwn == 1 ~ "Own",
                                    T ~ "Rent")
) %>%
mutate(
  # Special case: determining poverty status
  # First determine household size
  hh = case_when(MailHHAdults %in% c(-1,1) ~ 1,
                 MailHHAdults %in% c(-9,-4,-2,2) ~ 2,
                 MailHHAdults == 3 ~ 3,
                 MailHHAdults == 4 ~ 4,
                 MailHHAdults == 5 ~ 5,
                 MailHHAdults >= 6 ~ 6),
  IsPoverty = case_when(
    # Households with 2 or fewer people
    hh %in% c(1,2)   & IncomeRanges %in% c(-9,1,2) ~ 1,
    hh %in% c(1,2)   & IncomeRanges > 2 ~ 0,
```

```
      # 3
      hh == 3          & IncomeRanges < 4 ~ 1,
      hh == 3          & IncomeRanges >= 4 ~ 0,
      # 4 or more
      hh %in% c(4,5,6) & IncomeRanges < 5 ~ 1,
      hh %in% c(4,5,6) & IncomeRanges >= 5 ~ 0
    )
  ) %>%
  select(-c(IncomeRanges,MailHHAdults,hh))
```

Feature Engineering 2: Technology Use

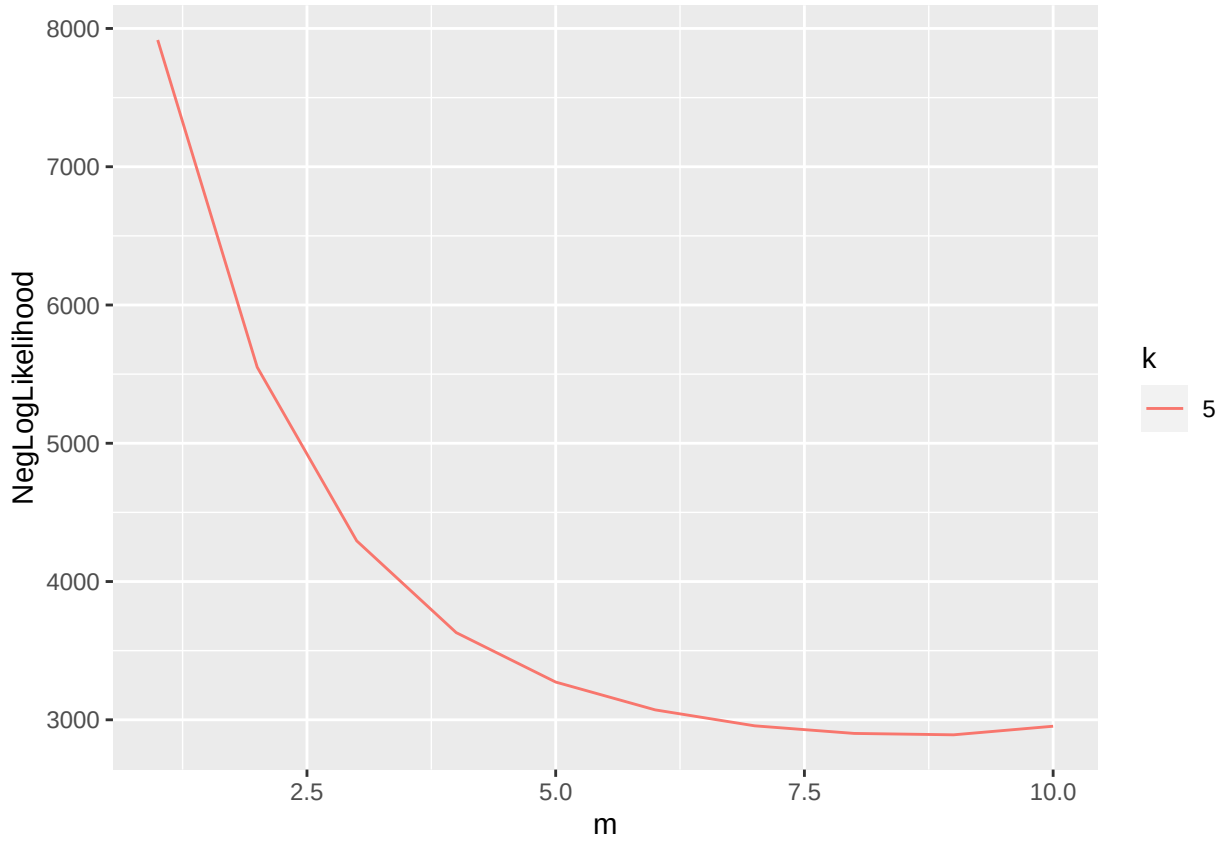
All values in technology use are encoded as simple binary variables, with missing values assumed to be 0/FALSE. With these variables, we can perform logistic PCA in the next subsection.

```
d.tech <- df %>%
  select(
    # Independent vars
    UseInternet,
    WhereSeekHealthInfo,
    Internet_Broadbnd,
    Internet_DialUp,
    Internet_Cell,
    Internet_WiFi,
    TabletHealthWellnessApps,
    OtherDevTrackHealth,
    IntRsn_VisitedSocNet,
    IntRsn_SharedSocNet,
    IntRsn_WroteBlog,
    IntRsn_SupportGroup,
    IntRsn_YouTube
  ) %>%
  mutate(
    UseInternet               = case_when(UseInternet == 1 ~ 1,
                                          T ~ 0),
    WhereSeekHealthInfo       = case_when(WhereSeekHealthInfo == 7 ~ 1,
                                          T ~ 0),
    Internet_Broadbnd         = case_when(Internet_Broadbnd == 1 ~ 1,
                                          T ~ 0),
    Internet_DialUp           = case_when(Internet_DialUp == 1 ~ 1,
                                          T ~ 0),
    Internet_Cell             = case_when(Internet_Cell == 1 ~ 1,
                                          T ~ 0),
    Internet_WiFi             = case_when(Internet_WiFi == 1 ~ 1,
                                          T ~ 0),
    TabletHealthWellnessApps  = case_when(TabletHealthWellnessApps == 1 ~ 1,
                                          T ~ 0),
    OtherDevTrackHealth       = case_when(OtherDevTrackHealth == 1 ~ 1,
                                          T ~ 0),
    IntRsn_VisitedSocNet      = case_when(IntRsn_VisitedSocNet == 1 ~ 1,
                                          T ~ 0),
    IntRsn_SharedSocNet       = case_when(IntRsn_SharedSocNet == 1 ~ 1,
```

```
                                    T ~ 0),
    IntRsn_WroteBlog          = case_when(IntRsn_WroteBlog == 1 ~ 1,
                                    T ~ 0),
    IntRsn_SupportGroup       = case_when(IntRsn_SupportGroup == 1 ~ 1,
                                    T ~ 0),
    IntRsn_YouTube            = case_when(IntRsn_YouTube == 1 ~ 1,
                                    T ~ 0)
  )
```

To determine which value of the parameter `m` to use with logistic PCA (note that the `m` parameter specifies how the saturated model approximates the natural parameters of the data), we perform a simple cross-validation grid search to find the value that minimises negative log-likelihood at the desired amount of principal components to be returned.

```
cv.lpca(d.tech, ks = 5, ms = 1:10) %>% plot()
```



With m = 8, our 5 principal components cover 77% of the variance from the original dataset. Sacrificing 23% of the variance, in exchange for reducing the number of variables from 13 to 5, can be considered a worthwhile endeavour.

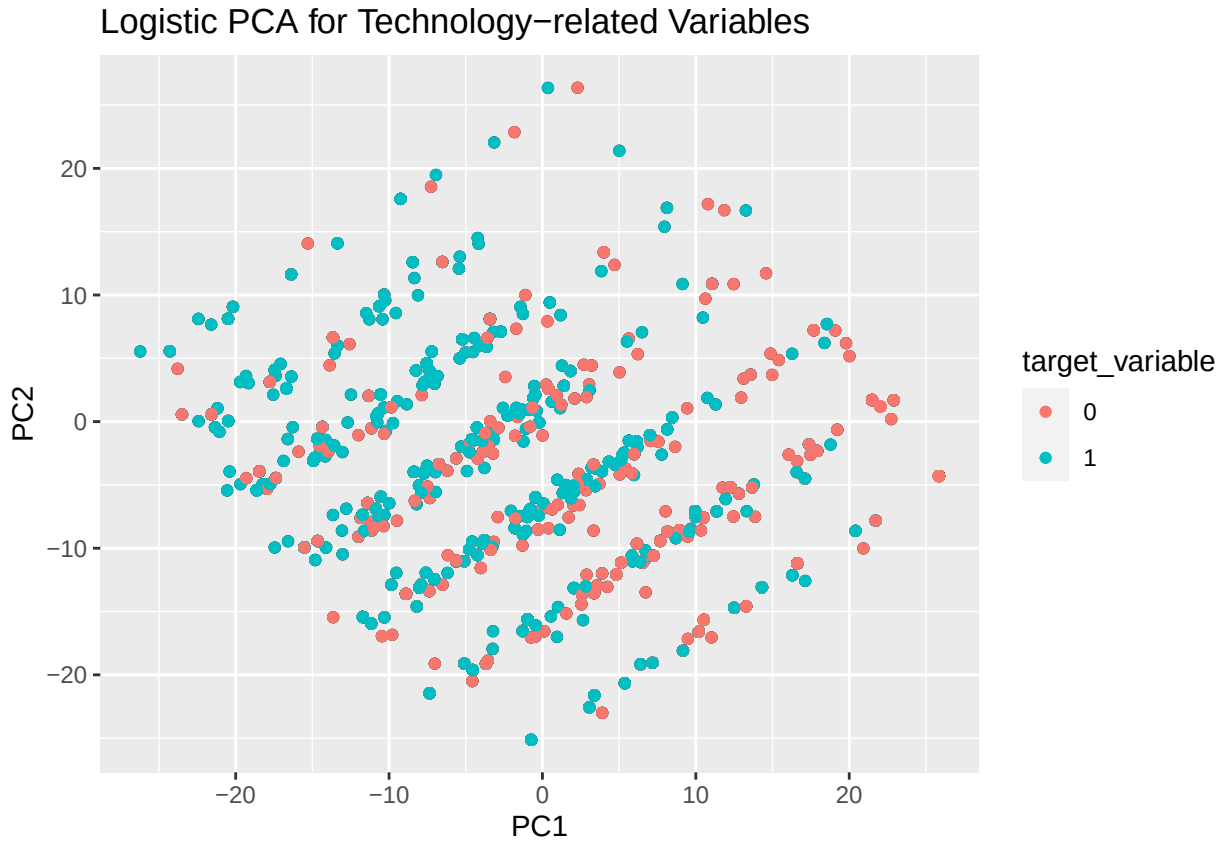
```
pca.tech <- logisticPCA(d.tech, k = 5, m = 8)
pca.tech
```

```
## 1863 rows and 13 columns
## Rank 5 solution with m = 8
```

```
##
## 77% of deviance explained
## 101 iterations to converge
```

When plotted across PC1 & PC2, the data does not look to be immediately linearly separable. This is likely due to the fact that PCA as a technique does not take into account the target variable in its calculations.

```
target_variable <- as.character(df$y)

plot(pca.tech, type = "scores") +
  geom_point(aes(col = target_variable)) +
  ggtitle("Logistic PCA for Technology-related Variables")
```



We save the 5 PCs, as future model development might make use of these variables. For current-stage modeling, we will let `tidymodels` handle the PCA backend.

```
d.tech.pca <- pca.tech$PCs %>%
  as.data.frame() %>%
  transmute(
    techPC1 = V1,
    techPC2 = V2,
    techPC3 = V3,
    techPC4 = V4,
    techPC5 = V5
  ) %>%
  rownames_to_column("new_id")
```

Feature Engineering 3: Health-mindedness & Lifestyle

Transformations for this section include:

- SeekHealthInfo: This simply encodes all patients who have sought health information from any source as 1, and all others (including missing data) as 0.
- MostRecentCheckup2: Reversed order from 4 to 0 - 4 now represents patients who had a checkup in the past year, 3 for 1-2 years ago, 2 for 3-5 years ago, 1 for more than 5 years, and 0 for "Never", "Don't know", and missing data.
- FreqGoProvider: kept as-is, as this is simply a count variable. Missing data is coerced to 0 (none).
- GeneralHealth: recoded to be from 4 to 0, with 4 being Excellent, descending to 0 being Poor. Missing and error data is assigned to "Fair" - corresponding to 1 in the new schema.
- HowLongModerateExerciseMinutes: kept as-is, as this is simply a count variable. Missing data is coerced to 0 (none).
- TimesStrengthTraining: kept as-is, as this is simply a count variable. Missing data is coerced to 0 (none).
- AverageTimeSitting: kept as-is, as this is simply a count variable. Missing data is coerced to 0 (none).

```
d.life <- df %>%
  select(
    # Independent vars
    SeekHealthInfo,
    MostRecentCheckup2,
    FreqGoProvider,
    GeneralHealth,
    HowLongModerateExerciseMinutes,
    TimesStrengthTraining,
    AverageTimeSitting
  ) %>%
  mutate(
    SeekHealthInfo                = case_when(SeekHealthInfo == 1 ~ 1,
                                              T ~ 0),
    # Re-assigning: 4 to 0, by descending order of recency
    MostRecentCheckup2            = case_when(MostRecentCheckup2 == 1 ~ 4,
                                              MostRecentCheckup2 == 2 ~ 3,
                                              MostRecentCheckup2 == 3 ~ 2,
                                              MostRecentCheckup2 == 4 ~ 1,
                                              T ~ 0),
    # Keeping as is, assigning -9 missing as 0
    FreqGoProvider                = case_when(FreqGoProvider == -9 ~ 0,
                                              T ~ FreqGoProvider),
    # Re-assigning, 4 to 0, by descending order of condition,
    # -9 assigned as "fair", or 1
    GeneralHealth                 = case_when(GeneralHealth == 1 ~ 4,
                                              GeneralHealth == 2 ~ 3,
                                              GeneralHealth == 3 ~ 2,
                                              GeneralHealth %in% c(-9,-5,4) ~ 1,
                                              GeneralHealth == 5 ~ 0),
    # Keeping as is, assigning missing and errors as 0
```

```
    HowLongModerateExerciseMinutes = case_when(HowLongModerateExerciseMinutes <= 0 ~ 0,
                                               T ~ HowLongModerateExerciseMinutes),
    # Keeping as is, assigning missing and errors as 0
    TimesStrengthTraining          = case_when(TimesStrengthTraining <= 0 ~ 0,
                                               T ~ TimesStrengthTraining),
    # Keeping as is, assigning missing and errors as 0
    AverageTimeSitting             = case_when(AverageTimeSitting <= 0 ~ 0,
                                               T ~ AverageTimeSitting)
  )
```

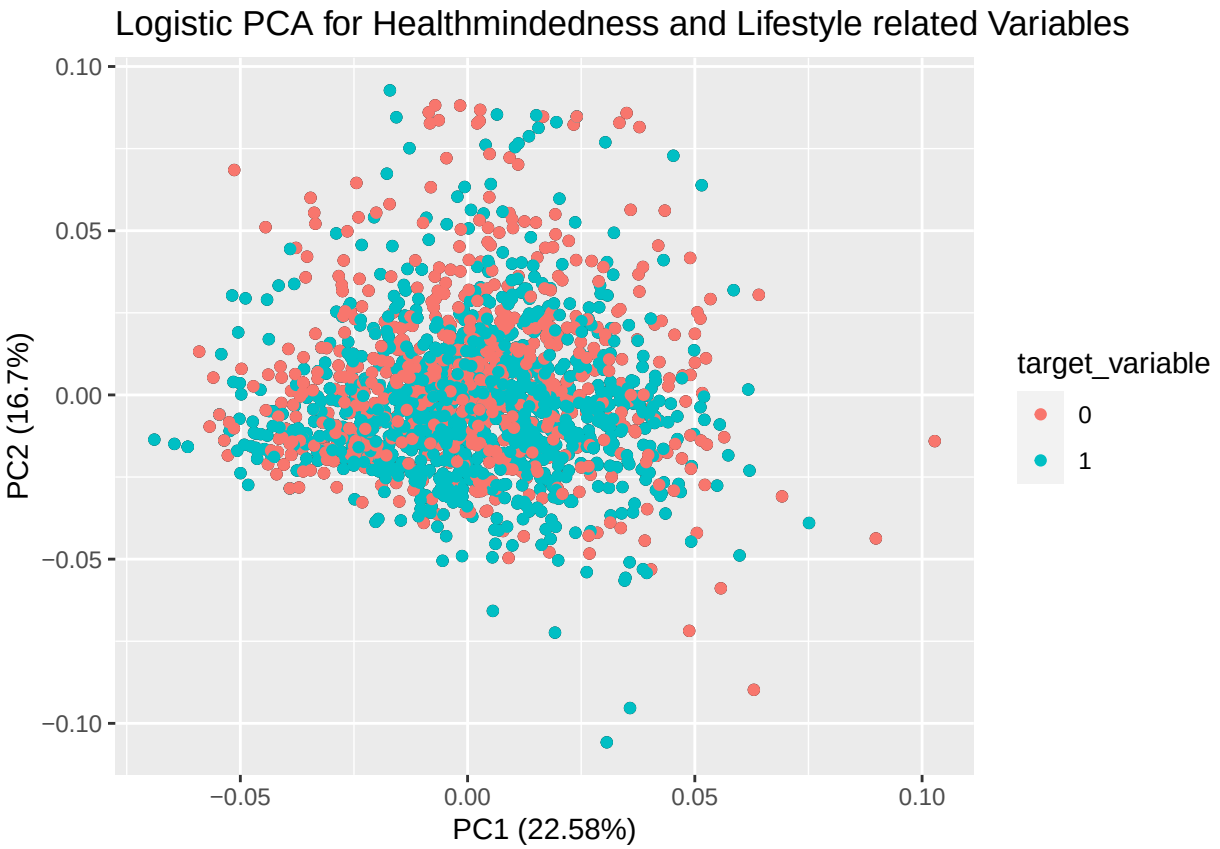
A conventional PCA process is used here. 5 PCs explain 79.97% of variance, and reduces the dimensionality significantly.

```
pca.life <- prcomp(d.life,center = T, scale. = T)

pca.life %>% summary()
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4    PC5    PC6     PC7
## Standard deviation     1.2573 1.0811 1.0368 0.9477 0.9355 0.8792 0.79312
## Proportion of Variance 0.2258 0.1670 0.1536 0.1283 0.1250 0.1104 0.08986
## Cumulative Proportion  0.2258 0.3928 0.5464 0.6747 0.7997 0.9101 1.00000
```

When plotted and colored by response variable, once again we see that the data is not very linearly separable.

```
target_variable <- as.character(df$y)

autoplot(pca.life) +
  geom_point(aes(col = target_variable)) +
  ggtitle("Logistic PCA for Healthmindedness and Lifestyle related Variables")
```



We still save 5 PCs, as future model development might make use of these variables. For current-stage modeling, we will let `tidymodels` handle the PCA backend.

```
d.life.pca <- pca.life$x %>%
  as.data.frame() %>%
  transmute(
    lifePC1 = PC1,
    lifePC2 = PC2,
    lifePC3 = PC3,
    lifePC4 = PC4,
    lifePC5 = PC5
  ) %>% rownames_to_column("new_id")
```

Feature Engineering 4: Perceived Need/Availability of EMR

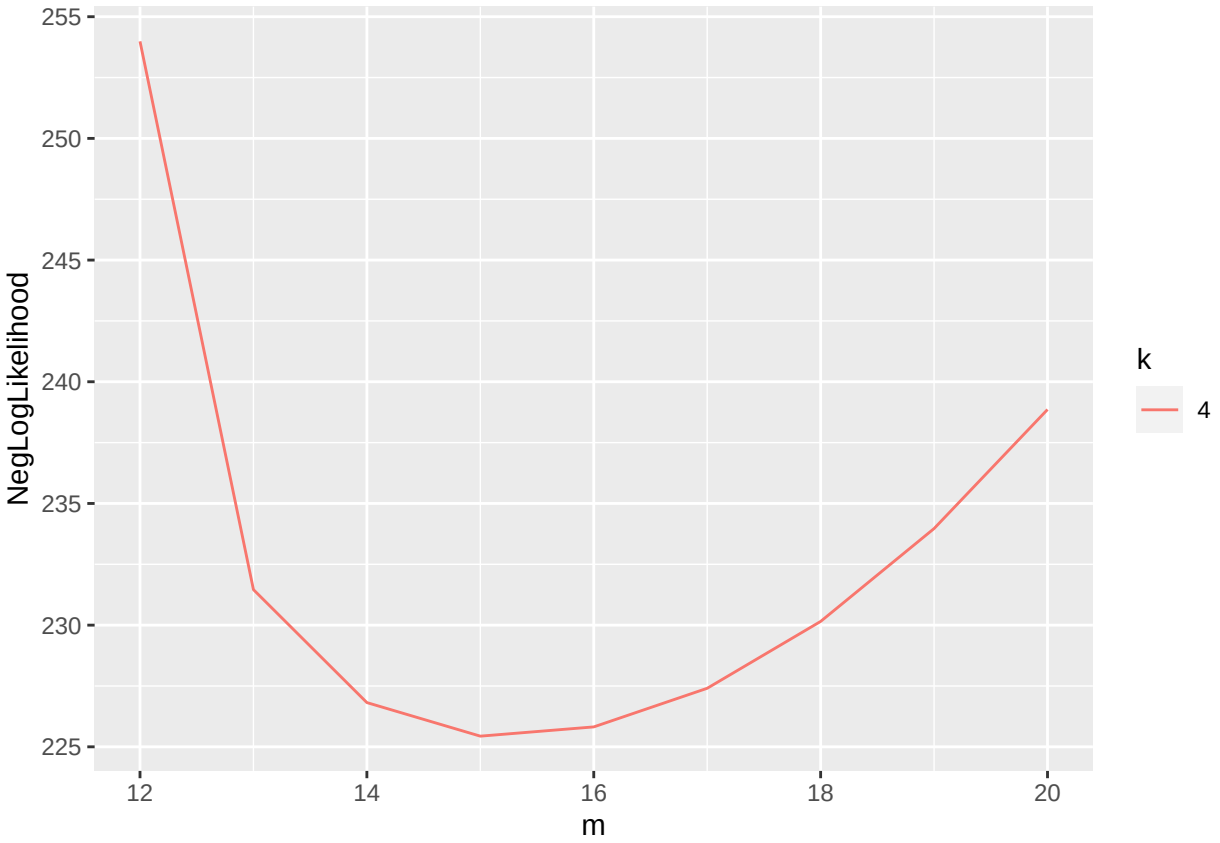
All variables used in this section are, in the original dataset, coded simply as a binary variable with one extra value to account for missing data. Missing data in this section are all collapsed into 0/FALSE.

```
d.access <- df %>%
  select(
    # Independent vars
    ProviderMaintainEMR2,
    WhoOffered_HCP,
    WhoOffered_Insurer,
    ESent_AnotherHCP,
```

```
    ProbCare_BringTest,
    ProbCare_RedoTest,
    ProbCare_ProvideHist
  ) %>%
  mutate(
    # Assigning yes as 1, everything else as 0
    ProviderMaintainEMR2 = case_when(ProviderMaintainEMR2 == 1 ~ 1,
                                     T ~ 0),
    # Assigning yes as 1, everything else as 0
    WhoOffered_HCP       = case_when(WhoOffered_HCP == 1 ~ 1,
                                     T ~ 0),
    # Assigning yes as 1, everything else as 0
    WhoOffered_Insurer   = case_when(WhoOffered_Insurer == 1 ~ 1,
                                     T ~ 0),
    # Assigning yes as 1, everything else as 0
    ESent_AnotherHCP     = case_when(ESent_AnotherHCP == 1 ~ 1,
                                     T ~ 0),
    # Assigning yes as 1, everything else as 0
    ProbCare_BringTest   = case_when(ProbCare_BringTest == 1 ~ 1,
                                     T ~ 0),
    # Assigning yes as 1, everything else as 0
    ProbCare_RedoTest    = case_when(ProbCare_RedoTest == 1 ~ 1,
                                     T ~ 0),
    # Assigning yes as 1, everything else as 0
    ProbCare_ProvideHist = case_when(ProbCare_ProvideHist == 1 ~ 1,
                                     T ~ 0)
  )
```

Similar to before, we search for the optimal m value.

```
cv.lpca(d.access, ks = 4, ms = 12:20) %>% plot()
```



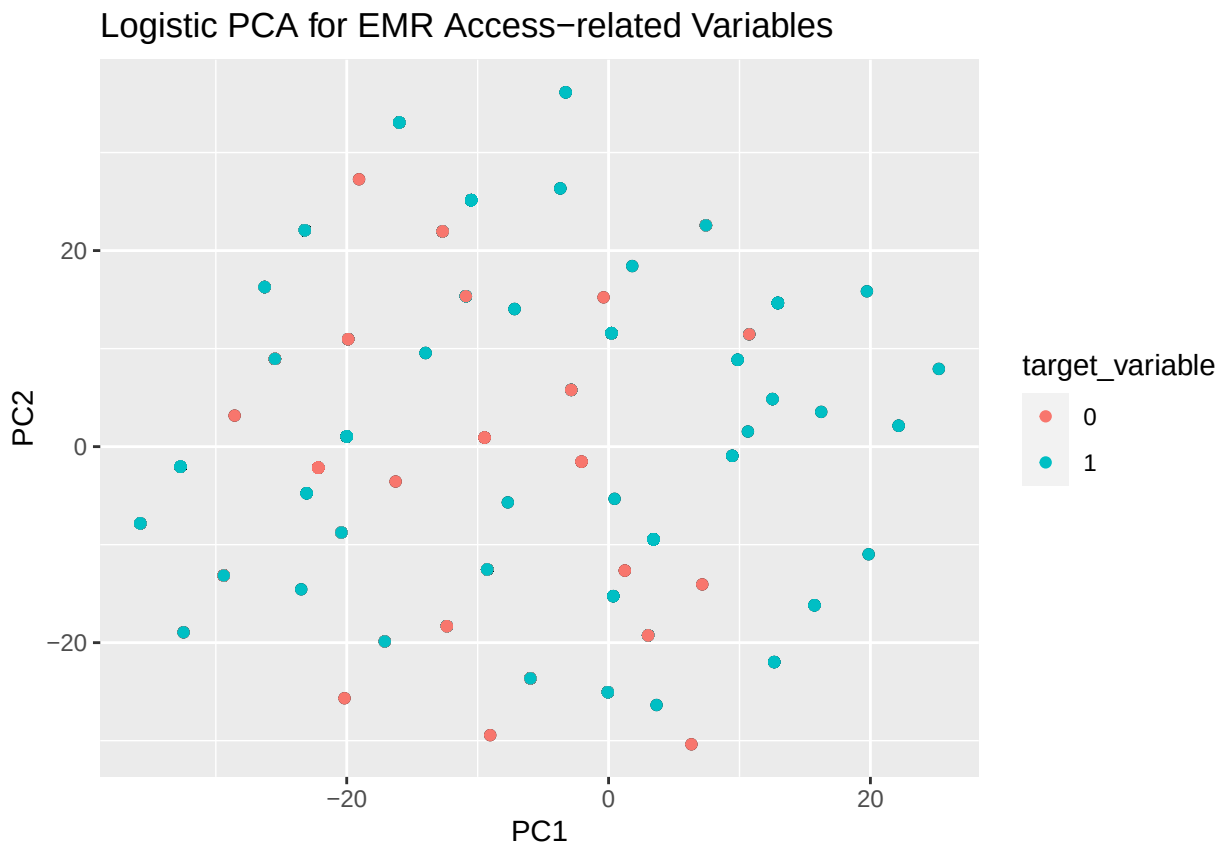
With m = 16, our 4 PCs explain 95.2% of the variance, which is very good.

```
pca.access <- logisticPCA(d.access, k = 4, m = 16)
pca.access
```

```
## 1863 rows and 7 columns
## Rank 4 solution with m = 16
##
## 95.2% of deviance explained
## 551 iterations to converge
```

When plotted and colored by response variable, once again we see that the data is not very linearly separable.

```
target_variable <- as.character(df$y)

plot(pca.access, type = "scores") +
  geom_point(aes(col = target_variable)) +
  ggtitle("Logistic PCA for EMR Access-related Variables")
```



We still save the 4 PCs, as future model development might make use of these variables. For current-stage modeling, we will let `tidymodels` handle the PCA backend.

```
d.access.pca <- pca.access$PCs %>%
  as.data.frame() %>%
  transmute(
    accessPC1 = V1,
    accessPC2 = V2,
    accessPC3 = V3,
    accessPC4 = V4
  ) %>%
  rownames_to_column("new_id")
```

Feature Engineering 5: Security Concerns:

This section only contains 2 variables:

- ConfidentInfoSafe: Missing data and "Not Confident" are coded as 0, "Somewhat Confident" is coded as 1, and "Very Confident" is coded as 2.
- WithheldInfoPrivacy: Left as-is as a binary variable. Missing data is assigned to 0 (No).

```
d.trust <- df %>%
  select(
    # Independent vars
```

```
    ConfidentInfoSafe,
    WithheldInfoPrivacy
  ) %>%
  mutate(
    ConfidentInfoSafe   = case_when(ConfidentInfoSafe %in% c(-9,-5,3) ~ 0,
                                    ConfidentInfoSafe == 2 ~ 1,
                                    ConfidentInfoSafe == 1 ~ 2),
    WithheldInfoPrivacy = case_when(WithheldInfoPrivacy == 1 ~ 1,
                                    T ~ 0)
  )
```

Re-joining Engineered Features:

Having processed the variables separately by topic section, here we combine the separate categories' tables into one.

```
d.all.nopca <- cbind(d.demo,d.tech,d.life,d.access,d.trust) %>% select(-new_id)
d.all.pca   <- cbind(d.demo,d.tech.pca,d.life.pca,d.access.pca,d.trust) %>% select(-new_id)
```

Modeling w/ tidymodels

Splitting into training/test/validation sets

Setting a seed for reproducibility, we split the data 75/25 into a training and holdout test set, stratified by target variable to account for any imbalance. Of the 75% training split, at each bootstrapping/cross-validation grid-search step, we take 80% as the analysis set, and 20% as the assessment set. In other words, the cross-validation processes here have 5 folds of 20% each.

```
set.seed(8675309)

# splitting to train/test
splits  <- initial_split(d.all.nopca,
                         prop = .75,
                         strata = y)

d.other <- training(splits)
d.test  <- testing(splits)

# Setting validation split for CV resampling
val_set <- validation_split(d.other,
                            prop = .8,
                            strata = y)
```

Creating processing workflow & model specification

`tidymodels` allows us to mass-process variables. The steps here are as follows:

- `step_dummy` transforms listed variables into one-hot encoded dummy variables.

- `step_zv` removes all predictors with zero variance. This is not needed for our current dataset, but is included for the sake of caution should this code be reused on new data.
- `step_normalize` centers and scales all numeric data. This is not strictly necessary, but helps with the execution of logistic regression and PCA.
- `step_pca` executes Principal Component Analysis on listed variables as a group. There are 3 groups here for 3 topic categories (Technology Use, Health & Lifestyle, and EMR Access/Availability).

These steps are stored in a `tidymodels::recipe` object, to allow these pre-processing steps to be stored in the final model, which gives us flexibility to predict on new data should the need arise.

```
recipe.d <- recipe(y ~ ., data = d.other) %>%
  # This step creates dummy variables out of discrete nominal variables.
  step_dummy(
    AgeGrpB,
    OccupationStatus,
    MaritalStatus,
    EducA,
    SpeakEnglish,
    RaceEthn5,
    SelfGender,
    SexualOrientation,
    RentOrOwn, -all_outcomes()
  ) %>%
  # This step removes all zero-variance variables. For our data this is not needed, but included
  # for the sake of caution.
  step_zv(all_numeric_predictors()) %>%
  # Centering and scaling of numeric variables.
  step_normalize(all_numeric_predictors()) %>%
  # Per-category PCA for Technology
  step_pca(
    UseInternet,
    WhereSeekHealthInfo,
    Internet_Broadbnd,
    Internet_DialUp,
    Internet_Cell,
    Internet_WiFi,
    TabletHealthWellnessApps,
    OtherDevTrackHealth,
    IntRsn_VisitedSocNet,
    IntRsn_SharedSocNet,
    IntRsn_WroteBlog,
    IntRsn_SupportGroup,
    IntRsn_YouTube,
    num_comp = 5, prefix = "tech.pc.") %>%
  # Per-category PCA for Health-mindedness and lifestyle
  step_pca(
    SeekHealthInfo,
    MostRecentCheckup2,
    FreqGoProvider,
    GeneralHealth,
    HowLongModerateExerciseMinutes,
    TimesStrengthTraining,
```

```
    AverageTimeSitting,
    num_comp = 5, prefix = "life.pc."
  ) %>%
  # Per-category PCA for Access
  step_pca(
    ProviderMaintainEMR2,
    WhoOffered_HCP,
    WhoOffered_Insurer,
    ESent_AnotherHCP,
    ProbCare_BringTest,
    ProbCare_RedoTest,
    ProbCare_ProvideHist,
    num_comp = 4, prefix = "access.pc."
  )
```

Train & Tune

We will be building two models:

- Logistic Regression: using the `glmnet` package, we will build a binary classifier based on the GLM family of models.
- Linear SVM: using the `kernlab` package, we will build a linear SVM model.

Both models will undergo a grid-search/resampling process to tune for best hyperparameters.

GLMNET-based Logistic Regression

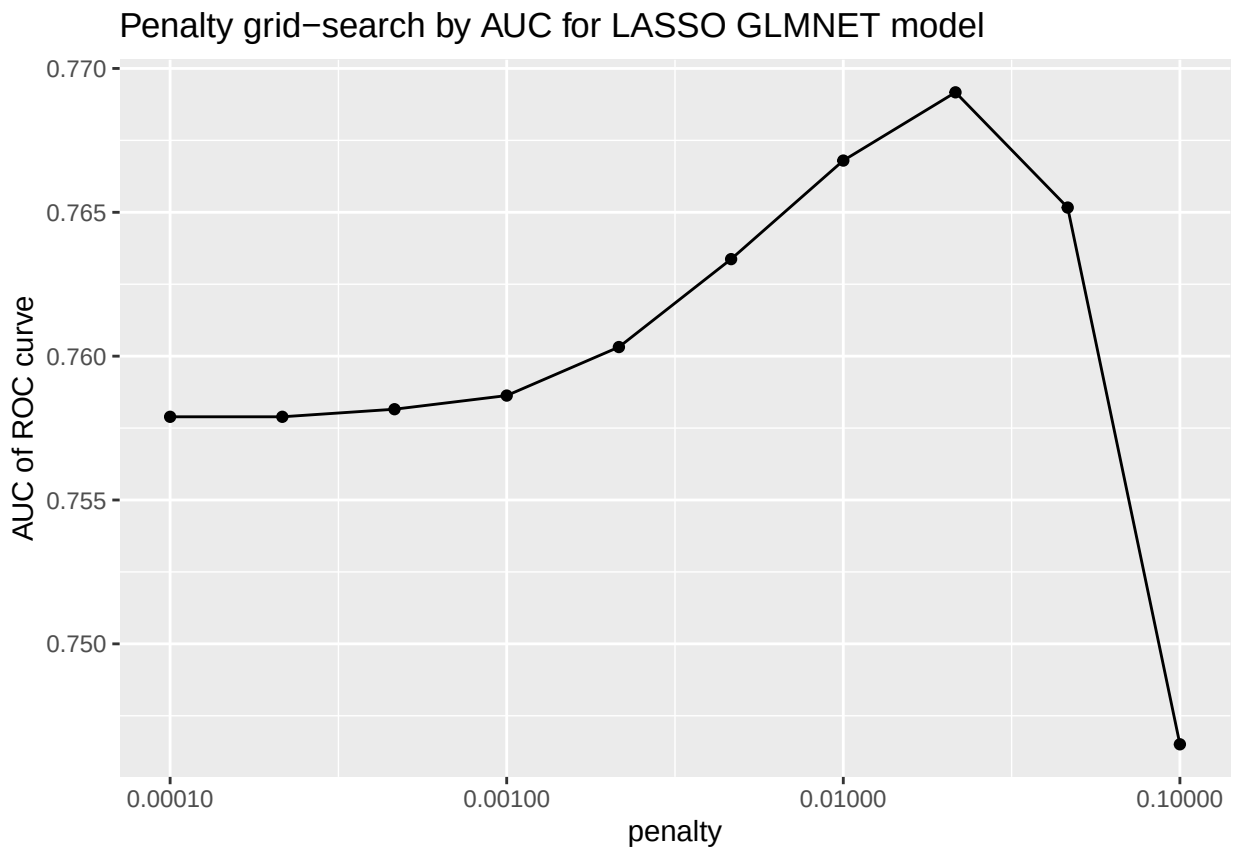
For `glmnet` logistic regression, the default setting is set to alpha = 1, which means there is only LASSO penalty. This penalty term is tuned according to a grid:

```
grid.logreg <- tibble(
  penalty = c(10^seq(-4,-1,length.out = 10))
)

grid.logreg
```

```
## # A tibble: 10 x 1
##     penalty
##       <dbl>
##  1 0.0001
##  2 0.000215
##  3 0.000464
##  4 0.001
##  5 0.00215
##  6 0.00464
##  7 0.01
##  8 0.0215
##  9 0.0464
## 10 0.1
```

Results of the parameter grid search, tuned over AUC of the ROC curve is as below:

```
model.logreg <-
  logistic_reg(penalty = tune(),
               mixture = 1) %>%
  set_engine("glmnet")

wf.logreg <-
  workflow() %>%
  add_model(model.logreg) %>%
  add_recipe(recipe.d)

tune.logreg <-
  wf.logreg %>%
  tune_grid(val_set,
            grid = grid.logreg,
            control = control_grid(save_pred = T),
            metrics = metric_set(roc_auc))

# Keep best param
best.logreg <- tune.logreg %>%
  select_best(metric = "roc_auc")

# Plotting tuning results
tune.logreg %>%
  collect_metrics() %>%
  ggplot(aes(x = penalty,
             y = mean)) +
  geom_point()+
  geom_line()+
  ylab("AUC of ROC curve")+
  scale_x_log10(labels = scales::label_number())+
  labs(title = "Penalty grid-search by AUC for LASSO GLMNET model")
```



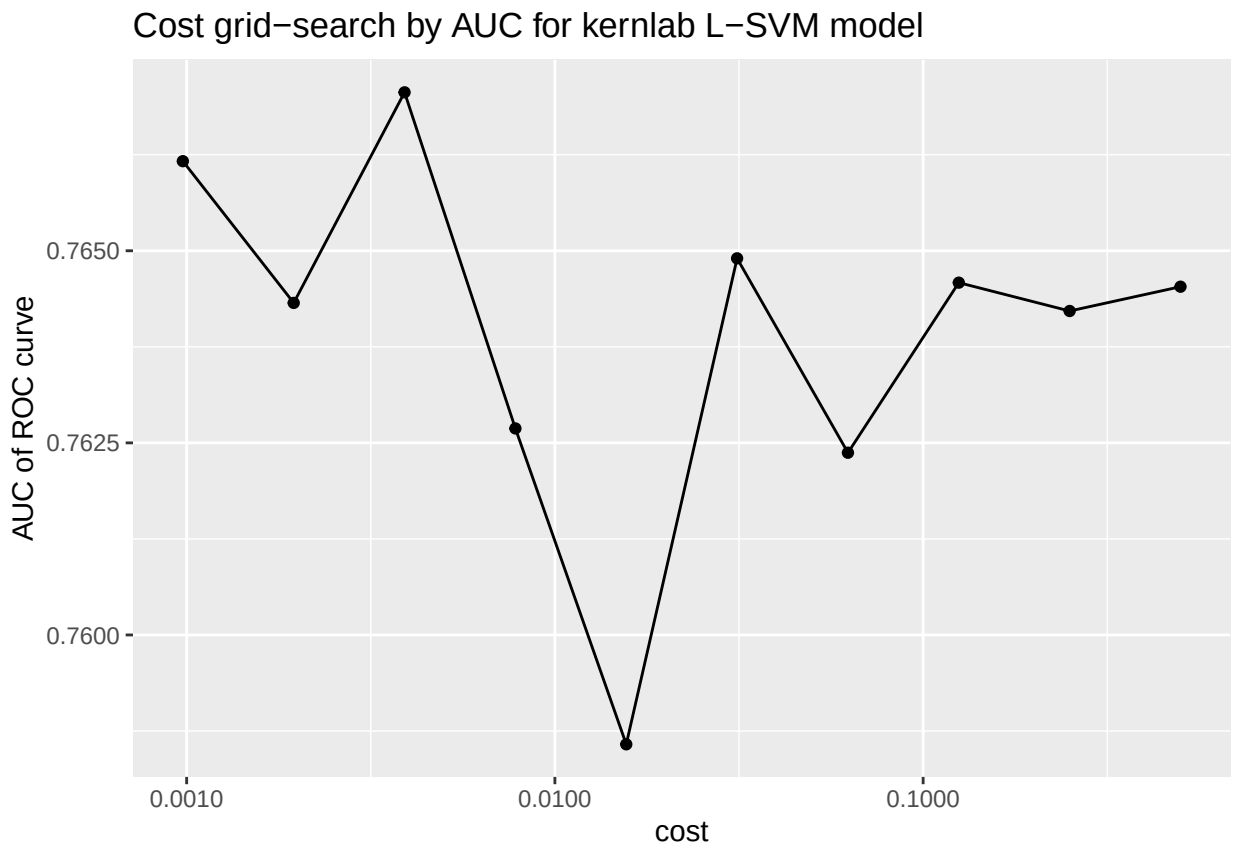
kernlab-based linear SVM

For `kernlab` based linear SVM, we will be tuning the `cost` parameter. The grid is as follows (note that the specific interval is the result of extra preliminary optimisation loops):

```
grid.svm <- tibble(
  cost = c(2^seq(-10,-1,length.out = 10))
)

grid.svm
```

```
## # A tibble: 10 x 1
##        cost
##       <dbl>
##  1 0.000977
##  2 0.00195
##  3 0.00391
##  4 0.00781
##  5 0.0156
##  6 0.0312
##  7 0.0625
##  8 0.125
##  9 0.25
## 10 0.5
```

```
model.svm <-
  svm_linear(mode = "classification",
             cost = tune()) %>%
  set_engine("kernlab")

wf.svm <-
  workflow() %>%
  add_model(model.svm) %>%
  add_recipe(recipe.d)

tune.svm <-
  wf.svm %>%
  tune_grid(val_set,
            grid = grid.svm,
            control = control_grid(save_pred = T),
            metrics = metric_set(roc_auc))

# Keep best param
best.svm <- tune.svm %>%
  select_best(metric = "roc_auc")

# Plotting tuning results
tune.svm %>%
  collect_metrics() %>%
  ggplot(aes(x = cost,
             y = mean)) +
  geom_point()+
  geom_line()+
  ylab("AUC of ROC curve")+
  scale_x_log10(labels = scales::label_number())+
  labs(title = "Cost grid-search by AUC for kernlab L-SVM model")
```

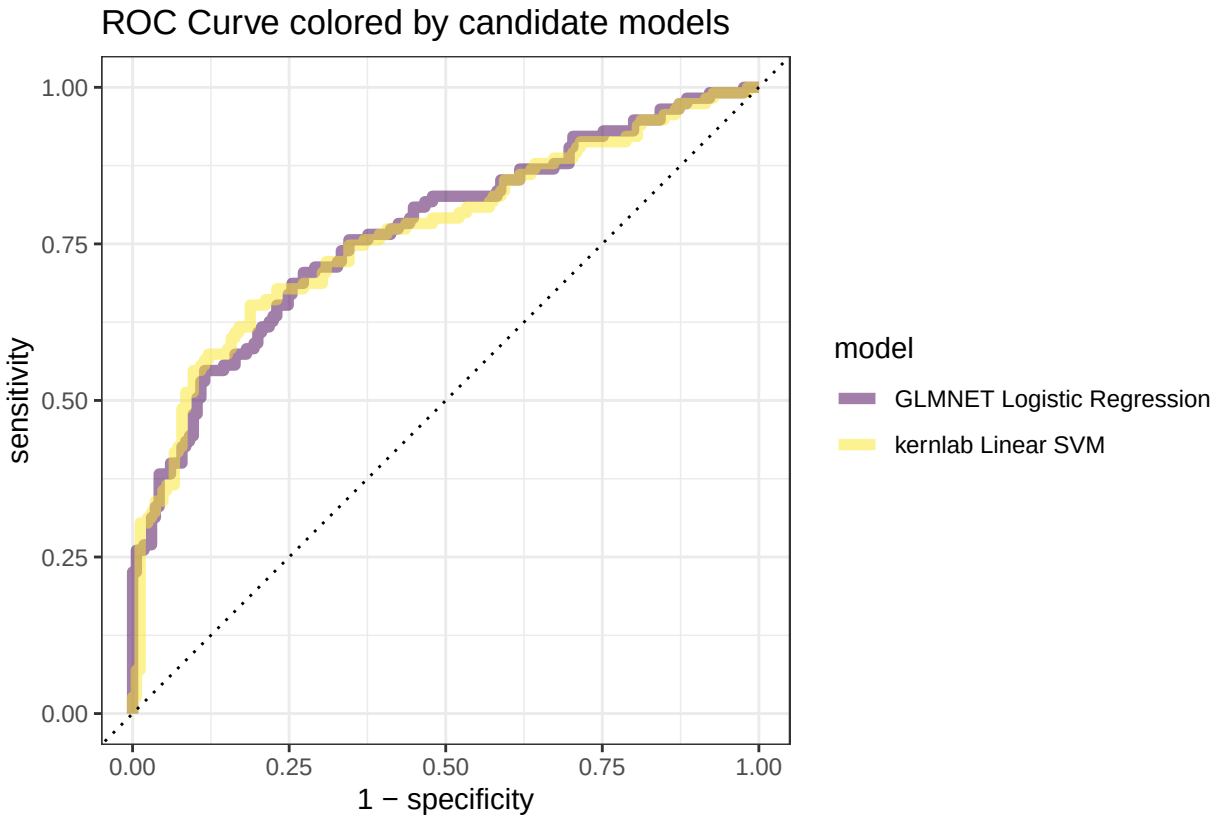


Comparing Candidate Models

The two models' performance can be observed in the overlaid ROC curve below. We can see that the two models have relatively similar performance, and the ROC convex hull appears small enough that the extra effort in building a mixed model is not worth the marginal gain in performance.

```
# Comparison
auc.logreg <- tune.logreg %>%
  collect_predictions(parameters = best.logreg) %>%
  roc_curve(y,.pred_0) %>%
  mutate(model = "GLMNET Logistic Regression")

auc.svm    <- tune.svm %>%
  collect_predictions(parameters = best.svm) %>%
  roc_curve(y,.pred_0) %>%
  mutate(model = "kernlab Linear SVM")

rbind(auc.logreg,auc.svm) %>%
  ggplot(
    aes(
      x = 1-specificity,
      y = sensitivity,
      col = model,
      fill = model
    )
```

```
  )+
  geom_path(size = 2,
            alpha = .5)+
  geom_abline(lty = 3)+
  coord_equal()+
  scale_color_viridis_d()+
  theme_bw() +
  labs(title = "ROC Curve colored by candidate models")
```



The AUC values of the above ROC curve are tabled below. We can see that there is very marginal difference between the two.

```
auc.logreg <- tune.logreg %>%
  collect_predictions(parameters = best.logreg) %>% roc_auc(y,.pred_0) %>% .$.estimate

auc.svm    <- tune.svm %>%
  collect_predictions(parameters = best.svm) %>% roc_auc(y,.pred_0) %>% .$.estimate

data.frame(
  Model = c("Log. Regression","Linear SVM"),
  AUC   = c(auc.logreg, auc.svm)
)
```

```
##             Model       AUC
## 1 Log. Regression 0.7691700
## 2      Linear SVM 0.7670619
```

Building Selected Model

With finalised hyperparameters, we can build the final models for the next steps. We call `set.seed` immediately before modeling to ensure reproducibility.

```
model.logreg.final <-
  logistic_reg(penalty = best.logreg$penalty,
               mixture = 1) %>%
  set_engine("glmnet")

wf.logreg.final <- workflow() %>%
  add_model(model.logreg.final) %>%
  add_recipe(recipe.d)

set.seed(8675309)
fit.logreg.final <- wf.logreg.final %>%
  fit(d.other)

model.svm.final <-
  svm_linear(cost = best.svm$cost,
             mode = "classification") %>%
  set_engine("kernlab")

wf.svm.final <- workflow() %>%
  add_model(model.svm.final) %>%
  add_recipe(recipe.d)

set.seed(8675309)
fit.svm.final <- wf.svm.final %>%
  fit(d.other)
```

```
##  Setting default kernel parameters
```

Evaluation of Selected Model on Test Set

Below are model performances based on the hold-out test set.

```
wf.logreg.test   <-
  wf.logreg %>%
  update_model(model.logreg.final)

wf.svm.test   <-
  wf.svm %>%
  update_model(model.svm.final)

set.seed(8675309)
fit.logreg.test <-
  wf.logreg.test %>%
  last_fit(splits)

set.seed(8675309)
fit.svm.test <-
```

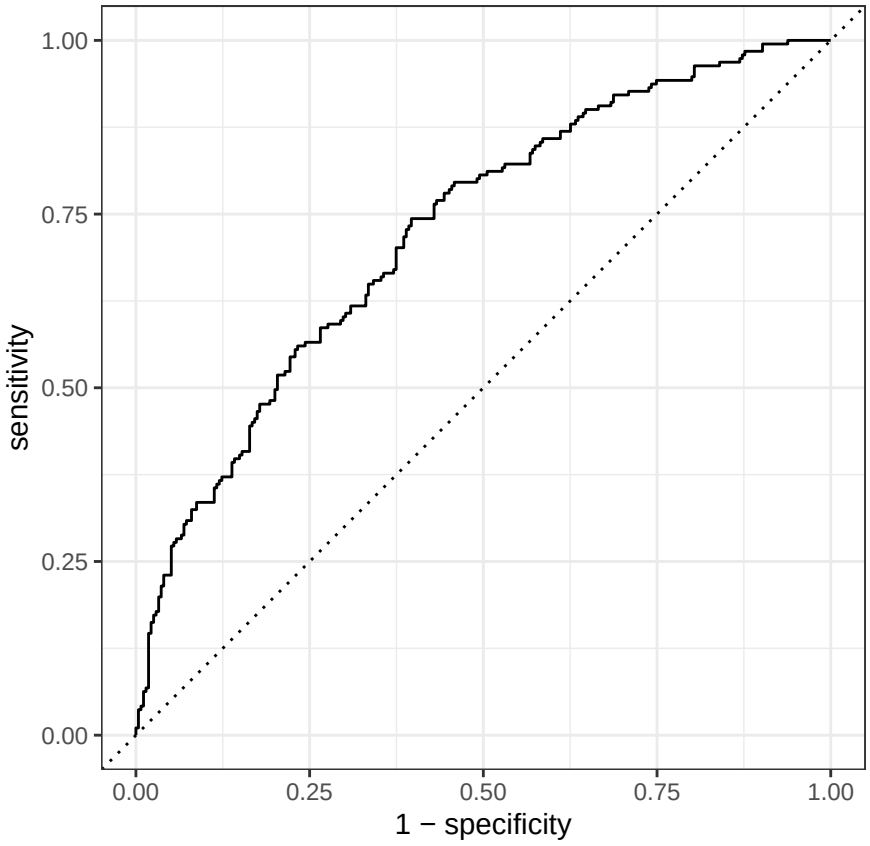
```
  wf.svm.test %>%
  last_fit(splits)
```

Test metric for GLM

```
# Test metrics for Logistic Regression
fit.logreg.test %>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.674 Preprocessor1_Model1
## 2 roc_auc  binary         0.725 Preprocessor1_Model1
```

```
fit.logreg.test %>%
  collect_predictions() %>%
  roc_curve(y, .pred_0) %>%
  autoplot(title("GLMNET logistic regression"))
```

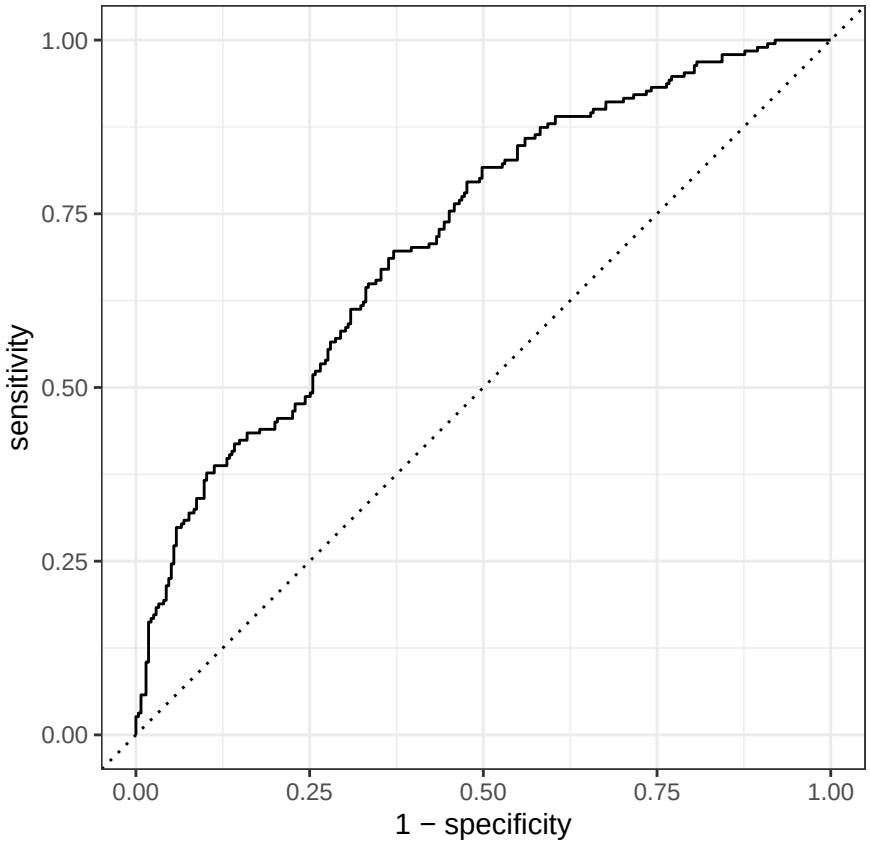


Test metric for L-SVM

```
# Test metrics for l-SVM
fit.svm.test %>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.680 Preprocessor1_Model1
## 2 roc_auc  binary         0.719 Preprocessor1_Model1
```

```
fit.svm.test %>%
  collect_predictions() %>%
  roc_curve(y, .pred_0) %>%
  autoplot(title = "kernlab linear SVM")
```



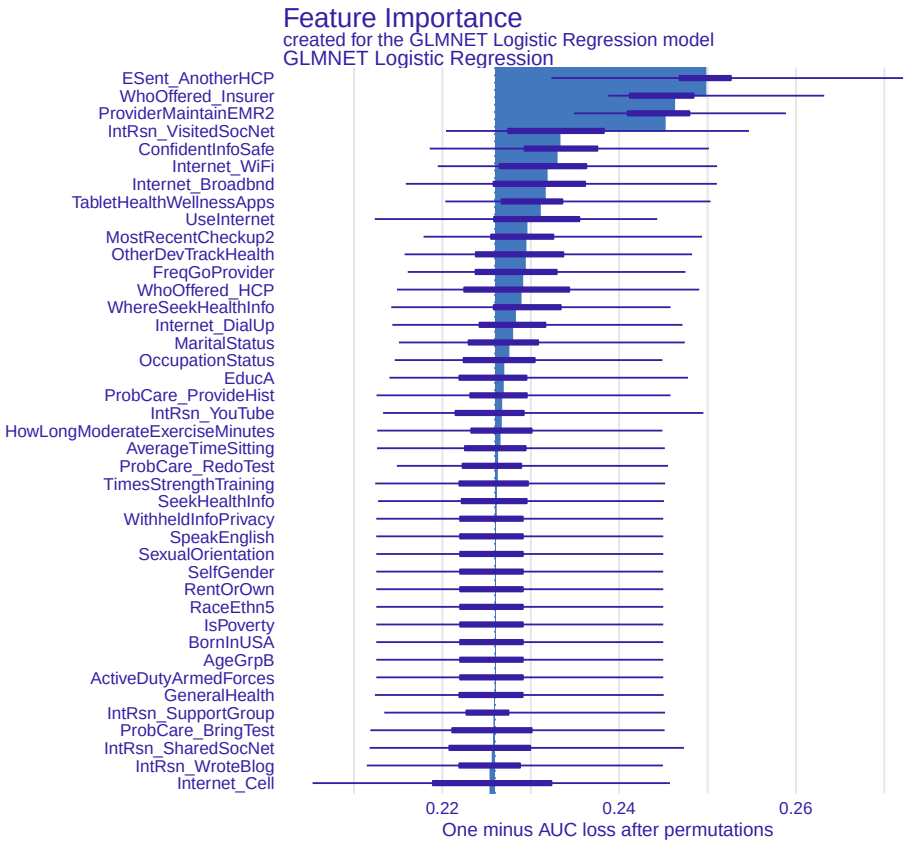
Variable Importance

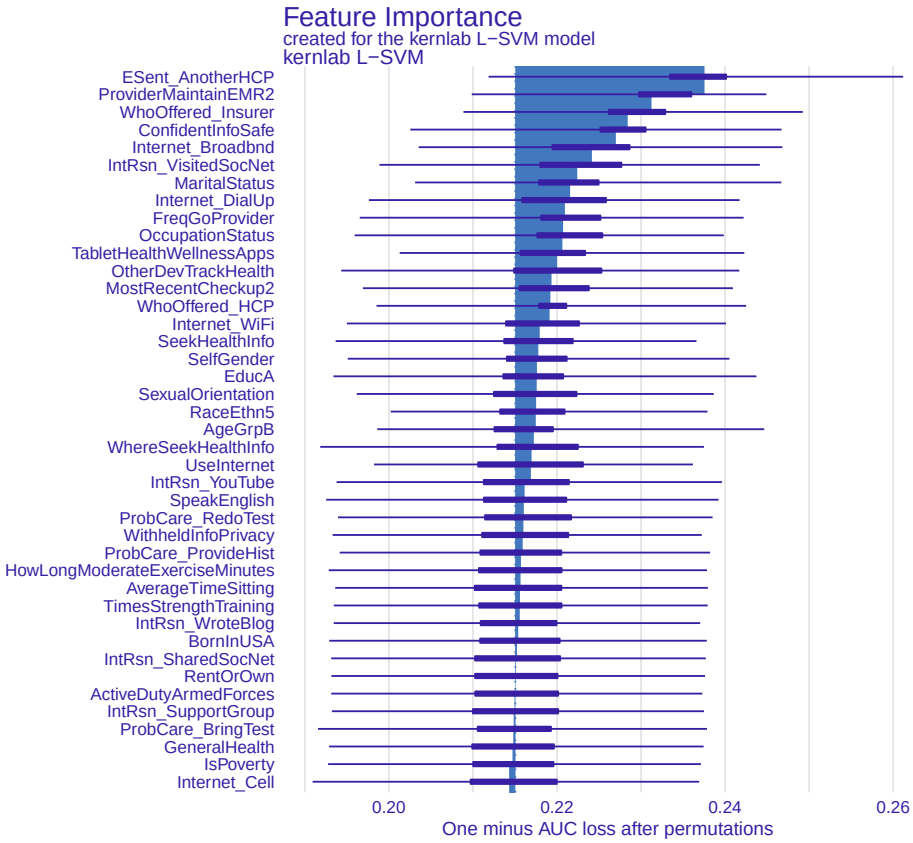
Finally, we can create model explainer objects to perform variable importance analysis. Variable importance is derived by permuting each variable and measuring how much each model's performance suffers, measured by some accuracy metric. Here, the accuracy metric is measured as AUC loss.

```r
# By Logistic Regression
logreg.explainer <- explain_tidymodels(
  model = fit.logreg.final,
  data = d.other %>% select(-y),
  y = d.other %>% pull(y) %>% as.character() %>% as.numeric(),
  label = "GLMNET Logistic Regression",
  verbose = F
)

set.seed(8675309)
logreg.varImp <- model_parts(logreg.explainer)

# By SVM
svm.explainer <- explain_tidymodels(
  model = fit.svm.final,
  data = d.other %>% select(-y),
  y = d.other %>% pull(y) %>% as.character() %>% as.numeric(),
  label = "kernlab L-SVM",
  verbose = F
)

set.seed(8675309)
svm.varImp <- model_parts(svm.explainer)
```

Variable importance is plotted as below:

```r
plot(logreg.varImp, show_boxplots = TRUE)/plot(svm.varImp, show_boxplots = TRUE)
```


Feature Importance
created for the GLMNET Logistic Regression model
GLMNET Logistic Regression
ESent_AnotherHCP
WhoOffered_Insurer
ProviderMaintainEMR2
IntRsn_VisitedSocNet
ConfidentInfoSafe
Internet_WiFi
Internet_Broadbnd
TabletHealthWellnessApps
UseInternet
MostRecentCheckup2
OtherDevTrackHealth
FreqGoProvider
WhoOffered_HCP
WhereSeekHealthInfo
Internet_DialUp
MaritalStatus
OccupationStatus
EducA
ProbCare_ProvideHist
IntRsn_YouTube
HowLongModerateExerciseMinutes
AverageTimeSitting
ProbCare_RedoTest
TimesStrengthTraining
SeekHealthInfo
WithheldInfoPrivacy
SpeakEnglish
SexualOrientation
SelfGender
RentOrOwn
RaceEthn5
IsPoverty
BornInUSA
AgeGrpB
ActiveDutyArmedForces
GeneralHealth
IntRsn_SupportGroup
ProbCare_BringTest
IntRsn_SharedSocNet
IntRsn_WroteBlog
Internet_Cell
0.22
0.24
0.26
One minus AUC loss after permutations


Feature Importance
created for the kernlab L−SVM model
kernlab L−SVM
ESent_AnotherHCP
ProviderMaintainEMR2
WhoOffered_Insurer
ConfidentInfoSafe
Internet_Broadbnd
IntRsn_VisitedSocNet
MaritalStatus
Internet_DialUp
FreqGoProvider
OccupationStatus
TabletHealthWellnessApps
OtherDevTrackHealth
MostRecentCheckup2
WhoOffered_HCP
Internet_WiFi
SeekHealthInfo
SelfGender
EducA
SexualOrientation
RaceEthn5
AgeGrpB
WhereSeekHealthInfo
UseInternet
IntRsn_YouTube
SpeakEnglish
ProbCare_RedoTest
WithheldInfoPrivacy
ProbCare_ProvideHist
HowLongModerateExerciseMinutes
AverageTimeSitting
TimesStrengthTraining
IntRsn_WroteBlog
BornInUSA
IntRsn_SharedSocNet
RentOrOwn
ActiveDutyArmedForces
IntRsn_SupportGroup
ProbCare_BringTest
GeneralHealth
IsPoverty
Internet_Cell
0.20
0.22
0.24
0.26
One minus AUC loss after permutations