ShapleyVIC: Shapley Variable Importance Cloud for Interpretable Machine Learning

Yilin Ning, Chenglin Niu, Mingxuan Li, Siqi Li, Nan Liu

2023-02-03

Table of contents

ShapleyVIC Introduction

Variable importance assessment is important for interpreting machine learning models. Current practice in interpretable machine learning applications focuses on explaining the final models that optimize predictive performance. However, this does not fully address practical needs, where researchers are willing to consider models that are "good enough" but are easier to understand or implement. Shapley variable importance cloud (ShapleyVIC) fills this gap by extending current method to a set of "good models" for comprehensive and robust assessments. Building on a common theoretical basis (i.e., Shapley values for variable importance), ShapleyVIC seamlessly complements the widely adopted SHAP assessments of a single final model to avoid biased inference. Please visit GitHub page for source code.

Usage

As detailed in Chapter 3 ShapleyVIC analysis of variable importance consists of 3 general steps:

1. Training an optimal prediction model (e.g., a logistic regression model).
2. Generating a reasonable number of (e.g., 350) nearly optimal models of the same model class (e.g., logistic regression).
3. Evaluate Shapley-based variable importance from each nearly optimal model and pool information for inference.

ShapleyVIC does not require variable centering or standardization, but requires some data checking and pre-processing for stable and smooth processing, which we summarize in Chapter 2.

The ShapleyVIC-based variable ranking can also be used with the AutoScore framework to develop clinical risk scores for interpretable risk prediction, which we demonstrate in Chapter 4 and Chapter 5.

Installation

The ShapleyVIC framework is now implemented using a **Python library** that trains the optimal model, generates nearly optimal models and evaluate Shapley-based variable impor-

tance from such models, and an **R package** that pools information across models to generate summary statistics and visualizations for inference.

Python library

- **Required:** Python version 3.6 or higher.
  - ***Recommended:*** latest stable release of Python 3.9 or 3.10.
- **Required:** latest version of git.

Execute the following command in Terminal/Command Prompt to install the Python library from GitHub:

- Linux/macOS:

```
pip install git+"https://github.com/nliulab/ShapleyVIC#egg=ShapleyVIC&subdirectory=python"
```

- Windows:

```
python.exe -m pip install git+"https://github.com/nliulab/ShapleyVIC#egg=ShapleyVIC&subdir
```

> **Note**
>
> - *ShapleyVIC uses a modified version of the SAGE library (version 0.0.4b1), which avoids occasional stack overflow problems on Windows but does not affect variable importance evaluation.*

R package

- **Required:** R version 3.5.0 or higher.
  - ***Recommended:*** use latest version of R with RStudio.

Execute the following command in R/RStudio to install the R package from GitHub:

```
if (!require("devtools", quietly = TRUE)) install.packages("devtools")
devtools::install_github("nliulab/ShapleyVIC/r")
```

Citation

Core paper

- Ning Y, Ong ME, Chakraborty B, Goldstein BA, Ting DS, Vaughan R, Liu N. Shapley variable importance cloud for interpretable machine learning. *Patterns* 2022; 3: 100452.

Method extension

- Ning Y, Li S, Ong ME, Xie F, Chakraborty B, Ting DS, Liu N. A novel interpretable machine learning system to generate clinical risk scores: An application for predicting early mortality or unplanned readmission in a retrospective cohort study. *PLOS Digit Health* 2022; 1(6): e0000062.

Contact

- Yilin Ning (Email: yilin.ning@duke-nus.edu.sg)
- Nan Liu (Email: liu.nan@duke-nus.edu.sg)

1 Data requirements

1.1 General requirements

- Currently ShapleyVIC only applies to binary outcomes.
- Code binary outcomes as 0/1.
- No space or special characters (e.g., `[`, `]`, `(`, `)`, `,`) in variable names. Replace them using `_`.
- Variable centering/standardization is *not required*.

1.2 Missing values and sparsity

- Handle missing entries appropriately before applying ShapleyVIC. Missing entry is not allowed.
- Check data distribution and handle data sparsity before applying ShapleyVIC. Data sparsity may increase run time and lead to unstable results.

1.3 Additional pre-processing for high-dimensional data

- Although theoretically permissible, it is not advisable to apply ShapleyVIC to data with a large number of variables.
- Screen out variables with low importance (e.g., based on univariable or multivariable analysis p-values) to reduce dimension (e.g., to $<$50 variables) before applying ShapleyVIC.

2 ShapleyVIC for Variable Importance Assessment

ShapleyVIC is model agnostic, and its benefits of has been demonstrated in empirical experiments in applications for multiple domains. This tutorial illustrates ShapleyVIC implementation using the Python library and R package in a study that predicts 2-year recidivism using a logistic regression of 6 binary variables.

Cite the following papers for ShapleyVIC:

- Ning Y, Ong ME, Chakraborty B, Goldstein BA, Ting DS, Vaughan R, Liu N. Shapley variable importance cloud for interpretable machine learning. *Patterns* 2022

2.1 [Python] ShapleyVIC calculation

This part of the ShapleyVIC workflow is implemented in Python.

In this part of the workflow, we load and prepare data, train optimal logistic regression model, generate nearly optimal models, and compute Shapley-based variable importance for each model.

2.1.1 Load data

- Read data from CSV or Excel files.
- For this demo, use the integrated data in the library that contains 7214 samples analyzed in Experiment 1 (i.e., the recidivism prediction study) of the paper.

```
from ShapleyVIC import df_compas

compas = df_compas.load_data()
# See data description using the following command:
# help(df_compas.load_data)
compas.loc[:5]
```

y	age	race	prior	gender	juvenilecrime	currentcharge	train_test
0	0	0	1	1	1	0	train
1	0	1	1	1	1	0	train
1	0	1	0	1	0	0	train
0	0	1	0	1	0	0	train
0	0	0	0	1	1	0	train
0	0	0	1	1	1	1	train

- y: 2-year recidivism (the binary outcome, 1=event and 0=non-event).
- age, race, prior, gender, juvenilecrime, currentcharge: binary predictors.
- train_test: training/explanation set membership indicator (`"train"` for training and `"test"` for explanation). *Not to include in models.*

2.1.2 Prepare training and explanation sets

- When there is sufficient data, users can split the full dataset into a training set to train optimal and nearly optimal models, and an explanation set to compute ShapleyVIC values.
- Otherwise, users may use the full dataset to train models and compute ShapleyVIC values.

> **!** Important
>
> - *As detailed in Chapter 1, check for and handle data issues before applying ShapleyVIC.*
> - *This demo will show impact of data sparsity on ShapleyVIC results.*

> **!** General suggestions on the size of explanation set
>
> - *Larger number of variables generally requires larger explanation set for stable results.*
> - *Increase in the size of explanation set and/or number of variables increases time required to compute ShapleyVIC values.*
> - *Use of >3500 samples in explanation set leads to long run time and is generally not recommended.*

In the experiment, we used 10% of the full dataset as explanation set:

2.1.2.1 Use the `train_test` indicator available

```
dat_train = compas.loc[compas['train_test']=='train']
# Drop the indicator column after using it to split data:
dat_train = dat_train.drop(columns=['train_test'])
dat_train.reset_index(drop=True, inplace=True)

dat_expl = compas.loc[compas['train_test']=='test']
dat_expl = dat_expl.drop(columns=['train_test'])
dat_expl.reset_index(drop=True, inplace=True)
```

2.1.2.2 Random split for general cases

```
# Drop the column 'train_test' that indicates set membership in example data:
compas = compas.drop(columns=['train_test'])
# Generate row indices for training and explanation sets:
from sklearn.model_selection import train_test_split
i_train, i_expl = train_test_split(list(range(compas.shape[0])),
    test_size=int(0.1 * compas.shape[0]), random_state=0)

dat_train = compas.iloc[i_train, :]
dat_train.reset_index(drop=True, inplace=True)

dat_expl = compas.iloc[i_expl, :]
dat_expl.reset_index(drop=True, inplace=True)
```

2.1.3 Train optimal model

- Specify training data to initialize the model object and train the optimal model.
- `x`, `y`: predictors (as a data frame) and outcome from the training set.
- `outcome_type`: type of the outcome (currently only supports binary outcomes).
- `x_names_cat`: names of categorical predictors. Optional for binary predictors encoded as 0/1.
- `output_dir`: the directory to save key outputs to. Will be used as input in the subsequent R workflow.
- `save_data`: whether to save `x` and `y` to `output_dir` (default is to save). If not, `x` and `y` must be supplied separately in subsequent R analysis.

2.1.3.1 Save data

Default option is to save input data `x` and `y`, so that they are not needed as input in the subsequent R workflow.

```
# Specify the name of outcome, which is 'y' in this example:
y_name = 'y'
from ShapleyVIC import model
model_object = model.models(
    x=dat_train.drop(columns=[y_name]), y=dat_train[y_name],
    outcome_type="binary",
    x_names_cat=['age','race','prior','gender','juvenilecrime','currentcharge'],
    output_dir="compas_output", save_data=True
)
# To display the optimal logistic regression trained:
model_object.model_optim.summary().tables[1]
```

2.1.3.2 Do not save data

Specify `save_data=False` to avoid saving `x` and `y` to output folder. See Chapter 3 for another example.

```
# Specify the name of outcome, which is 'y' in this example:
y_name = 'y'
from ShapleyVIC import model
model_object = model.models(
    x=dat_train.drop(columns=[y_name]), y=dat_train[y_name],
    outcome_type="binary",
    x_names_cat=['age','race','prior','gender','juvenilecrime','currentcharge'],
    output_dir="compas_output", save_data=False
)
# To display the optimal logistic regression trained:
model_object.model_optim.summary().tables[1]
```

	coef	std err	z	P	[0.025	0.975]
const	0.4455	0.107	4.160	0.000	0.236	0.655
age	1.5001	0.187	8.011	0.000	1.133	1.867
race	0.4164	0.053	7.858	0.000	0.313	0.520
prior	-0.8543	0.061	-13.984	0.000	-0.974	-0.735
gender	0.3835	0.068	5.651	0.000	0.251	0.517

	coef	std err	z	P	[0.025	0.975]
juvenilecrime	-0.8646	0.084	-10.238	0.000	-1.030	-0.699
currentcharge	-0.2544	0.056	-4.562	0.000	-0.364	-0.145

2.1.4 Generate nearly optimal models

Nearly optimal logistic regression models are defined as models with logistic loss less than $(1+\varepsilon)$ times the minimum loss (i.e., logistic loss of the optimal model). Default value for ε is 5%.

- `u1` and `u2` are key hyper-parameters for generating nearly optimal models, which control the sampling range of initial models to fully explore the model space.
- Use the following command to generate a set of reasonable values for `u1` and `u2` (using `m=200` initial models), such that approximately 70%-80% of initial models are eligible:

```
u1, u2 = model_object.init_hyper_params(m=200)
(u1, u2)
```

```
(0.5, 80.3125)
```

- Use the following command to generate a final set of nearly optimal models (e.g., `n_final=250`) from 500 initial samples (`m=500`):

```
model_object.draw_models(u1=u1, u2=u2, m=500, n_final=250, random_state=1234)
model_object.models_plot
```

```
model_object.models_near_optim.iloc[:5]
```

const	age_1	race_1	prior_1	gender_1	juvenilecrime_1	currentcharge_1	perf_metric
-0.2307	3.1195	0.5047	-1.1409	0.2644	-0.1170	0.2664	1.0280
0.5503	0.7759	0.8971	-1.1164	-0.3083	-0.6398	-0.1481	1.0285
0.1068	0.8697	-0.0176	-0.6963	0.6987	-0.5041	-0.1812	1.0187
0.9715	0.8669	-0.1101	-1.0772	0.6450	-1.3590	-0.3310	1.0212
-1.0476	2.0026	0.6911	-0.3203	1.4661	-0.6633	-0.0397	1.0438
0.4006	1.6629	0.1719	-0.5450	0.3218	-0.9498	0.6260	1.0445

2.1.5 Assess variable importance

This step assesses variable importance for each nearly optimal model generated in the previous step using the SAGE method, and write the results to the output folder for further processing in the subsequent R workflow. Parallel processing is used to reduce run time.

- `model_object`: the model object created above.
- `x_expl`, `y_expl`: predictors (as a data frame) and outcome from the explanation set.
- `n_cores`: number of CPU cores to use in parallel processing.
  - For a computer with `n` cores, do not use more than `n-1` cores.
- `threshold`: threshold parameter used in SAGE algorithm for convergence criterion. A reasonable value is 0.05 (default).
  - Smaller `threshold` value may improve accuracy of uncertainty measure but notably increases run time.

```
from ShapleyVIC import compute
m_svic = compute.compute_shapley_vic(
    model_obj=model_object,
    x_expl=dat_expl.drop(columns=[y_name]), y_expl=dat_expl[y_name],
    n_cores=7, # running on a MacBook Air with 8 cores
    threshold=0.05
)
```

> **Note**
>
> - *Use built-in software (e.g., Activity Monitor/Task Manager) to monitor CPU and Memory usage. Avoid taking up 100% CPU, which can slow down computation.*
> - *This step can be time consuming with larger number of variables and/or larger explanation data.*
> - *For users' reference, the command above took approximately 11 minutes on a 2022 MacBook Air (Apple M2 chip with 8-core CPU, 8-core GPU; 16GB unified memory; 256GB SSD storage).*

2.2 [R] ShapleyVIC summary and visualizations

This part of the ShapleyVIC workflow is implemented in R.

This part of the workflow works on output from Python (all saved in `output_dir`), pooling information across models to compute (and visualize) overall variable importance and derive ensemble variable rankings.

2.2.1 Compute overall importance

As detailed in the paper, raw Shapley-based variable importance needs to be adjusted based on variable colinearity to derive final ShapleyVIC values.

- `output_dir`: output folder generated from the Python workflow.
- `outcome_type`: type of outcome, as specified in the Python workflow.
- `x` and `y`: training data specified in the Python workflow, required if `save_data=False` was specified when setting up `model.models(...)` in Python.
- `x_names_cat`: names of categorical variables, as specified in the Python workflow. Used when assessing variable colinearity from the training set. Optional for binary variables coded as 0/1.
- `x_names_display`: variable names to use in summary statistics and visualizations. If not provided, column names in the training set will be used.

2.2.1.1 Data saved in Python workflow

```
library(ShapleyVIC)
model_object <- compile_shapley_vic(
  output_dir = "compas_output", outcome_type = "binary",
  x_names_cat = c('age','race','prior','gender','juvenilecrime','currentcharge'),
  x_names = c("Age", "Race", "Prior criminal history", "Gender",
              "Juvenile criminal history", "Current charge")
)
```

2.2.1.2 Data not saved in Python workflow

When training data was not saved in the Python workflow, they must be supplied as input (`x` and `y`) in this step. See Chapter 3 for another example.

```
# Prepare training data the same way as in Python workflow:
library(ShapleyVIC)
suppressPackageStartupMessages(library(dplyr))
data("df_compas")
# Use the `train_test` indicator to filter out training data used in Python workflow:
df_train <- df_compas %>% filter(train_test == "train") %>%
  select(-train_test) %>% as.data.frame()
y_name <- "y"
x_names <- setdiff(names(df_train), y_name)
# Supply x and y when compiling results from Python workflow:
```

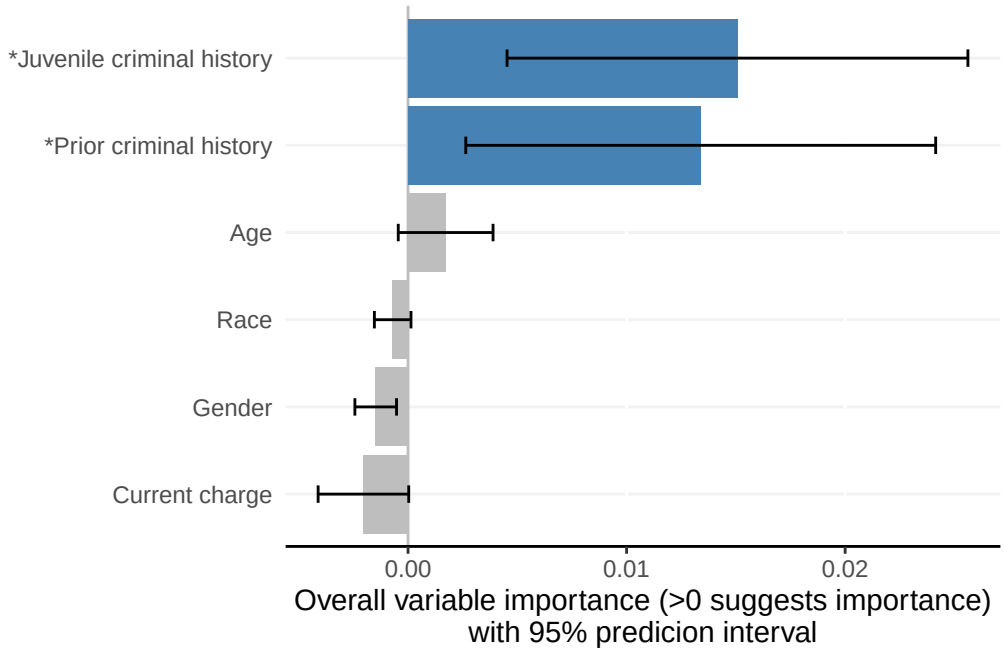
```
model_object <- compile_shapley_vic(
  output_dir = "compas_output", outcome_type = "binary",
  x = df_train[, x_names], y = df_train[, y_name],
  x_names_cat = c('age','race','prior','gender','juvenilecrime','currentcharge'),
  x_names = c("Age", "Race", "Prior criminal history", "Gender",
              "Juvenile criminal history", "Current charge")
)
```

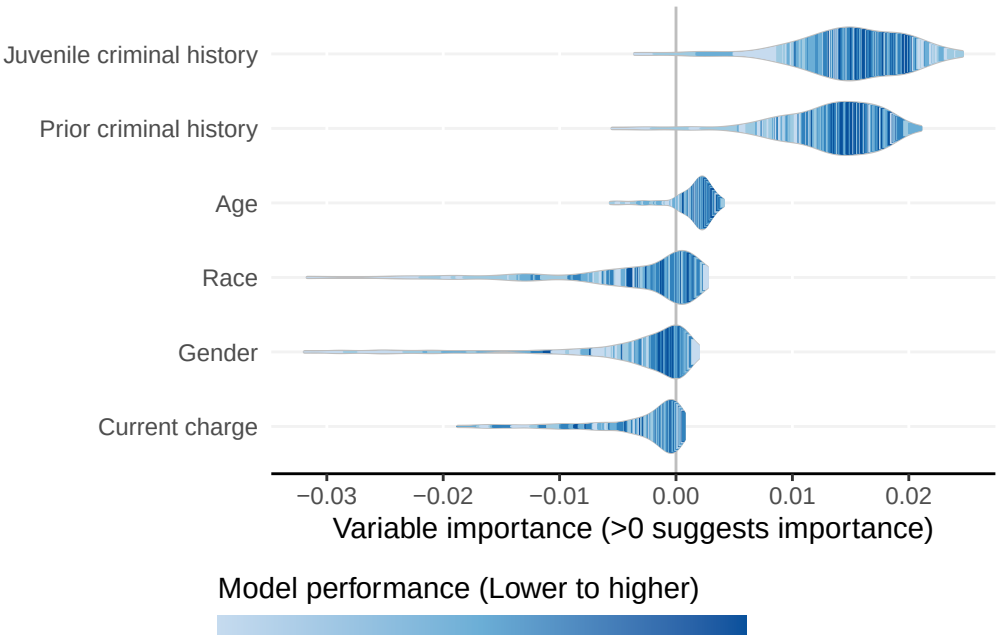
2.2.2 Visualize overall variable importance

Each ShapleyVIC value (`shapley_vic_val`) is reported with a standard deviation (`sage_sd`). We pool information across models to compute overall variable importance and uncertainty interval, visualized using bar plot. The relationship between variable importance and model performance is visualized using violin plot.

- For clarity, in the bar plot variables with significant overall importance are indicated by blue color and "*" next to variable names.

```
model_plots <- plot(model_object)
```





> **Note**
>
> - *Plots above reproduce key findings reported in the paper: race had non-significant overall importance, and prior criminal history and juvenile criminal history had higher overall importance than other variables.*
> - *Overall importance of age now becomes non-significant, showing that data sparsity (only 20 [2.8%] of 721 subjects had age=1 in explanation data) leads to less stable results.*

The bar plot can be further edited using ggplot functions, e.g., edit text font size using `theme()` or add plot title using `labs()`:

```
library(ggplot2)
model_plots$bar + theme(text = element_text(size = 14)) + labs(title = "Bar plot")
```

To apply similar formatting to the violin plot, use the following function:

```
library(ggplot2)
plot_violin(x = model_object, title = "Violin plot",
            plot_theme = theme(text = element_text(size = 14)))
```

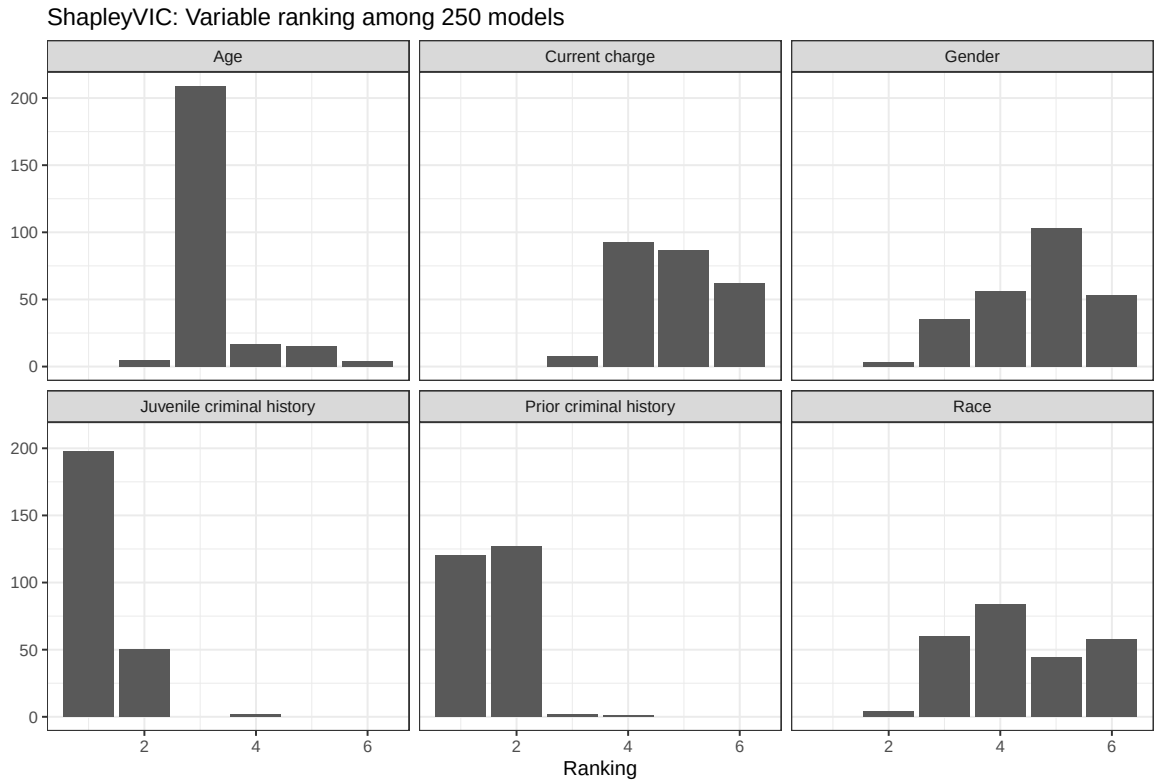
2.2.3 Ensemble variable ranking

ShapleyVIC values can also be used to rank variables by their importance to each model. The bar plot of ranks may help identify models with increased reliance on specific variable of interest for further investigation.

```
val_ranks <- rank_variables(model_object)
head(val_ranks, 6)
```

```
  model_id                  Variable rank
1        0                       Age    3
2        0                      Race    3
3        0    Prior criminal history    1
4        0                    Gender    3
5        0 Juvenile criminal history    2
6        0            Current charge    6
```

```
library(ggplot2)
ggplot(val_ranks, aes(x = rank, group = Variable)) +
  geom_bar() +
  facet_wrap(~ Variable, nrow = 2) +
  theme_bw() +
  labs(x = "Ranking", y = "",
       title = "ShapleyVIC: Variable ranking among 250 models")
```



The ensemble ranking averages the ranks across models, and can be used to guide downstream model building, e.g., using AutoScore. See the next chapter for detailed demonstration.

```
rank_variables(model_object, summarise = TRUE)
```

```
                  Variable mean_rank
1 Juvenile criminal history     1.224
2    Prior criminal history     1.536
```

```
# To return variable ranking as named vector for convenient integration with
# AutoScore:
rank_variables(model_object, summarise = TRUE, as_vector = TRUE)
```

```
Juvenile criminal history    Prior criminal history
                    1.224                     1.536
```

3 AutoScore-ShapleyVIC for Interpretable Risk Score Development

Risk scores are widely used for clinical decision making and commonly generated from logistic regression models. Machine-learning-based methods may work well for identifying important predictors to create parsimonious scores, but such 'black box' variable selection limits interpretability, and variable importance evaluated from a single model can be biased. We propose a robust and interpretable variable selection approach using ShapleyVIC, and integrate it with the AutoScore framework for convenient development of risk scoring models.

In this chapter, we describe the application of the AutoScore-ShapleyVIC workflow using an empirical example in our paper, and provide code for generating a risk score (i.e., Model 2 in the paper) to predict the risk of 30-day readmission or death from 41 candidate variables.

In the next chapter, we provide a fully reproducible example to demonstrate the use of the AutoScore-ShapleyVIC workflow using a simulated data that is publicly available.

Cite the following papers for AutoScore-ShapleyVIC:

- Ning Y, Ong ME, Chakraborty B, Goldstein BA, Ting DS, Vaughan R, Liu N. Shapley variable importance cloud for interpretable machine learning. *Patterns* 2022
- Ning Y, Li S, Ong ME, Xie F, Chakraborty B, Ting DS, Liu N. A novel interpretable machine learning system to generate clinical risk scores: An application for predicting early mortality or unplanned readmission in a retrospective cohort study. *PLOS Digit Health* 1(6): e0000062.
- Xie F, Chakraborty B, Ong MEH, Goldstein BA, Liu N. AutoScore: A machine learning-based automatic clinical score generator and its application to mortality prediction using electronic health records. *JMIR Medical Informatics* 2020; 8(10): e21798.

3.1 [R] Prepare data

This part of the workflow is implemented in R.

3.1.1 Load R packages and data

```
if (!require(AutoScore, quietly = TRUE)) install.packages("AutoScore")
library(AutoScore)
library(tidyverse) # For convenient data manipulation and visualization

# Read the final clean data with 41 candidate variables and the binary outcome
# (`label`):
dat <- readRDS("dat_readmit_or_death.RDS")
```

3.1.2 Prepare training, validation and test datasets

- Use the `split_data()` function of the AutoScore package to split data into training (70%), validation (10%) and test (20%) sets for risk score development.
- Perform median imputation for vital signs and lab tests based on training set.

> **!** Important
>
> - *As detailed in Chapter 1, handle missingness (and any other potential data issue) before applying ShapleyVIC.*

```
set.seed(1234)
Out_split <- split_data(data = dat, ratio = c(7, 1, 2))
# Median imputation for vital signs and lab tests based on training set:
train_lab_test <- Out_split$train_set %>% select(Pulse:SODIUM)
train_lab_test_median <- apply(train_lab_test, 2, function(x) median(x, na.rm = TRUE))
Out_split <- lapply(Out_split, function(dat) {
  for (nm in names(train_lab_test)) {
    dat[, nm] <- ifelse(is.na(dat[, nm]), train_lab_test_median[nm], dat[, nm])
  }
  dat
})

train_set <- Out_split$train_set
validation_set <- Out_split$validation_set
test_set <- Out_split$test_set
```

- Prepare `output_dir` for ShapleyVIC, using `train_set` as training set and the first 3500 observations in `validation_set` as the explanation data.

```r
output_dir <- "score_output"
if (!dir.exists(output_dir)) dir.create(output_dir)
write.csv(train_set, file = file.path(output_dir, "train_set.csv"),
          row.names = FALSE)
write.csv(validation_set[1:3500, ],
          file = file.path(output_dir, "validation_set.csv"),
          row.names = FALSE)
```

3.2 [Python] Compute ShapleyVIC values

This part of the workflow is implemented in Python.

- Load data and set up input information.
- Data used in this analysis is sensitive, therefore we do not save training data to the output folder to avoid any potential data security issue.

```python
import os
import pandas as pd
output_dir = "score_output"
dat_train = pd.read_csv(os.path.join(output_dir, 'train_set.csv'))
dat_expl = pd.read_csv(os.path.join(output_dir, 'validation_set.csv'))

y_name = 'label'
x_names_cat = ['Gender','Race','Triage_Class_Code','DayofWeek','MI','CHF','PVD',
    'Stroke','Dementia','Pulmonary','Rheumatic','PUD','LiverMild','Diabetes',
    'DMcx','Paralysis','Renal','Cancer','LiverSevere','Mets','admit_cat',
    'resuscitation','VENTILATION']
from ShapleyVIC import model
model_object = model.models(
    x=dat_train.drop(columns=[y_name]), y=dat_train[y_name],
    x_names_cat=x_names_cat, outcome_type="binary", output_dir=output_dir,
    save_data=False
)
```

- Draw 350 nearly optimal models.

```python
model_object.draw_models(u1=0.2, u2=300, m=800, n_final=350)
```

- Compute ShapleyVIC values.

```
from ShapleyVIC import compute
m_svic = compute.compute_shapley_vic(
    model_obj=model_object,
    x_expl=dat_expl.drop(columns=[y_name]), y_expl=dat_expl[y_name],
    n_cores=10, # running on a PC with 40 logical processors
    threshold=0.025
)
```

3.3 [R] Develop risk score

This part of the workflow is implemented in R.

3.3.1 Rank variables using ShapleyVIC

- Compile ShapleyVIC output.
- Since data was not saved in the Python workflow, we explicitly specify it in the R analysis.
- Explicitly specify names of categorical variables, identical to those specified in the Python workflow.

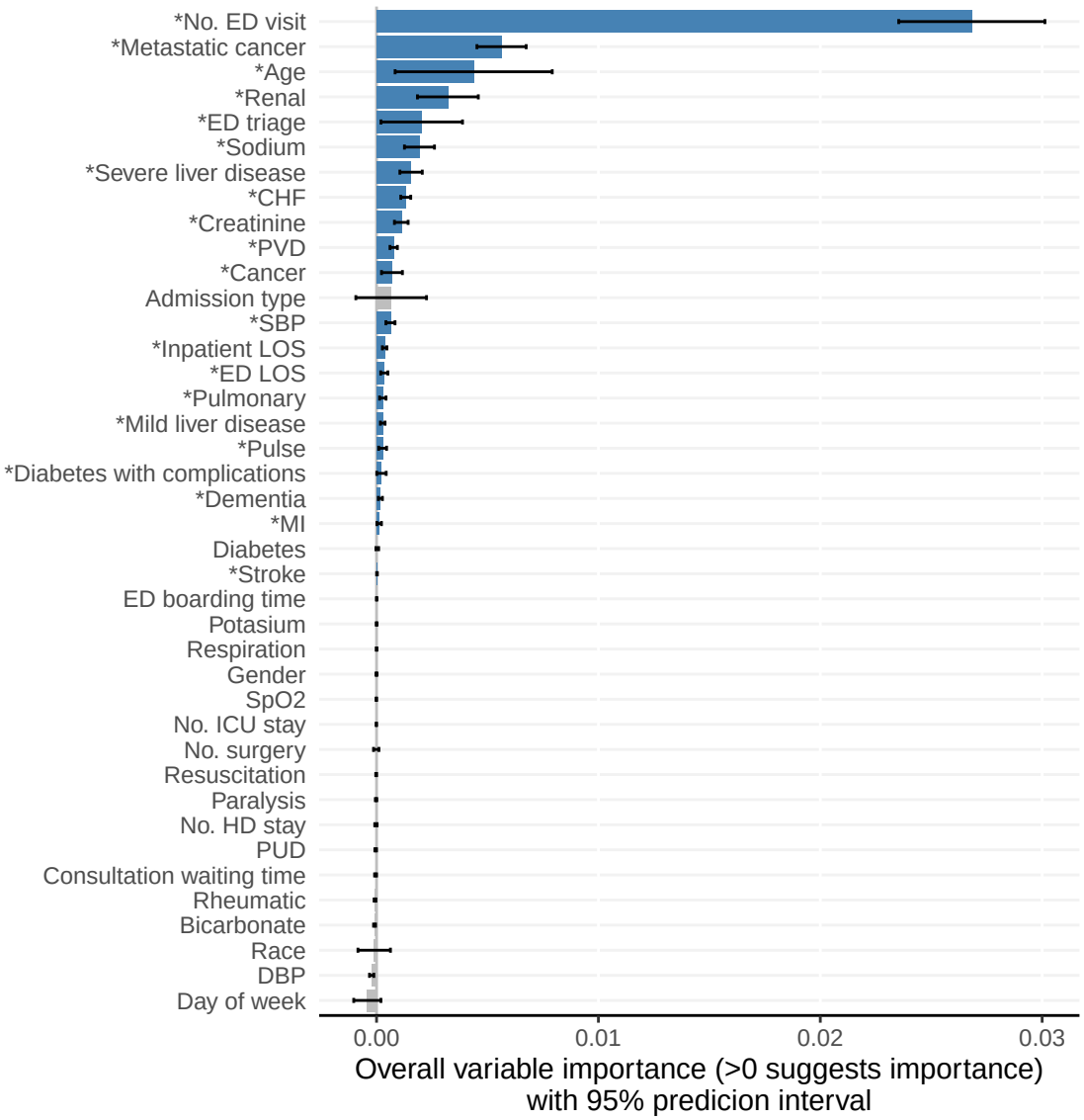
```
output_dir <- "score_output"
x_names_display <- c(
  "Age", "Gender", "Race", "ED LOS", "ED triage",
  "ED boarding time", "Consultation waiting time", "No. ED visit",
  "Day of week", "Inpatient LOS", "Ventilation", "Resuscitation",
  "No. surgery", "No. ICU stay",
  "No. HD stay", "Pulse", "Respiration", "SpO2",
  "DBP", "SBP", "Bicarbonate", "Creatinine",
  "Potasium", "Sodium", "MI", "CHF", "PVD", "Stroke",
  "Dementia", "Pulmonary", "Rheumatic", "PUD", "Mild liver disease",
  "Diabetes", "Diabetes with complications", "Paralysis", "Renal", "Cancer",
  "Severe liver disease", "Metastatic cancer", "Admission type"
)
y_name <- "label"
x_names <- setdiff(names(train_set), y_name)
library(ShapleyVIC)
model_object <- compile_shapley_vic(
  output_dir = output_dir, outcome_type = "binary",
  x = train_set[, x_names], y = train_set$label,
```

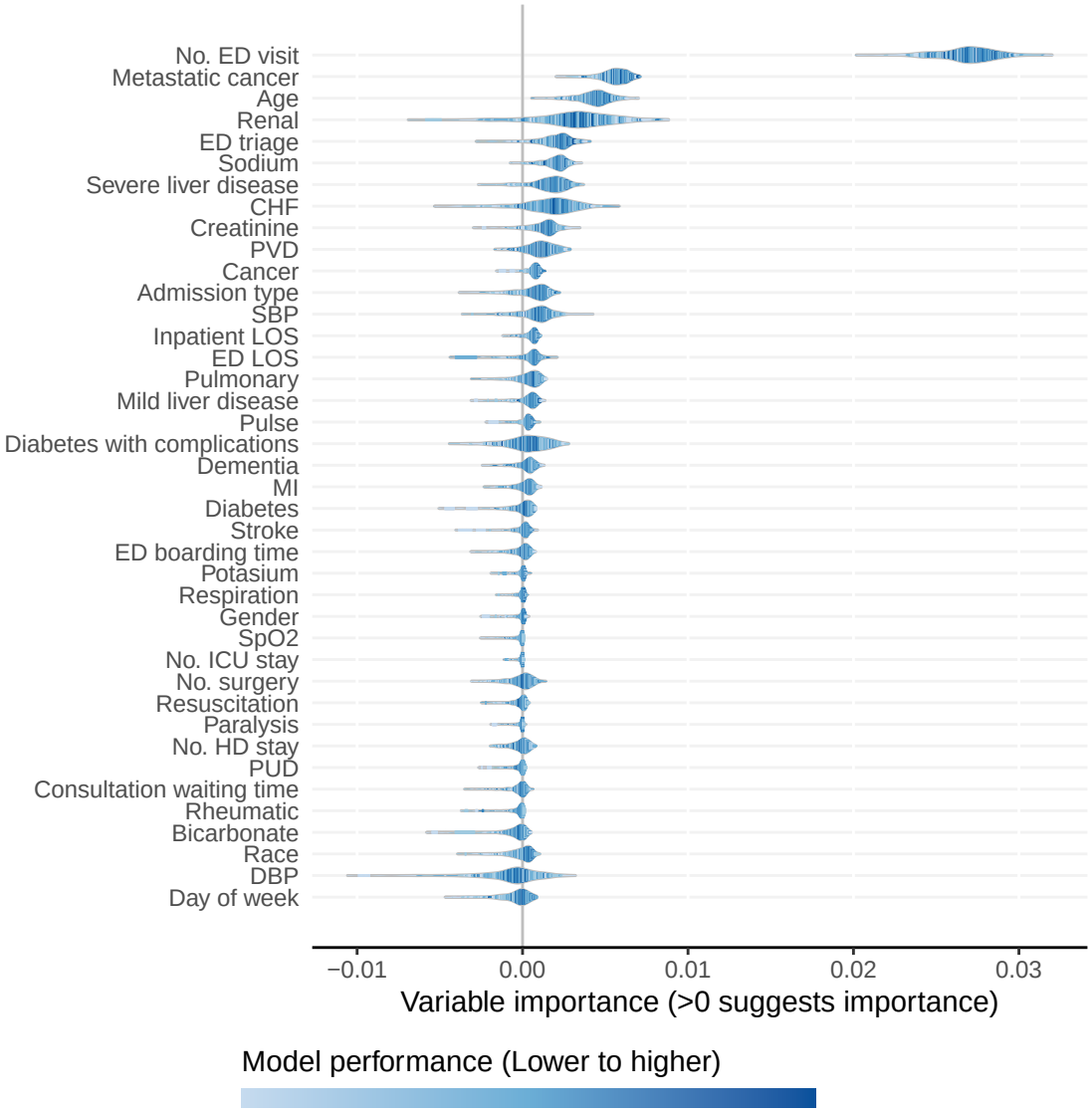
```
  x_names_cat = c(
    'Gender','Race','Triage_Class_Code','DayofWeek','MI','CHF','PVD',
    'Stroke','Dementia','Pulmonary','Rheumatic','PUD','LiverMild','Diabetes',
    'DMcx','Paralysis','Renal','Cancer','LiverSevere','Mets','admit_cat',
    'resuscitation','VENTILATION'
  ),
  x_names = x_names_display
)
```

- Visualize ShapleyVIC values for overall variable importance.

```
model_plots <- plot(model_object)
```

```
The following variables are excluded due to zero importance in all models analysed:

     Ventilation
```


*No. ED visit
*Metastatic cancer
*Age
*Renal
*ED triage
*Sodium
*Severe liver disease
*CHF
*Creatinine
*PVD
*Cancer
Admission type
*SBP
*Inpatient LOS
*ED LOS
*Pulmonary
*Mild liver disease
*Pulse
*Diabetes with complications
*Dementia
*MI
Diabetes
*Stroke
ED boarding time
Potasium
Respiration
Gender
SpO2
No. ICU stay
No. surgery
Resuscitation
Paralysis
No. HD stay
PUD
Consultation waiting time
Rheumatic
Bicarbonate
Race
DBP
Day of week
0.00
0.01
0.02
0.03
Overall variable importance (>0 suggests importance)
with 95% predicion interval



- Derive ShapleyVIC-based ensemble variable ranking.

```
ranking <- rank_variables(model_object, summarise = TRUE, as_vector = TRUE)
ranking
```

```
      No. ED visit  Metastatic cancer
          1.000000           2.182857
              Age             Sodium
          3.057143           6.931429
```

```
                Renal                   ED triage
             6.945714                    7.428571
Severe liver disease                         CHF
             9.071429                   10.365714
          Creatinine                         PVD
            10.642857                   12.074286
                 SBP                      Cancer
            14.120000                   14.597143
       Inpatient LOS                      ED LOS
            16.105714                   16.908571
  Mild liver disease                    Pulmonary
            17.785714                   18.040000
             Dementia Diabetes with complications
            19.697143                   20.291429
                Pulse                          MI
            20.691429                   21.214286
               Stroke
            24.185714
```

3.3.2 Develop risk score using AutoScore workflow

- Modify variable names in training, validation and test sets for publication-ready figures and printed output.

```
# Current raw variable names:
names(train_set)
```

```
 [1] "label"                    "Age"
 [3] "Gender"                   "Race"
 [5] "ED_LOS"                   "Triage_Class_Code"
 [7] "EDBoardingTime"           "ConsultationWaitingTime"
 [9] "n_ed_6mth"                "DayofWeek"
[11] "LOS_inp"                  "VENTILATION"
[13] "resuscitation"            "Total_Num_Surgery_last1yr"
[15] "Total_icu_count_last1yr"  "Total_hd_count_last1yr"
[17] "Pulse"                    "Respiration"
[19] "SPO2"                     "BP_Diastolic"
[21] "BP_Systolic"              "BICARBONATE"
[23] "CREATININE"               "POTASSIUM"
[25] "SODIUM"                   "MI"
[27] "CHF"                      "PVD"
```

```
[29] "Stroke"                    "Dementia"
[31] "Pulmonary"                 "Rheumatic"
[33] "PUD"                       "LiverMild"
[35] "Diabetes"                  "DMcx"
[37] "Paralysis"                 "Renal"
[39] "Cancer"                    "LiverSevere"
[41] "Mets"                      "admit_cat"
```

```r
# Modified variable names:
names(train_set)[-1] <- x_names_display
names(train_set)
```

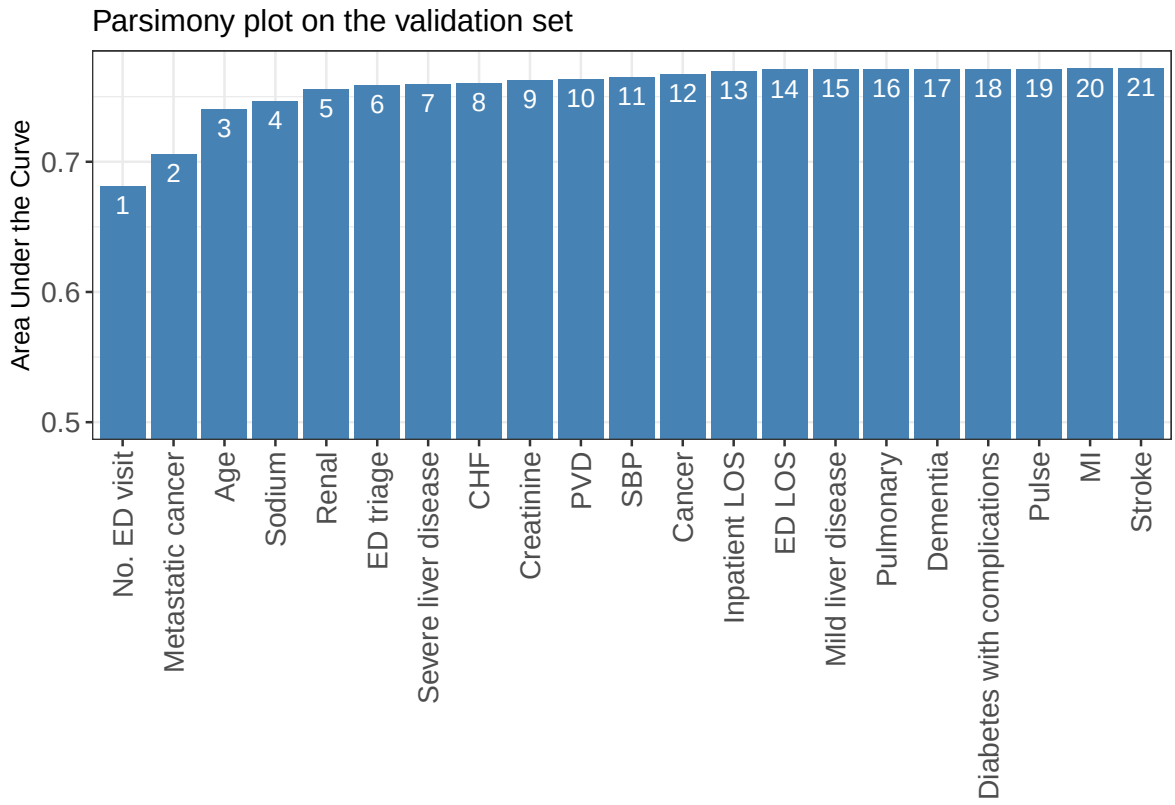
```
 [1] "label"                      "Age"
 [3] "Gender"                     "Race"
 [5] "ED LOS"                     "ED triage"
 [7] "ED boarding time"           "Consultation waiting time"
 [9] "No. ED visit"               "Day of week"
[11] "Inpatient LOS"              "Ventilation"
[13] "Resuscitation"              "No. surgery"
[15] "No. ICU stay"               "No. HD stay"
[17] "Pulse"                      "Respiration"
[19] "SpO2"                       "DBP"
[21] "SBP"                        "Bicarbonate"
[23] "Creatinine"                 "Potasium"
[25] "Sodium"                     "MI"
[27] "CHF"                        "PVD"
[29] "Stroke"                     "Dementia"
[31] "Pulmonary"                  "Rheumatic"
[33] "PUD"                        "Mild liver disease"
[35] "Diabetes"                   "Diabetes with complications"
[37] "Paralysis"                  "Renal"
[39] "Cancer"                     "Severe liver disease"
[41] "Metastatic cancer"          "Admission type"
```

```r
names(validation_set)[-1] <- x_names_display
names(test_set)[-1] <- x_names_display
```

- Based on the ensemble variable ranking, apply AutoScore STEP(ii) to select the best model with parsimony plot.

```r
AUC <- AutoScore_parsimony(
  train_set = train_set, validation_set = validation_set,
  rank = ranking, max_score = 100, n_min = 1, n_max = length(ranking)
)
```

```
Select 1 Variable(s):  Area under the curve: 0.6811
Select 2 Variable(s):  Area under the curve: 0.706
Select 3 Variable(s):  Area under the curve: 0.7406
Select 4 Variable(s):  Area under the curve: 0.7467
Select 5 Variable(s):  Area under the curve: 0.7555
Select 6 Variable(s):  Area under the curve: 0.7589
Select 7 Variable(s):  Area under the curve: 0.7595
Select 8 Variable(s):  Area under the curve: 0.7605
Select 9 Variable(s):  Area under the curve: 0.7624
Select 10 Variable(s):  Area under the curve: 0.7637
Select 11 Variable(s):  Area under the curve: 0.765
Select 12 Variable(s):  Area under the curve: 0.7674
Select 13 Variable(s):  Area under the curve: 0.7696
Select 14 Variable(s):  Area under the curve: 0.7708
Select 15 Variable(s):  Area under the curve: 0.7708
Select 16 Variable(s):  Area under the curve: 0.7713
Select 17 Variable(s):  Area under the curve: 0.7713
Select 18 Variable(s):  Area under the curve: 0.7712
Select 19 Variable(s):  Area under the curve: 0.7713
Select 20 Variable(s):  Area under the curve: 0.7715
Select 21 Variable(s):  Area under the curve: 0.7718
```

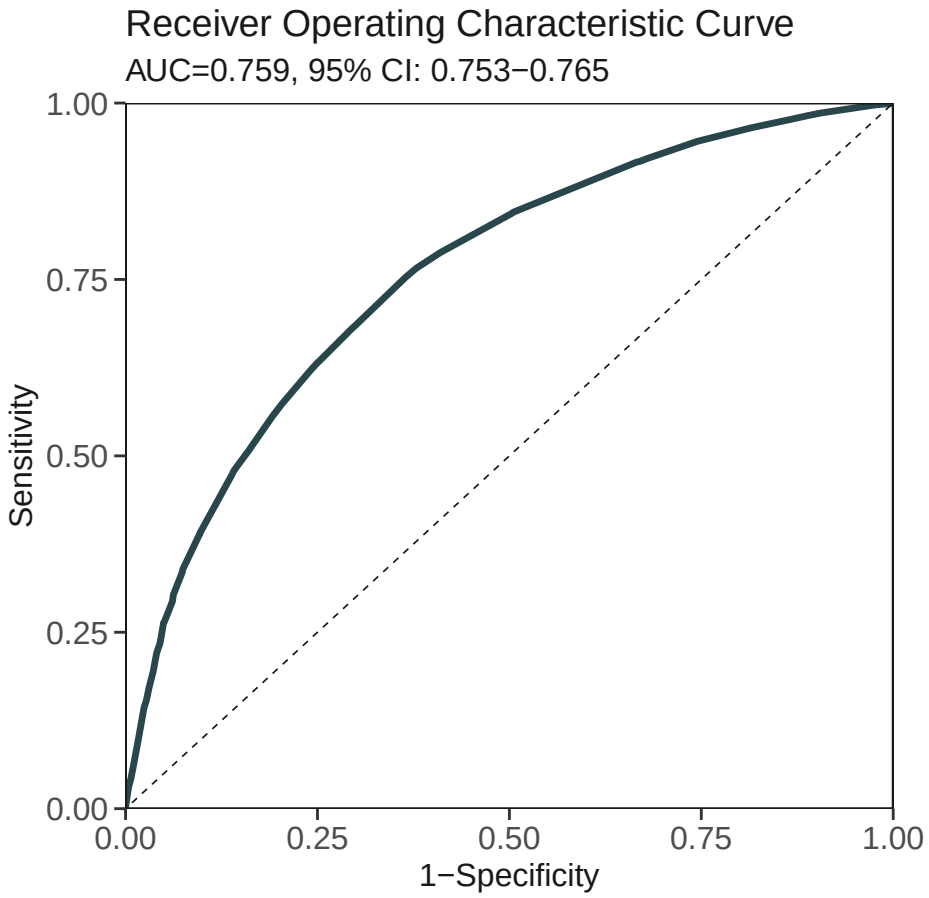


- This parsimony is somewhat smoother than that from random forest-based variable ranking used in AutoScore.
- A feasible choice is to select the top 6 variables, as adding additional variables does not substantially improve model performance.
- Apply AutoScore STEP(iii) to build initial scores from the top 6 variables.

```
cut_vec <- AutoScore_weighting(
  train_set = train_set, validation_set = validation_set,
  final_variables = names(ranking)[1:6], max_score = 100
)
```

```
****Included Variables:
      variable_name
1      No. ED visit
2 Metastatic cancer
3               Age
4            Sodium
```

```
5             Renal
6         ED triage
****Initial Scores:


=================  =========  =====
variable           interval   point
=================  =========  =====
No. ED visit       <1           0
                   [1,3)       14
                   >=3         32

Metastatic cancer  0            0
                   1           22

Age                <28          0
                   [28,46)      5
                   [46,78)     11
                   [78,87)     14
                   >=87        19

Sodium             <126        11
                   [126,132)    8
                   [132,138)    3
                   [138,141)    0
                   >=141        3

Renal              0            0
                   1            8

ED triage          P1           8
                   P2           5
                   P3 and P4    0
=================  =========  =====
```




```
***Performance (based on validation set):
AUC:  0.7589   95% CI: 0.7525-0.7654 (DeLong)
Best score threshold: >= 27
Other performance indicators based on this score threshold:
Sensitivity: 0.7546
Specificity: 0.6338
PPV:         0.2888
NPV:         0.9291
***The cutoffs of each variable generated by the AutoScore are saved in cut_vec. You can dec
```

- Users can apply additional AutoScore STEPs for subsequent model fine-tuning and evaluation.

4 AutoScore-ShapleyVIC Reproducible Example

This chapter provides a fully reproducible example to demonstrate in detail the use of the AutoScore-ShapleyVIC workflow, using a simulated dataset with binary outcome available from the AutoScore package. The data is described in detail in the AutoScore Guidebook.

4.1 [R] Prepare data

This part of the workflow is implemented in R.

4.1.1 Load data

- Load `sample_data` from the AutoScore package.
- As required by AutoScore, change the name of outcome variable to `label`.
- Read AutoScore Guidebook for detailed data requirement.

```
library(AutoScore)
data("sample_data")
names(sample_data)[names(sample_data) == "Mortality_inpatient"] <- "label"
check_data(sample_data)
```

```
Data type check passed.
```

```
No NA in data
```

```
dim(sample_data)
```

```
[1] 20000    22
```

- As required by ShapleyVIC, code the binary outcome as 0/1.
- All variables are continuous.

```r
sample_data$label <- as.numeric(sample_data$label == "TRUE")
head(sample_data)
```

```
  Vital_A Vital_B Vital_C Vital_D Vital_E Vital_F Vital_G Lab_A Lab_B Lab_C
1      87     143      78     101      13    35.7      99   160  13.0    23
2      43     133      64      83      20    36.1      95   116  15.3    24
3      80     115      48      72      23    37.4      99   133   8.0    27
4     106     121      68      84      16    37.6      99   206  12.1    25
5      86     135      70      83      24    37.2      96   100  18.1    26
6      69     123      72      88      16    36.5      95   204  19.9    20
  Lab_D Lab_E Lab_F Lab_G Lab_H Lab_I Lab_J Lab_K Lab_L Lab_M Age label
1   0.0   105    34    12   0.8    98   4.4     0   136    16  66     0
2   0.8   108    36    12   0.6   322   4.3    55   141    17  79     0
3   1.3   111    30    11   2.9     0   4.4    40   142     0  86     0
4   0.0   102    39    14   3.0   214   4.4     0   134     6  69     0
5   2.3    96    36    13   2.7   326   3.8    20   134    26  65     0
6   2.5   101    31    10   0.8   103   4.2    38   138    14  68     0
```

4.1.2 Prepare training, validation, and test datasets

- Given large sample size (n=20000), split the data into training (70%), validation (10%) and test (20%) sets for risk score development.

```r
set.seed(4)
out_split <- split_data(data = sample_data, ratio = c(0.7, 0.1, 0.2))
train_set <- out_split$train_set
dim(train_set)
```

```
[1] 14000    22
```

```r
validation_set <- out_split$validation_set
dim(validation_set)
```

```
[1] 2000   22
```

```r
test_set <- out_split$test_set
dim(test_set)
```

```
[1] 4000   22
```

- Prepare `output_dir` for ShapleyVIC, using `train_set` as training set and `validation_set` as the explanation data.

> **!** Important
>
> - *As detailed in Chapter 1, check for and handle data issues before applying ShapleyVIC.*
> - *This demo uses data as-is because it is simulated clean data.*

```r
output_dir <- "mort_output"
if (!dir.exists(output_dir)) dir.create(output_dir)
write.csv(train_set, file = file.path(output_dir, "train_set.csv"),
          row.names = FALSE)
write.csv(validation_set,
          file = file.path(output_dir, "validation_set.csv"),
          row.names = FALSE)
```

4.2 [Python] Compute ShapleyVIC values

This part of the workflow is implemented in Python.

- Load data and set up input information.

```python
import os
import pandas as pd
output_dir = "mort_output"
dat_train = pd.read_csv(os.path.join(output_dir, 'train_set.csv'))
dat_expl = pd.read_csv(os.path.join(output_dir, 'validation_set.csv'))

y_name = 'label'
from ShapleyVIC import model
model_object = model.models(
    x=dat_train.drop(columns=[y_name]), y=dat_train[y_name],
    # No need to specify x_names_cat because all variables are continuous
    outcome_type="binary", output_dir=output_dir
)
```

- Set values for hyper-parameters `u1` and `u2`.

```
u1, u2 = model_object.init_hyper_params(m=200)
(u1, u2)
```

```
(0.5, 15.625)
```

- Draw 250 nearly optimal models from 500 initial samples.

```
model_object.draw_models(u1=u1, u2=u2, m=500, n_final=250, random_state=1234)
model_object.models_plot
```

- Compute ShapleyVIC values.

```
from ShapleyVIC import compute
m_svic = compute.compute_shapley_vic(
    model_obj=model_object,
    x_expl=dat_expl.drop(columns=[y_name]), y_expl=dat_expl[y_name],
    n_cores=25, # running on a PC with 40 logical processors
    threshold=0.05
)
```

> **Note**
>
> - *For users' reference, the command above took approximately 17 hours on a PC (Windows 10 Education; Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz 2.19GHz (2 processors); 128GB RAM).*

4.3 [R] Develop risk score

This part of the workflow is implemented in R.

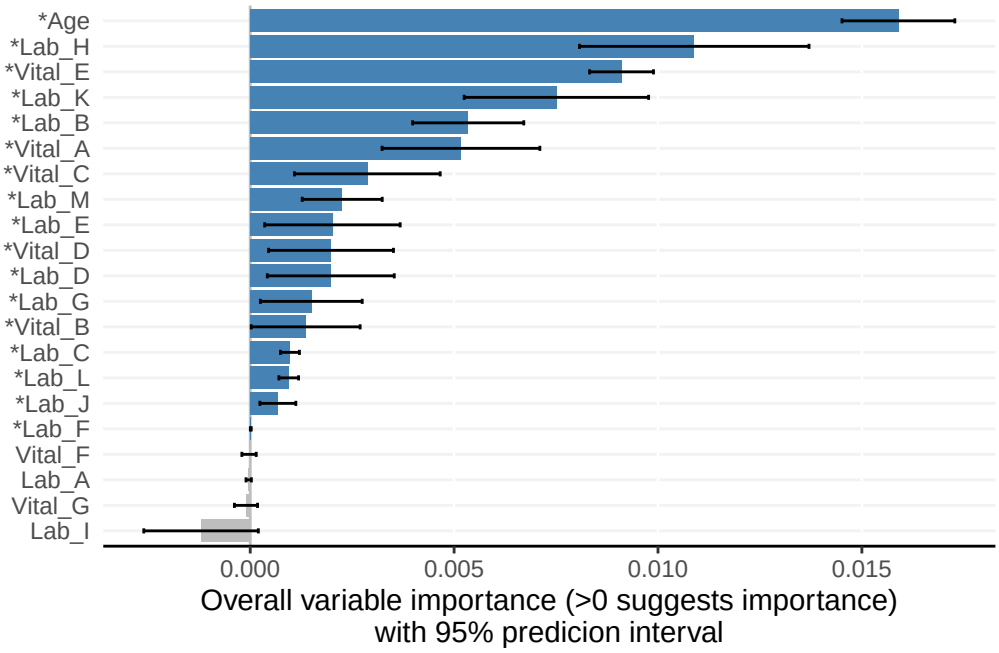
4.3.1 Rank variables using ShapleyVIC

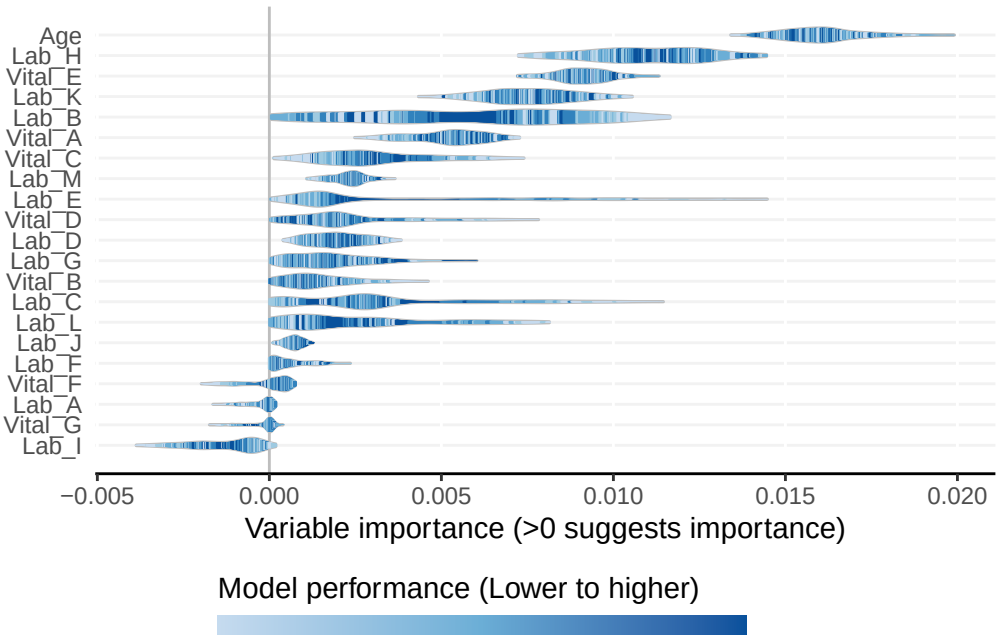
- Compile ShapleyVIC output.

```
library(ShapleyVIC)
model_object <- compile_shapley_vic(
  output_dir = output_dir, outcome_type = "binary"
)
```

- Visualize ShapleyVIC values for overall variable importance.

```
model_plots <- plot(model_object)
```





- Derive ShapleyVIC-based ensemble variable ranking.

```
ranking <- rank_variables(model_object, summarise = TRUE, as_vector = TRUE)
ranking
```

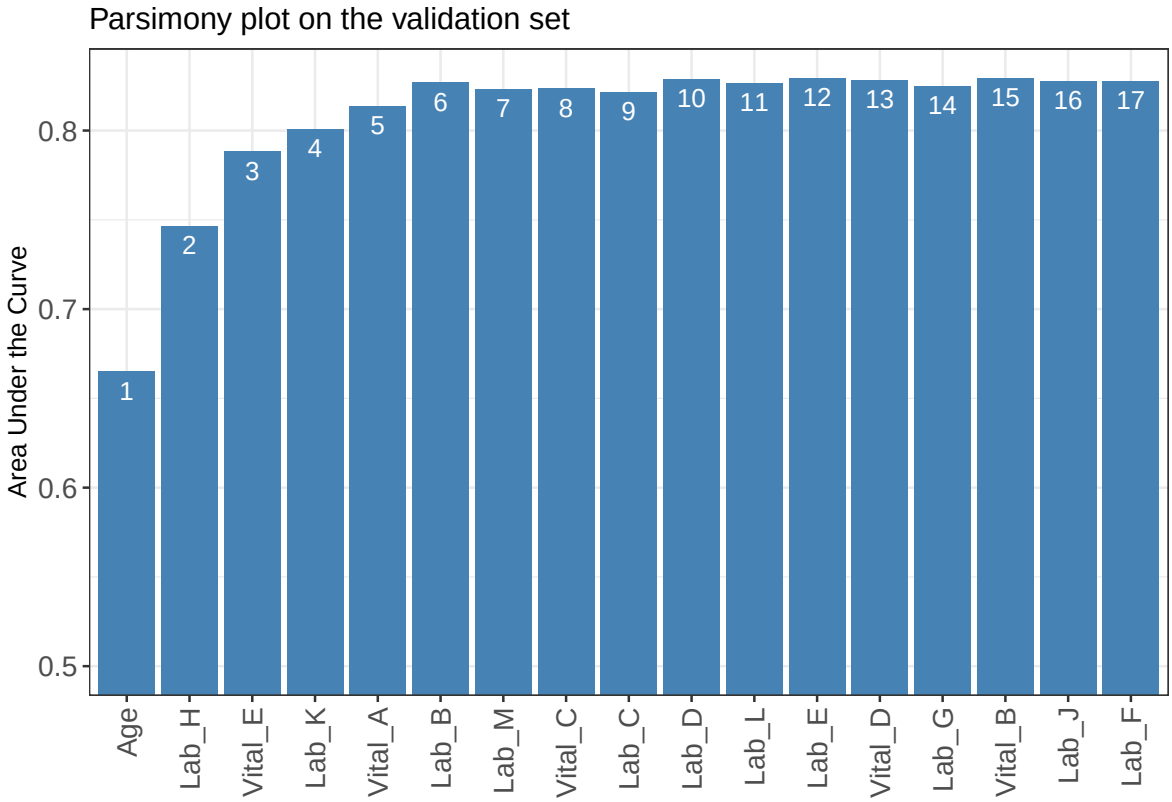
```
   Age   Lab_H Vital_E   Lab_K Vital_A   Lab_B   Lab_M Vital_C   Lab_C   Lab_D
 1.000   2.116   3.028   4.104   5.588   5.744   8.904   8.940   9.084   9.704
 Lab_L   Lab_E Vital_D   Lab_G Vital_B   Lab_J   Lab_F
10.120  10.268  11.628  12.952  13.220  13.328  16.052
```

4.3.2 Develop risk score using AutoScore workflow

- Based on the ensemble variable ranking, apply AutoScore STEP(ii) to select the best model with parsimony plot.

```
AUC <- AutoScore_parsimony(
  train_set = train_set, validation_set = validation_set,
  rank = ranking, max_score = 100, n_min = 1, n_max = length(ranking)
)
```

```
Select 1 Variable(s):  Area under the curve: 0.6649
Select 2 Variable(s):  Area under the curve: 0.7466
Select 3 Variable(s):  Area under the curve: 0.7881
Select 4 Variable(s):  Area under the curve: 0.8009
Select 5 Variable(s):  Area under the curve: 0.8137
Select 6 Variable(s):  Area under the curve: 0.8268
Select 7 Variable(s):  Area under the curve: 0.8232
Select 8 Variable(s):  Area under the curve: 0.8234
Select 9 Variable(s):  Area under the curve: 0.8215
Select 10 Variable(s):  Area under the curve: 0.8286
Select 11 Variable(s):  Area under the curve: 0.8267
Select 12 Variable(s):  Area under the curve: 0.8294
Select 13 Variable(s):  Area under the curve: 0.8281
Select 14 Variable(s):  Area under the curve: 0.8246
Select 15 Variable(s):  Area under the curve: 0.8293
Select 16 Variable(s):  Area under the curve: 0.8275
Select 17 Variable(s):  Area under the curve: 0.8278
```

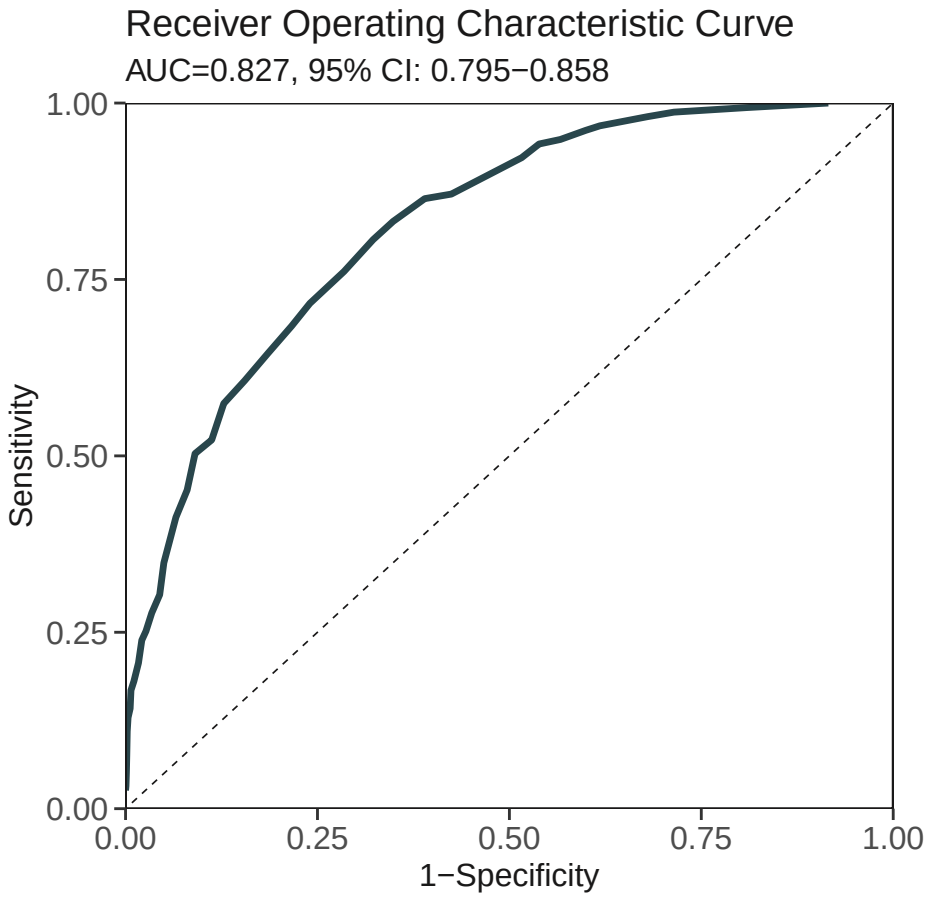


- This parsimony is somewhat smoother than that from random forest-based variable ranking used in AutoScore.
- A feasible choice is to select the top 6 variables, as adding additional variables does not substantially improve model performance.
- Apply AutoScore STEP(iii) to build initial scores from the top 6 variables.

```
cut_vec <- AutoScore_weighting(
  train_set = train_set, validation_set = validation_set,
  final_variables = names(ranking)[1:6], max_score = 100
)
```

```
****Included Variables:
  variable_name
1           Age
2         Lab_H
3       Vital_E
4         Lab_K
5       Vital_A
6         Lab_B
****Initial Scores:


========  ==========  =====
variable  interval    point
========  ==========  =====
Age       <35           0
          [35,49)       7
          [49,76)      17
          [76,89)      23
          >=89         27

Lab_H     <0.2          0
          [0.2,1.1)     4
          [1.1,3.1)     9
          [3.1,4)      15
          >=4          18

Vital_E   <12           0
          [12,15)       2
          [15,22)       7
          [22,25)      12
          >=25         15
```

```
Lab_K     <8            0
          [8,42)        6
          [42,58)      11
          >=58         14

Vital_A   <60           0
          [60,73)       1
          [73,98)       6
          [98,111)     10
          >=111        13

Lab_B     <8.5          0
          [8.5,11.2)    4
          [11.2,17)     7
          [17,19.8)    10
          >=19.8       12
========  ==========  =====
```




```
***Performance (based on validation set):
AUC:  0.8268   95% CI: 0.7953-0.8583 (DeLong)
Best score threshold: >= 57
Other performance indicators based on this score threshold:
Sensitivity: 0.8065
Specificity: 0.6775
PPV:         0.1736
NPV:         0.9766
***The cutoffs of each variable generated by the AutoScore are saved in cut_vec. You can dec
```

- Users can apply additional AutoScore STEPs for subsequent model fine-tuning and evaluation.