

midr : Black-Box モデルの解釈と保険実務への応用

midr パッケージをインストールする

midr パッケージのリリース版は CRAN からインストール可能です。

```
install.packages("midr") # 0.5.2
```

また、最新の開発版は GitHub からインストール可能です。

```
# install.packages("pak")
pak::pak("ryo-asashi/midr") # 0.5.1.900
```

パッケージのバージョンを確認しておきます。

```
packageVersion("midr")
```

```
[1] '0.5.1.900'
```

必要なパッケージを読み込む

このデモで利用するRパッケージを読み込みます。

```
library(ggplot2)    # 可視化
library(gridExtra) # プロットのレイアウト調整
library(dplyr)      # データ操作
library(ranger)    # ランダムフォレストの構築
library(midr)       # MIDモデルの構築
```

また、この資料のプロット用にテーマ設定を行います。

```
theme_set(theme_midr("y")) # プロットテーマ
set.color.theme(
  name = "taikai", type = "diverging",
  kernel = c("#004098", "#f5f5f5", "#D5001A"),
  kernel.args = list(mode = "ramp")
) # プロット用カラーテーマ
```

データを読み込む

このデモでは “Insurance Data for Homeowners and Motor Insurance Customers Monitored over Five Years” (以下、「スペインデータ」)を使用します。

スペインデータは、スペインの自動車・住宅保険の契約者40,284人に関する、2010年から2014年までのパネルデータです。契約番号などの21変数について、延べ122,935件年分の記録が含まれています。

```
# 変数のデータ型を指定する
dtypes <- c(gender = "factor", Car_2ndDriver_M = "factor",
            metro_code = "factor", Policy_PaymentMethodA = "factor",
            Policy_PaymentMethodH = "factor", apartment = "factor",
            Retention = "factor", Types = "factor")
# データを読み込む(行番号を示す1列目を除外)
path_data <- "data/data_ex.csv"
df_all <- read.csv(path_data, sep = ",", colClasses = dtypes)[-1]
```

データの概要を確認する(1/2)

スペインデータに含まれる21個の変数の概要是次のとおりです。

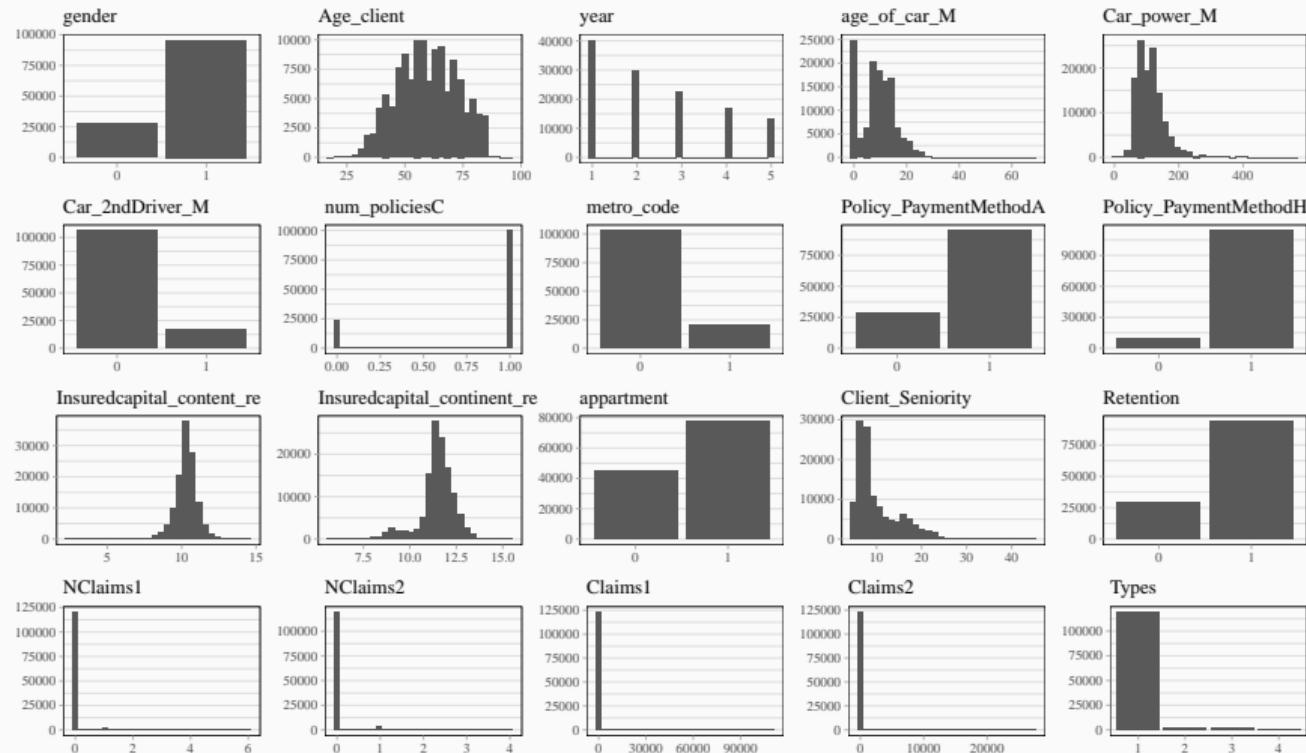
```
PolID # 契約者を一意に識別するための番号  
year # 観察年度(1から5で、2010年から2014年に対応)  
gender # 性別(1: 男性, 0: 女性)  
Age_client # 契約者の年齢  
age_of_car_M # 契約者が車を購入してからの年数  
Car_power_M # 車の馬力  
Car_2ndDriver_M # 第2の臨時ドライバーの申告有無(1: あり, 0: なし)  
num_policiesC # 同じ契約者が持つ保険契約の総数  
metro_code # 居住地域(1: 都市部, 0: 地方)  
Policy_PaymentMethodA # 自動車保険の保険料支払方法(1: 年払, 0: 月払)  
Policy_PaymentMethodH # 住宅保険の保険料支払方法(1: 年払, 0: 月払)
```

データの概要を確認する(2/2)

```
Insuredcapital_content_re    # 住宅保険における家財の保険価額(ユーロ)
Insuredcapital_continent_re # 住宅保険における建物の保険価額(ユーロ)
apartment          # 建物がアパートかどうか(1: アパート, 0: その他)
Client_Seniority   # 顧客としての契約継続年数
Retention          # 契約が更新されたかどうか(1: 更新された, 0: 更新されなかった)
NClaims1           # 自動車保険での年間クレーム(保険金請求)件数
NClaims2           # 住宅保険での年間クレーム(保険金請求)件数
Claims1            # 自動車保険での年間クレーム総額(ユーロ)
Claims2            # 住宅保険での年間クレーム総額(ユーロ)
Types              # クレーム発生の組み合わせ
                  # (1: 両方なし, 2: 自動車のみ, 3: 住宅のみ, 4: 両方あり)
```

データの分布を確認する

また、各変数の分布は次のようになっています。



データの形式を確認する

スペインデータは、契約番号ごとにその契約が更新停止される(Retention = 0)までの各観察年度に対応するレコードを保持しています。

	PolID	year	Retention	gender	Age_client	age_of_car_M	Car_power_M	metro_code
1	1	1	1	1	84	13	90	0
2	1	2	1	1	84	13	90	0
3	1	3	1	1	84	13	90	0
4	1	4	1	1	84	13	90	0
5	1	5	0	1	84	13	90	0

	PolID	year	Retention	gender	Age_client	age_of_car_M	Car_power_M	metro_code
1	2	1	1	1	83	0	177	0
2	2	2	0	1	83	0	177	0

	PolID	year	Retention	gender	Age_client	age_of_car_M	Car_power_M	metro_code
1	3	1	1	1	85	0	163	0
2	3	2	1	1	85	0	163	0
3	3	3	1	1	85	0	163	0
4	3	4	1	1	85	0	163	0
5	3	5	1	1	85	0	163	0

回帰タスクを設定する

このデモでは、第1観察年度(`year = 1`)のデータの2/3にあたる26,856件のデータを学習データとして、各契約の更新確率を予測するモデルを構築します。残った13,428件のデータはモデルの評価に利用します。

```
set.seed(42) # ランダムシードの設定
train_ids <- sample(seq_len(40284), 26856) # 契約番号の抽出
# 学習データを抽出する
df_train <- df_all |> filter(year == 1, PolID %in% train_ids)
nrow(df_train)
```

```
[1] 26856
```

```
# 検証データを抽出する
df_valid <- df_all |> filter(year == 1, !PolID %in% train_ids)
nrow(df_valid)
```

```
[1] 13428
```

ベースラインモデルを設定する

このタスクにおける性能比較用のベースラインモデルとして、「学習データにおける粗更新率を計算し、これを検証データの全契約に対する予測確率として利用する」という単純な粗更新率モデルを考えます。

```
crude_rate <- mean(df_train$Retention == 1)  
crude_rate
```

```
[1] 0.7431859
```

確率予測モデルの評価指標を設定する

このデモでは、二値分類モデルに対する評価指標としてloglossを用います。loglossはモデルの予測確率が真の確率にどれだけ近いかを評価する指標で、値が小さいほど、モデルの性能が優れていることを示します。

$$\text{logloss}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

ここで、 N はデータポイントの総数($= 13,428$)、 $y_i \in \{0, 1\}$ は真のラベルまたは真の確率、 $\hat{y}_i \in [0, 1]$ はモデルの予測確率を表します。

```
# 評価指標を定義する
logloss <- function(actual, pred)
  - mean(actual * log(pred) + (1 - actual) * log(1 - pred))
# 単純モデルによる予測を評価する
pred_crude <- rep_len(crude_rate, nrow(df_valid))
logloss(df_valid$Retention == 1, pred_crude) # ベースラインの評価値
```

[1] 0.5792336

ロジスティック回帰モデルを構築する

まず、解釈可能なモデルの例として、各契約の更新率を予測するロジスティック回帰モデルを構築します。

```
# ロジスティック回帰モデルを構築する
model_glm <- glm(
  Retention ~ (. - PolID - year), # モデル式(".." は「すべての変数」)
  family = binomial("logit"),      # 目的変数の分布とリンク関数
  data = df_train                 # 学習データ
)
```

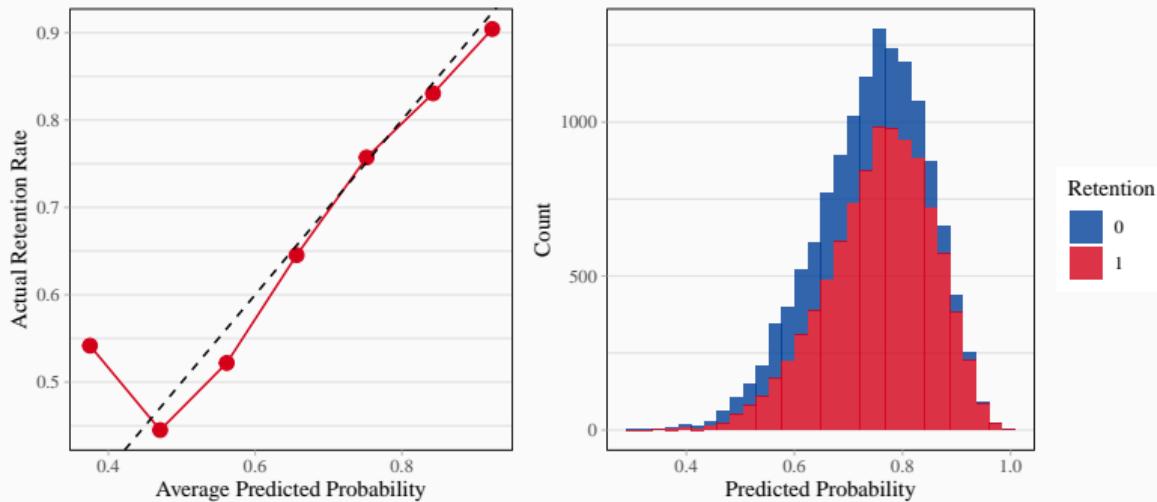
```
# 検証データに対する予測確率を得る
preds_glm <- predict(model_glm, df_valid, type = "response")
logloss(df_valid$Retention == 1, preds_glm) # 予測モデルの評価値
```

[1] 0.5507857

ロジスティック回帰モデルを検証する

予測値と正解ラベルをもとに較正プロットとヒストグラムを描きます。

予測確率が0.4未満の群で(おそらく群に含まれる契約件数の少なさにも起因して)実績更新率と予測確率に差があることを除けば、実績の更新率は予測確率におおむね対応しています。



ロジスティック回帰モデルを解釈する

midr パッケージを用いて予測モデルを解釈するには、まず、`interpret()` 関数を用いて対象モデルの「解釈モデル」を構築します。

```
# ロジスティック回帰モデルに関する1次の解釈モデルを構築する
mid_glm <- interpret(
  Retention ~ (. - PolID - year), # モデル式
  data = df_train,               # 学習データ
  model = model_glm,            # 対象モデル
  link = "logit"                # リンク関数(線形予測子ベースの解釈)
)
```

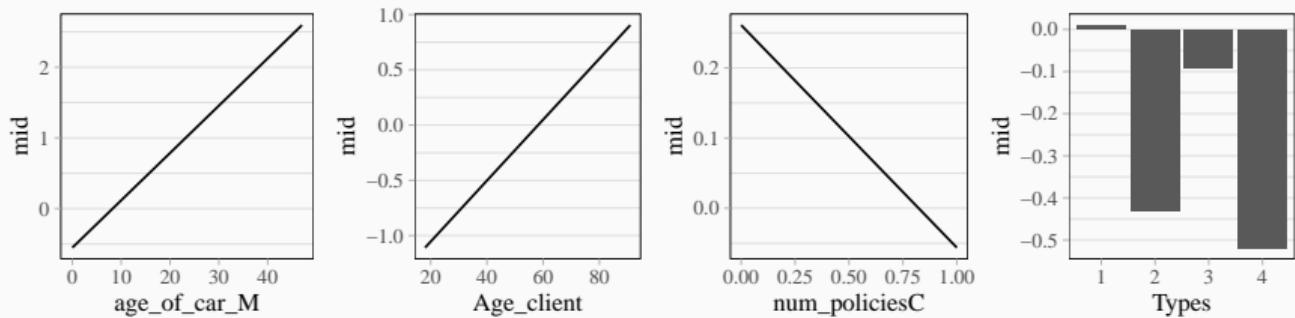
```
mid_glm$model.class # 対象モデルのクラス
```

```
[1] "glm" "lm"
```

ロジスティック回帰モデルの主効果を確認する(1/3)

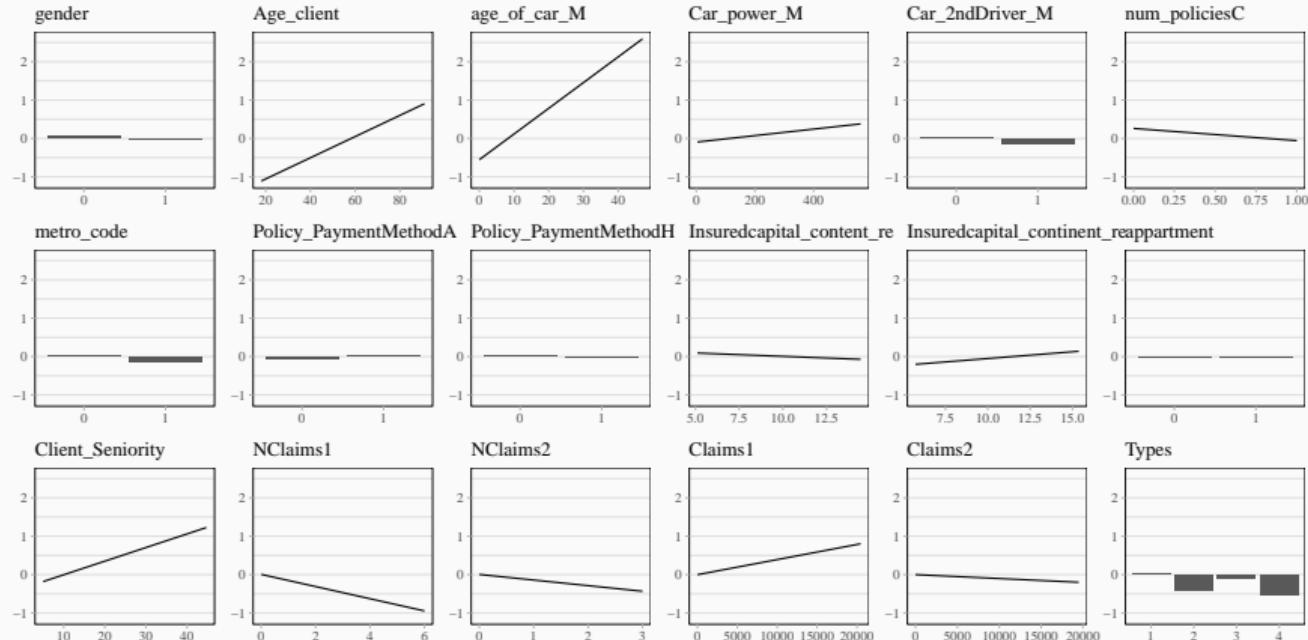
解釈モデルに対して `ggmid()` 関数や `plot()` 関数を適用することで、各特徴量の主効果を可視化することができます。

```
grid.arrange(ggmid(mid_glm, "age_of_car_M"),
             ggmid(mid_glm, "Age_client"),
             ggmid(mid_glm, "num_policiesC"),
             ggmid(mid_glm, "Types"),
             nrow = 1)
```



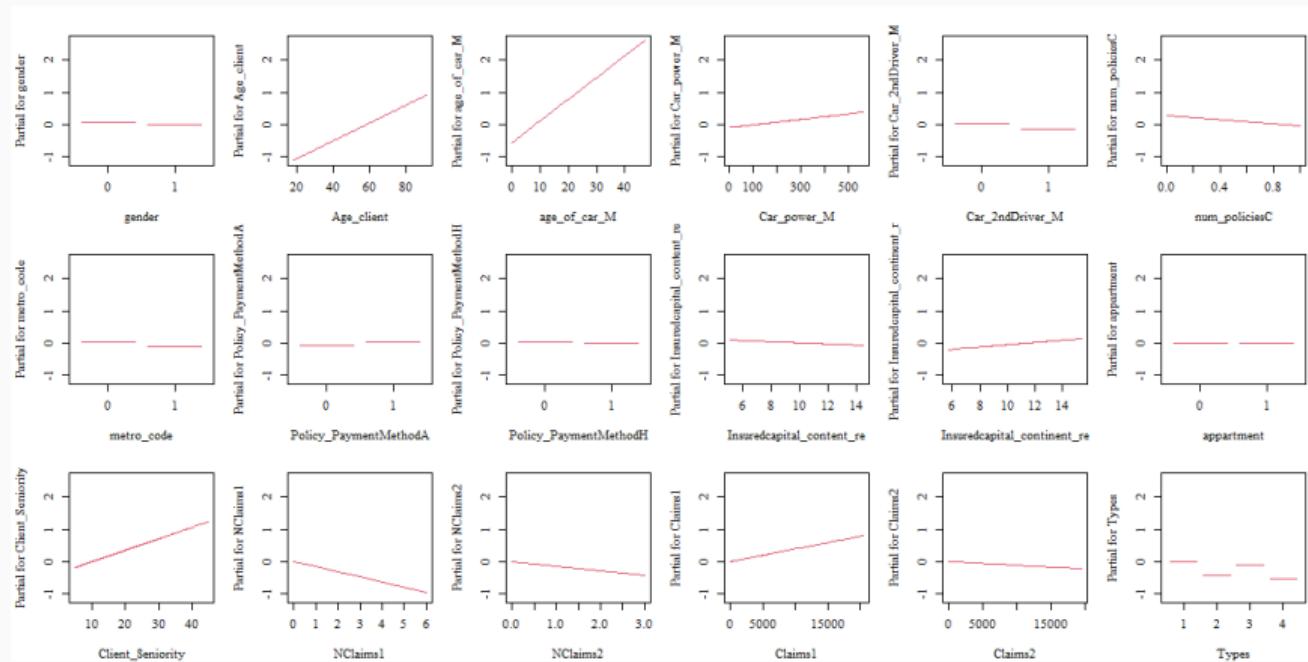
ロジスティック回帰モデルの主効果を確認する(2/3)

`mid.plots()`関数を用いると、全変数の主効果を一括で確認できます。



ロジスティック回帰モデルの主効果を確認する(3/3)

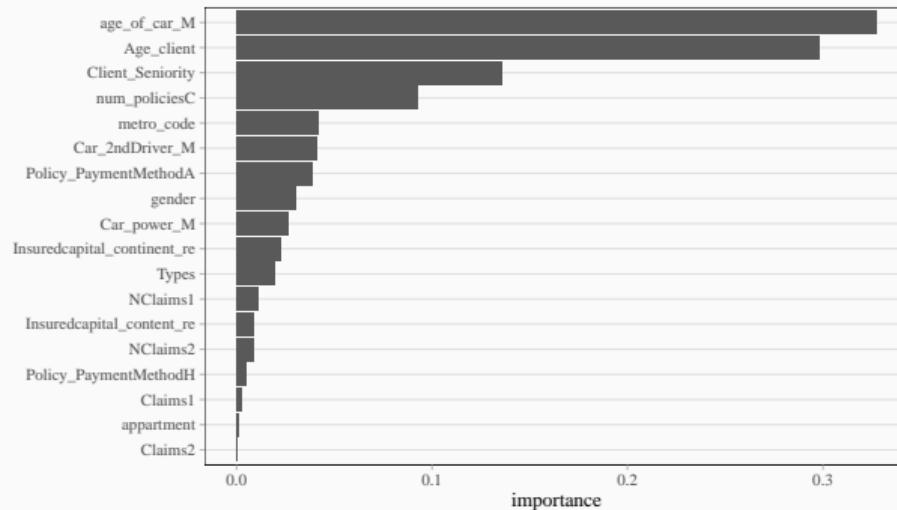
解釈モデルの正確性を確認するために、ロジスティック回帰モデルに対する
stats::termplot()関数の出力と比較します。



ロジスティック回帰モデルの特徴量重要度を確認する

学習済みの解釈モデルに `mid.importance()` 関数を適用すると、各特徴量の重要度を計算することができます。

```
imp_glm <- mid.importance(mid_glm)  
ggmid(imp_glm)
```



ランダムフォレストモデルを構築する

次に、解釈が難しい予測モデルの例として、各契約の更新率を予測するランダムフォレストモデルを構築します。

```
# ランダムフォレスト回帰モデルを構築する
set.seed(42)
model_rf <- ranger(
  Retention ~ (. - PolID - year), data = df_train,
  probability = TRUE, mtry = 5, min.node.size = 250
)
```

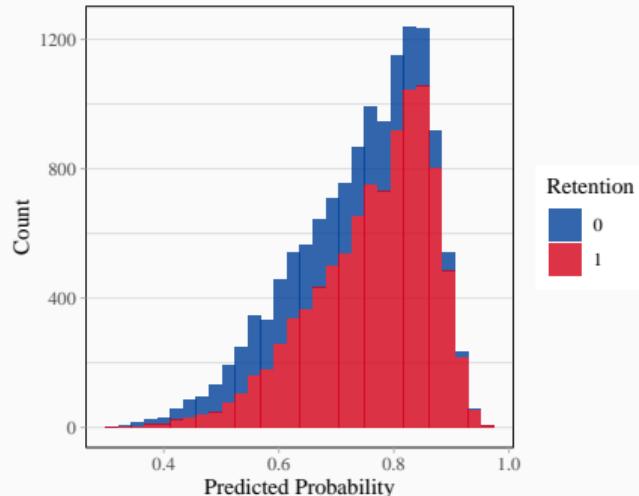
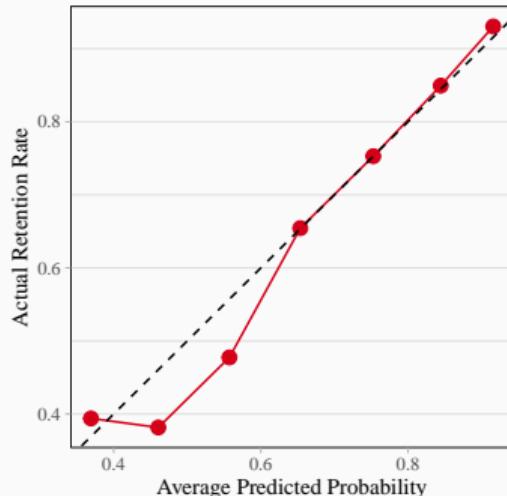
```
# 検証データに対する予測確率を得る
preds_rf <- predict(model_rf, df_valid)$prediction[,2L]
# モデルの Log Loss を計算する
logloss(df_valid$Retention == 1, preds_rf) # 0.5353356...
```

[1] 0.5353356

ランダムフォレストモデルを検証する

予測値と正解ラベルをもとに較正プロットとヒストグラムを描きます。

予測確率が0.4以上0.6未満の各群で実績更新率と予測確率に差があるものの、その他の群では実績と予測はよく対応しています。



ランダムフォレストモデルを解釈する

ランダムフォレストを対象とする解釈モデルを構築します。

モデル式を2乗することで、2変数間の交互作用をすべて含めることができます。また、引数 `k` と `lambda` で解釈モデルの柔軟性を調整します。

```
# ランダムフォレスト回帰モデルに関する2次のMIDモデルを構築する
mid_rf <- interpret(
  Retention ~ (. - PolID - year)^2, # 交互作用を含むモデル式
  k = c(100, 5), lambda = 0.05,      # 解釈モデルの柔軟性を調整
  data = df_train, model = model_rf,
  link = "logit", singular.ok = TRUE)
```

```
mid_rf$model.class # 対象モデルのクラス
```

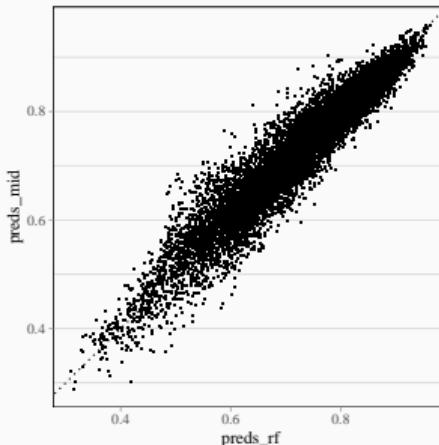
```
[1] "ranger"
```

解釈モデルの性能を評価する

構築した解釈モデルが対象モデルの良い解釈であることを確認するために、検証データに対する二つのモデルの予測確率の一致度を評価します。

```
# 解釈モデルによる予測確率を得てRMSEを計算する  
preds_mid <- predict(mid_rf, df_valid)  
sqrt(mean((preds_rf - preds_mid) ^ 2)) # RMSE
```

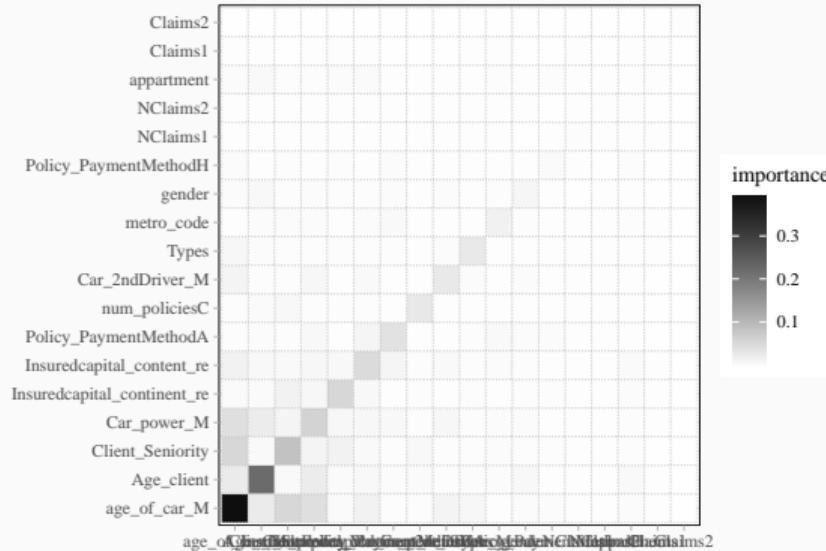
```
[1] 0.03223142
```



ランダムフォレストモデルの特徴量重要度を確認する

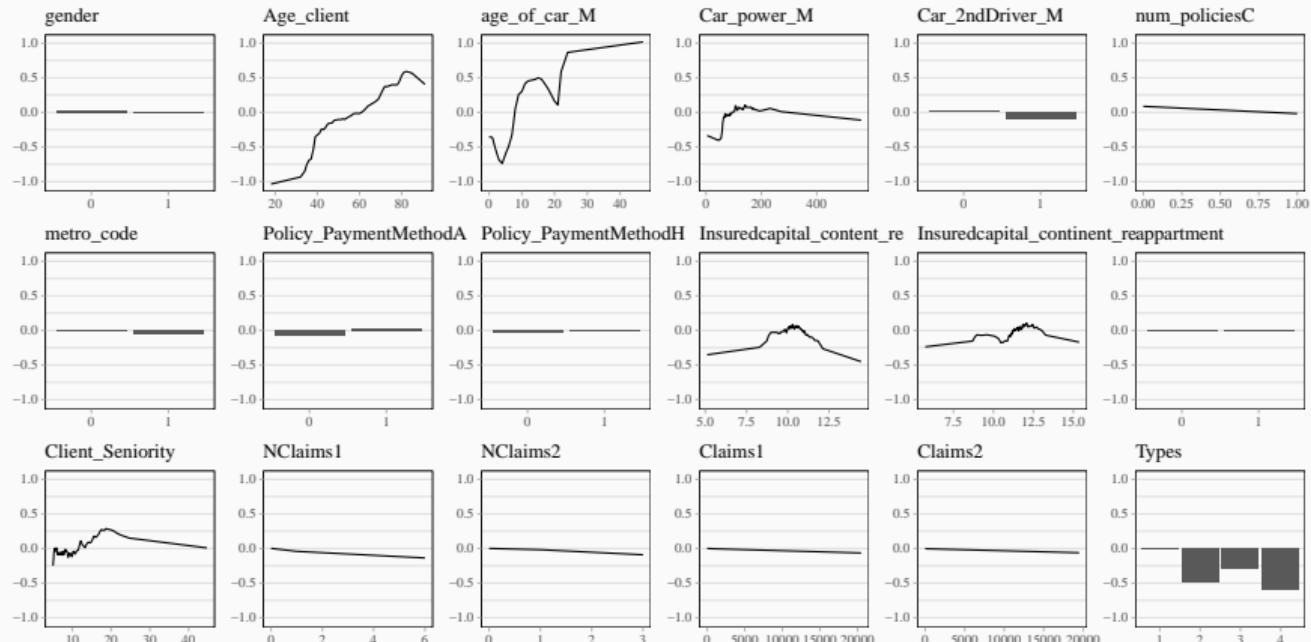
交互作用を含む解釈モデルの特徴量重要度を可視化する場合は、ヒートマップを利用することも可能です。

```
imp_rf <- mid.importance(mid_rf)
ggmid(imp_rf, "heatmap", col = "gray", lty = 3)
```



ランダムフォレストモデルの主効果を確認する

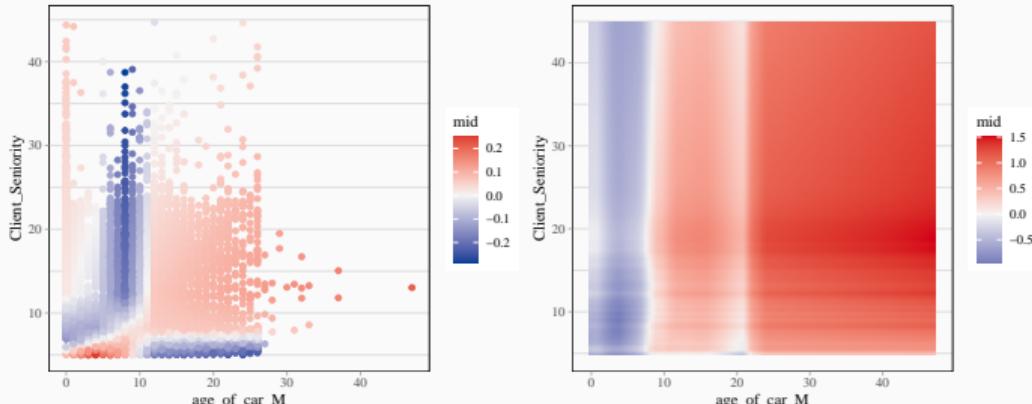
すべての変数の主効果を一覧で確認します。



ランダムフォレストモデルの交互作用を確認する

解釈モデルに対して `ggbmid()` 関数や `plot()` 関数を適用することで、交互作用を可視化することもできます。

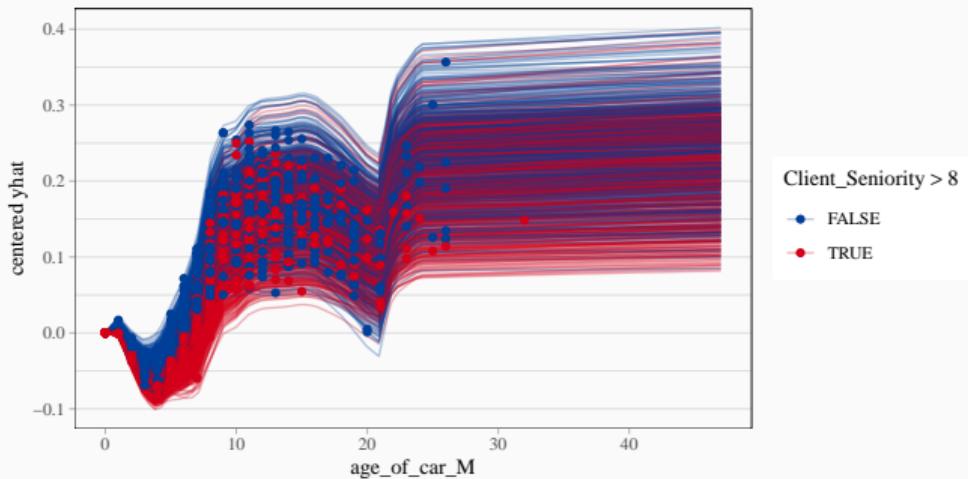
```
term_ie <- "age_of_car_M:Client_Seniority"
grid.arrange(nrow = 1,
             ggbmid(mid_rf, term_ie, type = "data", theme = "taikai"),
             ggbmid(mid_rf, term_ie, main.effects = TRUE, theme = "taikai")
           )
```



ランダムフォレストモデルの予測値の変化を確認する

解釈モデルに `mid.conditional()` 関数を適用することで、Individual Conditional Expectation (ICE) プロットを作成できます。

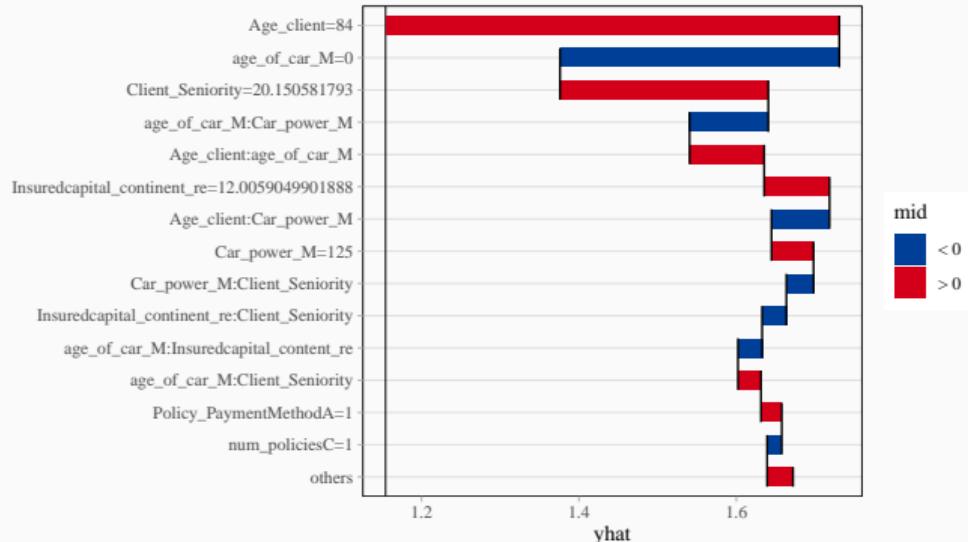
```
ice <- mid.conditional(mid_rf, variable = "age_of_car_M")
ggmid(ice, type = "centered", var.color = Client_Seniority > 8,
      theme = "taikai", alpha = .3)
```



ランダムフォレストモデルの予測を分解する

解釈モデルに `mid.breakdown()` 関数を適用することで、個別の契約に関する予測値を各項の貢献度に分解するプロットを作成できます。

```
mbd <- mid.breakdown(mid_rf, row = 10L)
ggmid(mbd, theme = "taikai@q")
```



解釈モデルをチューニングする(発展)

解釈モデルの柔軟性を決める `k` や `lambda` はハイパープラメーターであり、以下の方法を用いて自動でチューニングすることも可能です。

- `reticulate` パッケージを用いて R の中で Python を実行し、`optuna` などの高性能なハイパープラメータチューニング用ライブラリを利用する
- `parsnip` で解釈モデルを実装して `tune` のパラメータチューニング機能を利用する(この目的のパッケージ `midnight` を開発予定)

