応用プログラミング3

# 第10回　機械学習
# フレームワークによる
# 機械学習（1）

専修大学ネットワーク情報学部
田中健太

# 1．課題の振り返り

1

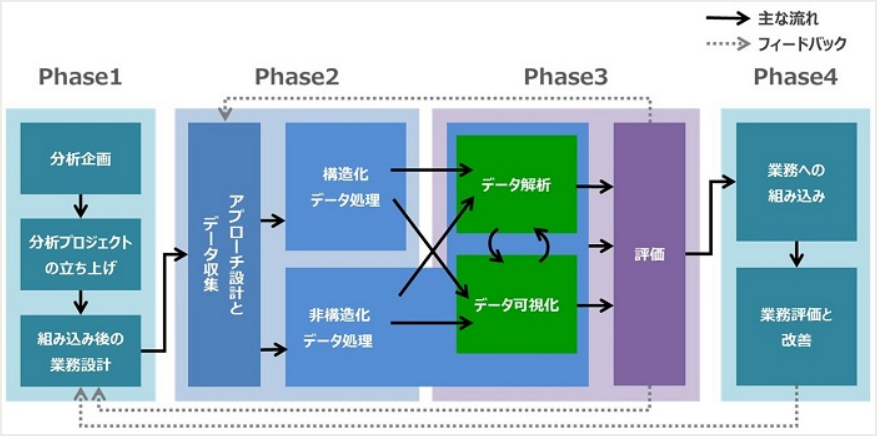
# 2. Rにおける機械学習フレームワーク

2

今回は、Rにおいて一貫したスタイルで、データの加工から機械学習モデルの作成、チューニング、精度評価、予測までを行うことができるフレームワークを紹介します。

## 2.1　機械学習フレームワークとは

- データ分析（機械学習）のプロセスは複雑
- 使用するパッケージごとに書式やデータ形式が異なる
- 機械学習の一連の作業を、統一されたインターフェースで行うことができるのが機械学習フレームワーク




ITSS+におけるデータ分析のフロー

3

研究やビジネスにおいて、本格的なデータ分析（機械学習）を行うためには、複雑で広範囲にわたる作業が必要です。

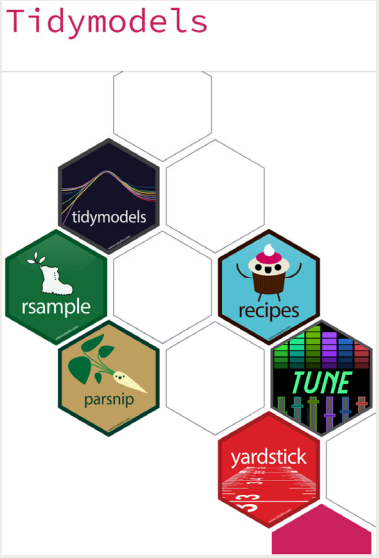
1. データを収集し、
2. その特徴を集計や可視化によって把握し、
3. 欠損値や外れ値に対処し、
4. 多数のアルゴリズムからデータにフィットしそうな候補を選び、
5. それぞれのアルゴリズムに適したハイパーパラメーターを探索してモデルを作成し、
6. モデルごとの精度を複数の指標で確認し、
7. 予測したいデータも適切に加工してから予測を行い、
8. 誤差を考慮して予測結果を活用する

といったような作業を、何度も試行錯誤しながら進めていきます。また、それぞれの処理に対応するパッケージがいくつも存在し、それらが受け付けるデータの構造や、関数の書式も異なります。そのため、データ分析が大規模になるにつれ、本質的な問題解決（モデリング）以外の部分が煩雑になります。

このような課題を解決するため、機械学習のフロー全体を管理し、各パッケージの違いを吸収した統一的なインターフェースで処理できるようにした機械学習フレームワークが利用されるようになっています。今回は、広く使われる2つのフレームワークを取り上げ、その利用法の概要を紹介します。

## 2.2　tidymodels

- https://www.tidymodels.org/
- Hadley Wickham, Posit社が主導する tidyverseをベースとしたフレームワーク
- 今後、RにおけるMLフレームワークのデファクトになるのでは




4

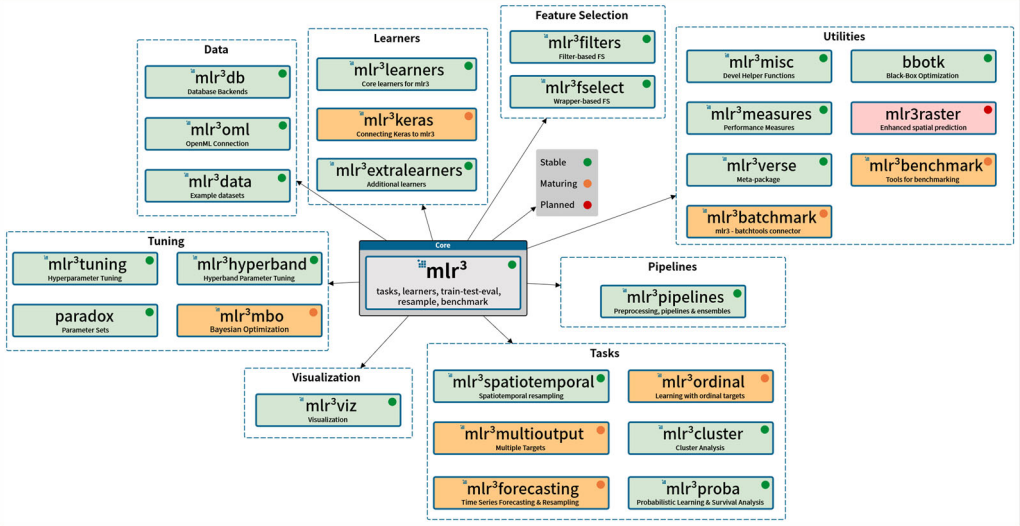
はじめに、tidymodelsを取り上げます。tidy, の名前の通り、Hadley WickhamやPosit社が主導して開発を進めているフレームワークです。tidyverseをベースに、パイプによるフローの中で、機械学習にまつわるさまざまな処理を効率的に記述できます。tidymodelsも、ひとつのパッケージですべてをカバーするものではなく、多数のパッケージを組み合わせたパッケージ群の総称です。

"R界" における最大勢力であるHadley WickhamとPosit社が関わっているため、今後Rで機械学習を行う際のスタンダードとなっていくものと思われます。

- 参考1: Tidy Modeling with R https://www.tmwr.org/
- 参考2: 【tidymodels講座1】当講座の概要 - データサイエンスの道標 https://datasciencemore.com/r-tidymodels-overview/

# 2.3 mlr3

- https://mlr3.mlr-org.com/
- 機械学習のプロセス全体にわたる広範な機能を提供
- 機能面ではtidymodelsに遜色ない




5

もうひとつ、広く使われているフレームワークとして、mlr3があります。もともと、mlrという名前で開発が進められてきましたが、そのバージョン3がリリースされたことを機にmlr3という名前になりました (今後バージョン4が出たらmlr4になるのかもしれませんが)。

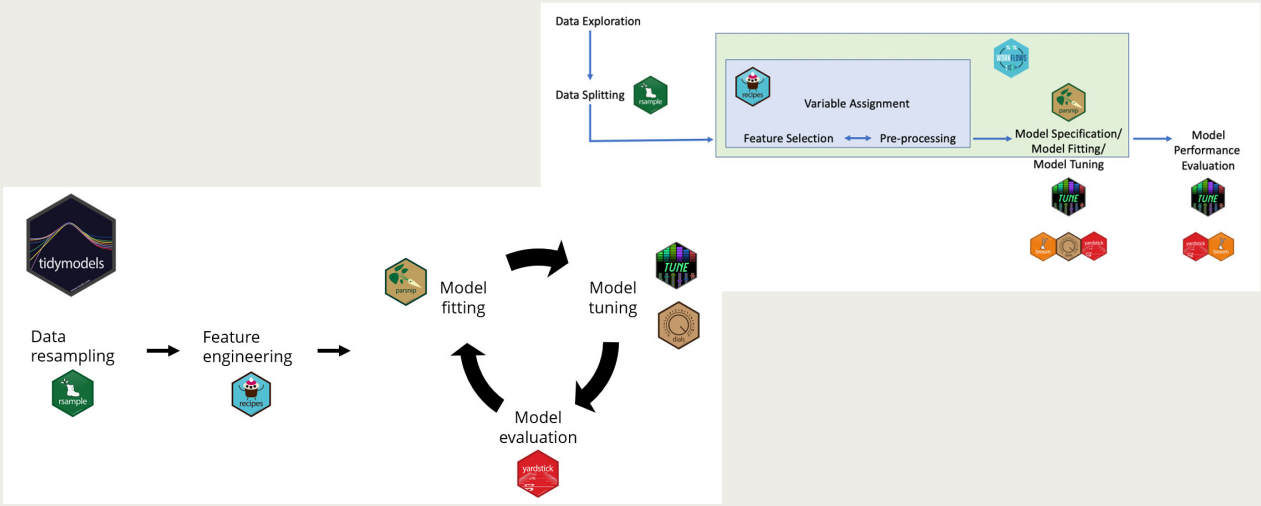
# 3. tidymodelsによる機械学習

6

まず、tidymodelsを用いて機械学習を行う際の全体の流れを紹介していきます。
複数のパッケージを組み合わせ、パイプでつなげていくことで、効率的で再現性の高いワークフローを実現できます。

## 3.1　tidymodelsによる機械学習の流れ

- tidymodelsはさまざまなパッケージ群の総称
- 機能ごとに細分化されたパッケージ、関数をパイプでつなげてフローを構築する




出典: RPubs - zetatech TidyModels tutorials https://rpubs.com/chenx/tidymodels_tutorial

7

前述のように、tidymodelsも多数のパッケージからなる「パッケージ群」です。ここでは、それらのパッケージがどのように使い分けられるかを整理した（誰かが作った）図を示します。詳しくは、このあと説明していきますが、以下のような流れになります。

1. rsampleパッケージでデータを学習用と検証用に分割
2. recipesパッケージで前処理
3. parsnipパッケージでモデルを作成
4. yardstickパッケージで精度評価
5. tuneパッケージなどでチューニング
6. 再度yardstickパッケージで精度評価

このようなワークフローを、統一されたインターフェースで扱うことができるのが、tidymodelsの利点です。

- 参考: RPubs - zetatech TidyModels tutorials
  https://rpubs.com/chenx/tidymodels_tutorial

## 3.2　rsampleパッケージによるデータの分割

- 機械学習においては精度の検証が必須
- 一般に、データを学習（モデル作成）用と検証用に分けて使用する
- rsampleパッケージの initial_split() 関数でデータフレームを分割し、training() 関数と testing() 関数でそれぞれのデータを抽出する
- 時系列性のあるデータでは、initial_time_split() 関数を使う

```
library(tidymodels)

iris_split <- initial_split(iris, prop = 0.8)

iris_train <- training(iris_split)
iris_test <- testing(iris_split)
```

8

はじめに、データをサンプリングして、学習（モデル作成）用と検証（精度評価）用に分割する工程です。機械学習においては、（当然ですが）作成したモデルの精度を検証することが必要です。その際、手元にあるすべてのデータを用いてモデルを作ると、検証のためのデータがなくなります。検証に、モデル作成に使用したデータを用いると、モデルは答えを知っていることになるため、精度を過大評価してしまうことになります。そのため、モデルを作成する前の段階で、手元のデータを学習用と検証用に分割しておく必要があります。

Rの基本機能でも、sample() 関数などの乱数と、そこで得られた値を行番号として使うことでランダムサンプリングができますが、より洗練された方法として、rsampleパッケージを使うことができます。rsampleパッケージの initial_split() 関数で、データフレームを学習用と検証用に分割できます。prop オプションに学習用データの割合を指定します。また、完全にランダムにサンプリングすると、実際の目的変数の割合に合致しないデータになるため、「ラベルAから〇件、ラベルBから〇件…」と層別抽出を行いますが、その設定も strata オプションなどで指定できます。分割したデータは、それぞれ training() 関数と testing() 関数で取り出すことができます。

なお、時系列性のあるデータは、ランダムサンプリングすると、「未来のデータで過去を予測する」ことになりかねないので、時期を決めて、学習用の期間と検証用の期間に分割します。これも、initial_time_split() 関数で容易に実行できます。

- 参考1: General Resampling Infrastructure • rsample
  https://rsample.tidymodels.org/index.html
- 参考2: 【tidymodels講座3】{rsample}データ分割 - データサイエンスの道標
  https://datasciencemore.com/tidymodels-rsample/

# 3.3　recipesパッケージによる前処理

- データの前処理工程を「レシピ」として管理、実行する
- recipe() 関数でモデルと処理対象のデータを定義する
- 多種多様な step_*() 関数で前処理を定義できる

```
iris_rec <- recipe(Species ~ ., data = iris_train) %>%
    step_normalize(all_numeric_predictors())
```

9

　データ分析においては、分析しやすくする、あるいはモデルの精度を高めるための前処理が必要不可欠です。「データ分析は前処理が8割」といった言葉もあります。さまざまな前処理を共通の書式で適用するための仕組みがrecipesパッケージです。

　実際には、recipesは前処理に限らず、データ分析の "レシピ" を共通化し、データ分析における様々な前処理、アルゴリズム、パッケージを統一的に扱うためのパッケージです。はじめに recipe() 関数でモデル式とデータを指定し、続けてさまざまな処理を順に step_*() 関数で登録し、パイプでつなげ、一括して実行します。

　詳細は次回解説しますが、正規化 (0から1の範囲) を行う step_center() 関数や、標準化 (平均0、標準偏差1) を行う step_normalize() 関数などがあります。

- 参考1: Preprocessing and Feature Engineering Steps for Modeling • recipes https://recipes.tidymodels.org/
- 参考2: dpp-cookbook | Baked your data :) https://uribo.github.io/dpp-cookbook/
- 参考3: 【tidymodels講座5】{recipes}特徴量エンジニアリング - データサイエンスの道標 https://datasciencemore.com/tidymodels-recipes/

## 3.4　parsnipパッケージによるモデルの設定

- 使用するアプローチ（回帰 / 分類）とアルゴリズムを登録する
- logistic_reg() 関数、decision_tree() 関数などでアプローチを指定する
- さまざまなパッケージへのインターフェースが提供されており、統一的にアクセスできる
- set_engine() 関数で使用するパッケージ、関数を指定する

```
dt_model <- decision_tree(mode = "classification") %>%
    set_engine("rpart")
```

10

次に、さまざまな機械学習アルゴリズムを適用し、モデルを作成します。これにはparsnipパッケージを使います。「パースニップはセリ科の一〜二年草で、根の部分がニンジンとよく似ており同じように根の部分を食用にします」だそうです。開発者のプレゼンテーション（ https://www.youtube.com/watch?v=ZFTjroC8bTg ）によると、「Rにはすでに機械学習フレームワークとしてcaretパッケージがあります。caret…carrot…carrrotは野菜…white carrotと呼ばれるparsnip…そうだ、このパッケージの名前はparsnipにしよう」という頭のよい🤓理由が解説されていました。

Rでは、前回紹介したように glm() 関数やrpart、randomForestパッケージなど多種多様な機械学習アルゴリズムが利用できます。しかし、それぞれ書式などが異なり、試行錯誤する際のプログラミングコストが高くなります。parsnipはその違いを吸収し、アルゴリズム（ロジスティック回帰、決定木など）とモデルの種類（回帰 / 分類）と使用するパッケージを指定すれば、どのパッケージ、関数も同じ書式で利用できるようにします。

parsnipパッケージはあらゆるアルゴリズム、パッケージをサポートしているわけではありませんが、その対象は随時増加しています。なお、この段階ではまだ計算は実行されません。

- 参考1: A Common API to Modeling and Analysis Functions • parsnip
  https://parsnip.tidymodels.org/
- 参考2: Explore tidymodels - Search parsnip models
  https://www.tidymodels.org/find/parsnip/
- 参考3: 【tidymodels講座4】{parsnip}学習ルール設定 - データサイエンスの道標
  https://datasciencemore.com/tidymodels-parsnip/

## 3.5 workflowsパッケージによるワークフローの登録

- モデル、前処理、アルゴリズムをまとめて一連の処理（ワークフロー）として管理する
- workflow() 関数で初期化し、add_model() 関数、add_recipe() 関数で登録する

```
iris_wflow <- workflow() %>%
    add_model(dt_model) %>%
    add_recipe(iris_rec)
```

11

rsampleパッケージでデータを分割し、recipesパッケージで前処理を指定し、parsnipパッケージで使用するアルゴリズムを決めたら、それらの一連の処理を「ワークフロー」として登録します。workflowsパッケージが、ワークフローの管理と実行を担います。

はじめに workflow() 関数で初期化し、続けて add_model() 関数や add_recipe() 関数で各種の設定を追加していきます。ここで登録している dt_model は、アルゴリズムと式を指定しているだけなので、前処理とどちらを先に登録しても問題ありません。

さまざまなモデルで共通して適用する前処理と、特定のアルゴリズムに固有の前処理などを別々のレシピとして登録しておくと、ワークフローを登録する際にそれらを入れ替えるだけで、効率的に試行錯誤を繰り返すことができます。また、この段階では、「流れ」を設定しているだけで、計算はまだ行われません。

- 参考1: Modeling Workflows • workflows https://workflows.tidymodels.org/
- 参考2: 7 A Model Workflow | Tidy Modeling with R https://www.tmwr.org/workflows.html
- 参考3: 【tidymodels講座6】{workflows}レシピ ＋ 学習ルール - データサイエンスの道標 https://datasciencemore.com/tidymodels-workflows/

## 3.6　モデルの作成と予測

- workflowsパッケージの fit() 関数でワークフローを実行し、モデルを作成できる
- extract_fit_parsnip() 関数で結果を確認できる
- predict() 関数で予測結果を得られる

```
iris_fit <- iris_wflow %>% fit(data = iris_train)
iris_fit %>% extract_fit_parsnip()
## parsnip model object
...
## 1) root 120 79 versicolor (0.32500 0.34167 0.33333)
...
predict(iris_fit, iris_test)
##    .pred_class
##  1 setosa
##  2 setosa
...
```

12

いよいよ、一連のワークフローを実行します。作成したワークフローオブジェクトに対して、fit() 関数を適用すると、登録したレシピが実行されます。また、作成したモデルの出力は、extract_fit_parsnip() 関数で確認できます。この関数は、基本的には呼び出した各関数、パッケージの出力をそのまま表示します。

作成したモデルを用いて、新しいデータに対する予測を行うには、(workflowsパッケージの) predict() 関数を使います。

# 3.7　モデルの精度評価

- workflowsパッケージの augment() 関数で評価のための情報を抽出する
- yardstickパッケージの metrics() 関数などで精度を評価できる

```
iris_aug <- augment(iris_fit, iris_test)

iris_aug %>% metrics(truth = Species, estimate = .pred_class)
## # A tibble: 2 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.933
## 2 kap      multiclass     0.900
```

13

モデルを作成したら、精度を評価します。tidymodelsでこれを担うのはyardstickパッケージです。ヤードスティックは物差し・定規のことだそうです。

はじめに、「正解」(実測値) と予測値からなる評価用のデータセットを作成します。これにはworkflowsパッケージの augment() 関数を使います。データセットに対して、どのような評価指標を算出するかは、metrics() 関数で指定します。この関数は、機械学習の種類 (回帰 / 分類) に合わせて、一般的な評価指標をまとめて算出します。

また、metric_set() 関数の引数に評価指標を列挙して、結果をオブジェクトに代入すると、それが関数のようになり、ユーザーが指定した任意の評価指標をまとめて算出できます。

ここまで、tidymodelsによる機械学習のワークフローを紹介しました。もちろん、個別のパッケージ、関数にはもっと多様な機能があるので、詳しく知りたい方は、各パッケージのドキュメントなどを参照してください。

- 参考1: Tidy Characterizations of Model Performance • yardstick
  https://yardstick.tidymodels.org/index.html
- 参考2: 9 Judging Model Effectiveness | Tidy Modeling with R
  https://www.tmwr.org/performance.html
- 参考3: 機械学習で使う指標総まとめ(教師あり学習編) - プロクラシスト
  https://www.procrasist.com/entry/ml-metrics

# 3.7　モデルの精度評価

```
mymetrics <- metric_set(accuracy, precision, recall)

iris_aug %>% mymetrics(truth = Species, estimate = .pred_class)
## # A tibble: 3 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  multiclass     0.933
## 2 precision macro          0.930
## 3 recall    macro          0.930
```

# 4. mlr3による機械学習ワークフロー

15

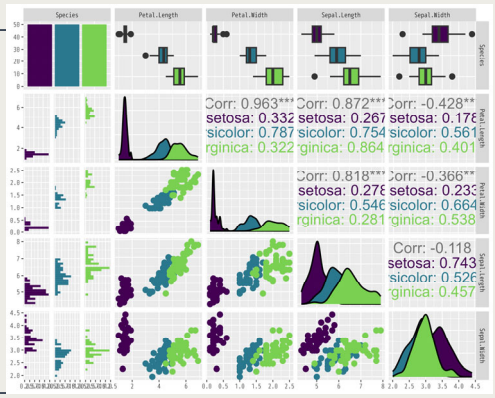
次に、mlr3フレームワークによる機械学習の流れを紹介します。

# 4.1　タスクの定義

- データと分析アプローチ（回帰、分類など）を対応付ける
- as_task_regr() 関数や as_task_classif() 関数を使い分ける
- mlr3vizパッケージの autoplot() 関数などでデータを観察する

```
library(mlr3verse)

task <- as_task_classif(iris,target="Species")
autoplot(task, type = "pairs")
```



16

mlr3を用いる場合、周辺のパッケージ群を含めたmlr3verseパッケージを読み込むのが便利です。

mlr3では、はじめに「タスク」を定義します。これは、データと機械学習の種類(回帰 / 分類) をセットで登録するものです。回帰の場合は as_task_regr() 関数、分類の場合は as_task_classif() 関数を使います。target 引数に目的変数となる列を指定します。

また、この段階でのデータの様子を、mlr3verseに含まれるmlr3vizパッケージの autoplot() 関数でグラフィックスとして確認できます。mlr3vizパッケージは、基本的に autoplot() 以外の関数は提供していません。

## 4.2　モデルの設定

- lrn() 関数で使用するアルゴリズムを登録する
- mlr3パッケージ単体ではrpartしか対応していないが、mlr3learnersパッケージなどが豊富なアルゴリズムへのインターフェースを提供している
- ハイパーパラメーターはオブジェクトの属性を変更することで設定する

```
learner <- lrn("classif.rpart")
learner$param_set
## <ParamSet>
##                id    class lower upper nlevels        default value
##  1:            cp ParamDbl     0     1     Inf          0.01
...
learner$param_set$values <- list(cp = 0.1, keep_model = TRUE)
```

17

次に、使用するアルゴリズムを登録します。mlr3では、lrn() (Learner) 関数にサポートするアルゴリズムを指定します。mlr3パッケージ単体ではrpartパッケージによる決定木しかサポートしていませんが、mlr3learnersパッケージやmlr3extralearnersパッケージが豊富なアルゴリズムへのインターフェースを提供しています。

また、それぞれのアルゴリズムごとに、さまざまなハイパーパラメーターが存在しますが、それらは作成したオブジェクトの $param_set$values にlist形式で指定します。指定可能なパラメーターは、$param_set$ids() で確認できます。

```
library(mlr3learners)
mlr_learners$get("classif.ranger")
learner <- lrn("classif.ranger")

learner$param_set$values <- list(mtry = 3, importance = "impurity_corrected",
    num.trees = 100, num.threads = 4)
```

## 4.3　データの分割

- partition(), rsmp() 関数で学習用と検証用に分割できる
- 分割の比率などはオブジェクトの属性を変更して設定する
- オブジェクト名$train_set() で学習用、オブジェクト名$test_set() で検証用データにアクセスできる

```
split <- partition(task, ratio = 0.8)
data <- rbind.data.frame(
    task$data(split$train),
    task$data(split$test)
)
data[["split"]] <- rep(c("train", "predict"), lengths(split))
```

18

続いて、データを分割します。mlr3では、タスクオブジェクトに対して、partition() 関数や resample() 関数を適用します。partition() 関数は、ratio 引数に学習用データの割合を指定します。返り値は、単に学習用データの行番号と検証用データの行番号が格納されたリストです。また、タスクオブジェクトの $data() メソッドでデータにアクセスできますが、その際に partition() 関数の返り値を指定することで、それぞれのデータを個別に取り出すことができます。

また、データを繰り返しサンプリングし、モデル作成と評価を複数回行う交差検証 (Cross validation) は、rsmp() 関数で設定できます。こちらは、rsmp() 関数で作成したオブジェクトの $instantiate() メソッドにタスクオブジェクトを登録することで、train_set() メソッドや test_set() メソッドで各繰り返し (folds) ごとのデータにアクセスできます。

交差検証を含む、より高度なモデリングについては、以下のURLを参照してください。

- 参考1: mlr3を使ってみる(①タスク及び学習器の構築) - Qiita
  https://qiita.com/kenkenvw/items/c2c27d917a95b4a8b8a5
- 参考2: mlr3を使った機械学習 (その２) - Qiita
  https://qiita.com/kenkenvw/items/3d80d5d66c4baf2b6ed6

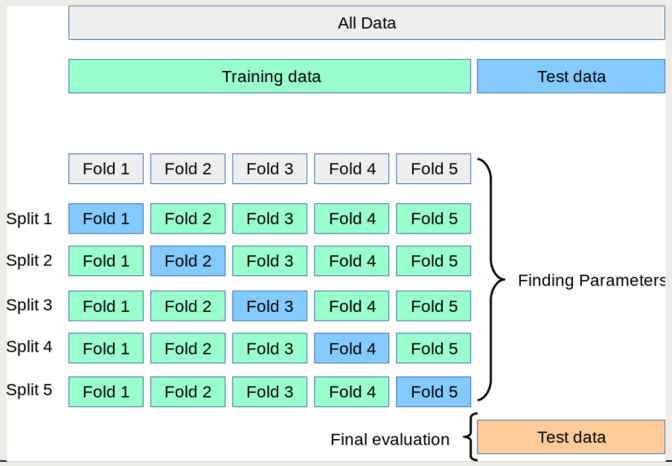
# 4.3　データの分割

```
resampling <- rsmp("cv", folds = 10)
resampling$instantiate(task)
resampling$train_set(1)
```

```
##   [1]   7  11  16  17  25  28  39  61  91  92 112 115 127 135 143   1  12  42 ...
```

```
resampling$test_set(1)
```

```
##  [1]  13  19  22  43  44  63  70  72  81  83  94 113 129 133 148
```




3.1. Cross-validation: evaluating estimator performance
scikit-learn 1.1.3 documentation
https://scikit-learn.org/stable/modules/cross_validation.html

19

## 4.4　学習

- learner オブジェクトの train() メソッドで学習を実行する
- 作成したモデルには model 属性でアクセスできる
- 決定木はmlr3vizパッケージの autoplot() 関数で描画できる
- 分類の場合、ラベルと所属確率の出力を predict_type 属性で指定できる

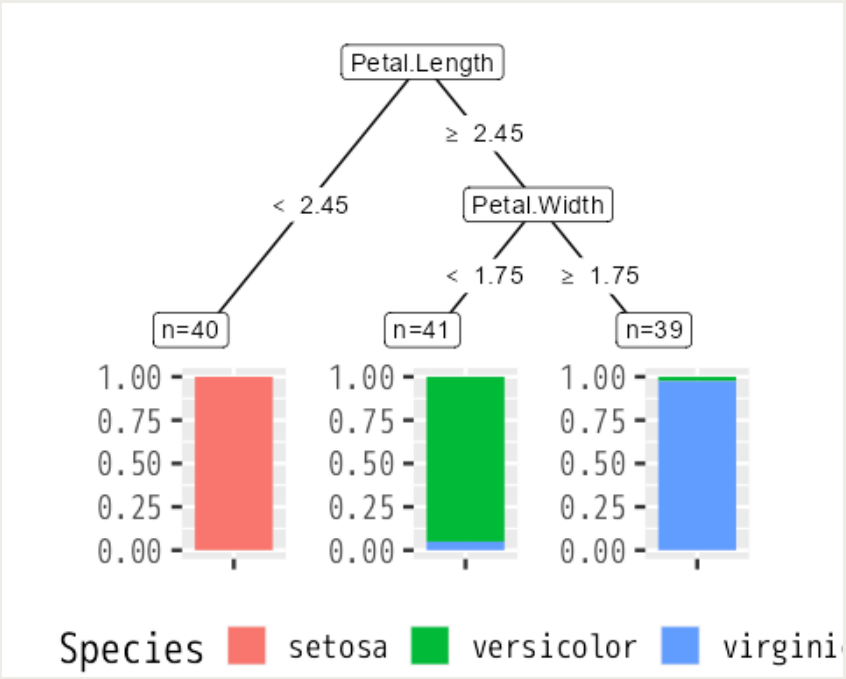
```
learner$predict_type <- "prob"
learner$train(task, split$train)
learner$model
...
## 1) root 120 80 setosa (0.33333 0.33333 0.33333)
...
```

20

準備が整ったら、学習を行います。学習は、すでに作成した learner オブジェクトの train() メソッドで実行します。引数に、タスクと学習用のデータを指定します。作成したモデルには、learner オブジェクトの $model 属性としてアクセスできます。また、決定木の場合、autoplot() 関数で前回紹介したggpartyパッケージを使って描画できます。

# 4.4 学習

```
autoplot(learner)
```




21

# 4.5　予測

- predict() メソッドに検証用データを指定して予測し、評価する
- 未知のデータに対しては predict_newdata() メソッドを使用する

```
pred <- learner$predict(task, split$test)
pred$response
```

```
##  [1] setosa     setosa     setosa     setosa ...
```

```
pred$prob
```

```
##       setosa versicolor virginica
##  [1,]      1    0.00000   0.00000
##  [2,]      1    0.00000   0.00000
...
```

```
autoplot(pred)
```


22

次に、作成したモデルに検証用データを与えて予測します。これには、learner オブジェクトの predict() メソッドを使います。引数は、train() と同様です。

learner オブジェクトの $response 属性で予測結果が得られます。また、検証用ではなく、未知のデータに対して予測したい場合は、$predict_newdata() メソッドを使います。こちらは、予測結果がダイレクトに返ってきます。

# 4.5 予測

```
learner$predict_newdata(iris[1:5, ])
## <PredictionClassif> for 5 observations:
##  row_ids  truth response prob.setosa prob.versicolor prob.virginica
##        1 setosa   setosa           1               0              0
##        2 setosa   setosa           1               0              0
##        3 setosa   setosa           1               0              0
##        4 setosa   setosa           1               0              0
##        5 setosa   setosa           1               0              0
```

## 4.6　精度評価

- score() メソッドで評価指標を出力する
  （デフォルトは誤り数）
- mlr_measures オブジェクトで評価指標を確認し、
  msr() 関数で選択する

```
as.data.table(mlr_measures)
##                   key                                        label
##  1:               aic                      Akaika Information Criterion
##  2:               bic                    Bayesian Information Criterion
##  3:        classif.acc                            Classification Accuracy
...
##    task_type                    packages predict_type task_properties
##  1:      <NA>                        mlr3     response
##  2:      <NA>                        mlr3     response
##  3:   classif           mlr3,mlr3measures     response
...
```

24

learner オブジェクトから、評価指標にもアクセスできます。$response 属性で予測結果が得られます。$confusion 属性では、混同行列（Confusion matrix）が得られます。また、正解率や適合率、再現率などの指標は、$score() メソッドの引数に msr("指標名") を指定して得ることができます。正解率は classif.accurary、適合率は classif.precision、再現率は classif.recall と指定します。

回帰や分類において、どのような評価指標が出力できるかの情報は、mlr_measures オブジェクトに格納されています。もちろん、Webでも確認できます
( https://mlr3.mlr-org.com/reference/mlr_measures.html )。

# 4.6 精度評価

```
pred$confusion
##             truth
## response     setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         3
##   virginica       0          0         7
pred$score(msr("classif.acc"))
## classif.acc
##         0.9
```

# 5. まとめ



26

## 5.1　今日の内容

- Rにおける機械学習フレームワーク
    - tidymodels
    - mlr3
- tidymodelsによる機械学習ワークフロー
- mlr3による機械学習ワークフロー

## 5.2　次回までの課題

- RStudio Cloudプロジェクト内の
  `11_ml_framework_exercise.R` について、
  指示に従ってプログラムを作成してください
- 編集したファイルは、ファイル一覧でチェックを
  入れ、[more] メニューから [Export] を選択し、
  [Download] ボタンを押してダウンロードして
  ください
- ダウンロードしたファイルを、Classroomの
  課題ページから提出してください
- **提出期限: 2023-12-04 23:59**



28