tidymodels_notes

Software for Modeling

Models are very multifaceted & serve various purposes in statistical methods, but they are ultimately meant to disentangle complicated statistical phenomena.

Software used to create models should:

- have a easy-to-use interface
- protect users from making mistakes

Types of Models

Descriptive Models

Descriptive models are used generally to describe data attributes/traits, more of an opportunity to get to know your data better rather than make any kind of inference about a relationship related to the data

Inferential Models

Models of this nature are use to test a particular claim of a relationship between variables, a particular hypothesis, etc. After this model is created, a state of statistical significance/insignificance is produced

Output from these models usually includes a p-value, confidence intervals, etc. to estimate probabilites (i.e., of committing Type 1 error, etc.)

Predictive Models

These models deal with questions of estimation as opposed to inference

This type of model is used to provide accurate predictions utilizing previous/historical data

Building Predictive Models

- Mechanistic models: using predetermined equations/assumption to derive a model equation, can make statements about how the model will perform based on how it performs with existing data
- Empirically driven models: machine learning models, example: K-nearest neighbor (given a dataset, a new sample is speculated using the K most similar data in the set.)
  - note: "the primary method of evaluating the appropriateness of the model is to assess its accuracy using existing data"

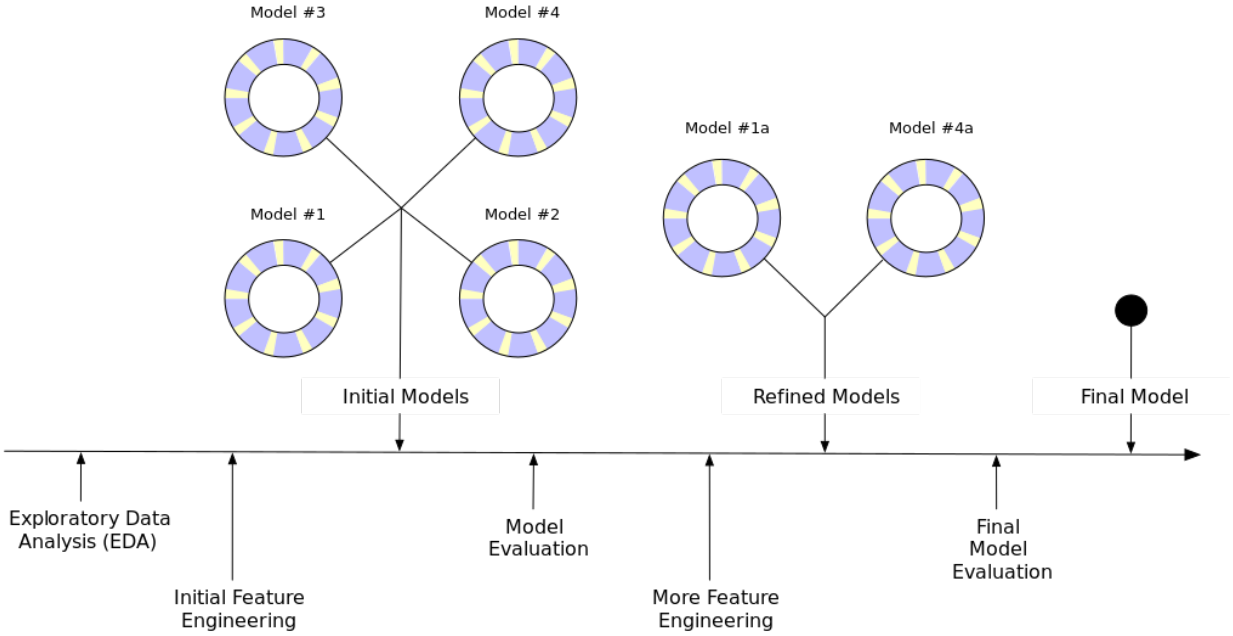
You may generate models that are statistically significant, but that doesn't necessarily mean its the best model for your data/represents your data appropriately

Terminology

- Unsupervised models: those models that learn patterns, clusters or other characteristics of the data but lack an outcome
- Supervised models: those models that have an outcome variable
  - Regression: predicts numeric outcome
  - Classification: produces ordered/unordered qualitative values

Phases of Data Analysis

- Cleaning data
  - involves looking closely at your data to see how it is presented/aligns with the research questions you are trying to answer
- Exploratory data analysis (EDA)
  - understanding how the data came about (sampling information, etc.); should also consider how appropriate/relevant the data are to your research
  - one should outline a clear goal for the model and how success with the model will be judged



Process for choosing appropriate model

- Exploratory data analysis (EDA)
  - oscillating between numerical analysis & data visualization
- Feature engineering
  - refiguring predictors to make them easier to model
- Model tuning and selection
  - creating initial models to assess & compare; usually involves parameter tuning, etc.
- Model evaluation
  - using formal evaluation techniques to compare model results, differences, etc.

Tidyverse notes

Gives broad explanation as to the differences between the tidyverse & base R, essentially providing evidence as to how tidyverse is more intutive & user-friendly for folks that are not super familiar with coding/programming. Goes over syntax, the importance of tibbles & pipes in the tidyverse and provides examples of how the tidyverse is super helpful in the modeling process

Review of R Modeling Fundamentals

Summarizes the workings of the R formula and typical functions that are utilized in model creation. As general principle, tidymodels relies heavily on the strength of R's existing functions (object-oriented programming) and emphasizes understandable defaults to function arguments, consistency across function names & arguments and user-friendliness (not restricting functions to specific data structures)

Briefly talks about combining base R w/ tidyverse that might be helpful in creating your models.

Tidymodels is essentially considered a "metapackage" with a core set of tidymodels & packages. It splits packages by their function (ex: data splitting/resampling w/ rsample, measuring performance w/ yardstick)

Basics (using Ames housing data)

models sale prices of homes in Ames, IA using a histograms (with transformed/not transformed data; i.e. log data)

conducting EDA (exploratory data analysis) to determine distributions of predictors, correlations between predictors or possible associations between predictors & outcomes

Spending our data

Steps to creating useful model:

- parameter estimation
- model selection
- tuning
- performance assessment

data spending: allocating specific subsets of data to do different things

common ways to split/"spend" data:

- split the larger dataset into two, using one as a training set to develop & optimize the model
  - the second set is the test set, used after 1-2 models are selected and the test set is used to produce these models & determine their efficacy
- r function initial_split() will perform this action for you, takes the dataset & desired ratio as arguments
  - training() and testing() will produce the necessary datasets with the object you just created with initial_split()

can either use simple random sampling/stratified sampling to contribute to the model (depends on the level of imbalance in the data) - the strata argument can be added to the initial_split() function to conduct stratified random sampling

**for time series data: initial_time_split() will save most recent data for the test set

validation sets can also be part of this data splitting process, getting a sense of model performance before using the test set

with multi-level data, have to keep in mind that data splitting should be happening according to the independent unit level of the data

Fitting models with parsnip

Parsnip is a tidymodels package that can be used to produce a variety of different models - the functions fit() and predict() are the most commonly used with a parsnip object

example: modeling linear regression

- base R: lm() for ordinary linear regression and stan_glm() for regularized linear regression (Bayesian); for non-Bayesian approach & regularized regression –> glmnet()
- tidymodels unifies this process, eliminates need to memorized different syntax for functions
  1) identify model type (linear regression, random forest, etc.)
  2) identify engine for fitting the model (i.e., software package)
  3) identify the mode of the model (regression/classification)

```
linear_reg() %>% set_engine("lm")
# linear regression model specification, computational engine: lm

linear_reg() %>% set_engine("glmnet")
# linear regression model specification, computational engine: glmnet
```

you can use fit() or fit_xy() to begin model estimation; fit() is more appropriately used with a formula, fit_xy() is more appropriate if data is pre-processed

translate() will explain the process of the user's code being converted to package syntax

```
linear_reg() %>% set_engine("lm") %>% translate()
# Linear Regression Model Specification (regression)
# Computational engine: lm
# Model fit template:
# stats::lm(formula = missing_arg(), data = missing_arg(), weights = missing_arg())

lm_model <-
  linear_reg() %>%
  set_engine("lm")

lm_form_fit <-
  lm_model %>%

fit(Sale_Price ~ Longitude + Latitude, data = ames_train)

lm_xy_fit <-
  lm_model %>%
  fit_xy(
      x = ames_train %>% select(Longitude, Latitude),
      y = ames_train %>% pull(Sale_Price)
    )

lm_form_fit

# Call:
# stats::lm(formula = Sale_Price ~ Longitude + Latitude, data = data)

# Coefficients:
# (Intercept)    Longitude     Latitude
#     -302.97        -2.07         2.71
#

lm_xy_fit

# parsnip model object
#
#
# Call:
# stats::lm(formula = ..y ~ ., data = data)
#
# Coefficients:
# (Intercept)    Longitude     Latitude
#     -302.97        -2.07         2.71
```

parsnip also provides consistency in model arguments ex: random forest function w/ parsnip (arguments)

- number of sampled predictors (mtry)
- number of trees (trees)
- number of data points to split (min_n)

model arguments in parsnip are separated into two different categories: - main arguments: commonly used & available across engines - engine arguments: specific to a particular engine, can be specified in set_engine()

Using Model Results

Examining model output using extract_fit_engine(), adding vcov() will return model output as matrix

However, matrices are not super malleable/easy to change & manipulate

- the broom package in tidymodels convert all different kinds of objects to a tidy structure
  - column names are standardized across models
  - row names converted to columns

Make predictions

predict() function

- returns results in tibble format
- column names are always predictable
- as many rows in the tibble as there are in the input data

these features make the merging of predictions & original data much easier

Table 6.4: The tidymodels mapping of prediction types and column names.

type value	column name(s)
numeric	.pred
class	.pred_class
prob	.pred_{class levels}
conf_int	.pred_lower, .pred_upper
pred_int	.pred_lower, .pred_upper

other packages can be used with parsnip, list found here

parsnip_addin() will provide a list of possible models for each model mode

A model workflow

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \ldots + \hat{\beta}_p x_{ip}$$

Figure 1: Model equation

Above is the typical structural equation for a linear model, including predictors (x) and outcome variable (y).

a feature selection algorithm is mentioned to determine the minimum predictors needed for the model, wish i knew what that was lmao

model workflow: pre-processing steps, model fit and potential post-processing activities

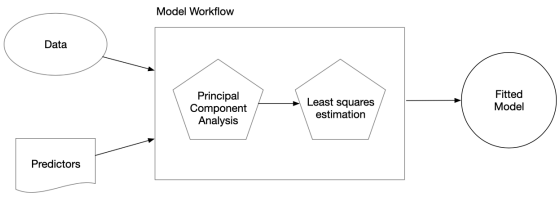


Figure 2: Correct mental model of where model estimation occurs in the data analysis process.

workflows package in R:

```
lm_wflow <-
  workflow() %>%
  add_model(lm_model)

lm_wflow
#>   Workflow
#> Preprocessor: None
#> Model: linear_reg()
#>
#>   Model
#> Linear Regression Model Specification (regression)
#>
#> Computational engine: lm
```

*with a simple model, a formula would be added as a preprocessor using add_formula()

- fit() can be used to create the model; takes created workflow & data as arguments, use predict() after on the fitted workflow
- use update_formula() to update/remove model and/or preprocessor