

Machine Learning

<http://datascience.tntlab.org>

Module 12





Today's Agenda

- How You're Already Using Machine Learning Models
- Overview of Statistical Analysis vs. Machine Learning
 - Terminology differences
 - Model selection
- Technical Walkthrough
 - Walkthrough of *caret*
 - Some starter algorithms: tree, random forest, LASSO, ridge
 - Cross-validation and model comparisons



Key to Understanding Machine Learning

- You've already learned a lot of these concepts in statistics classes.
- A lot of new terms are used for things you already have words for.
 - "Training dataset" = "dataset"
 - "Train" = "provide data to create a predictive model"
- The key to understanding machine learning is relating it back to what you already know, and extending those ideas where concepts are genuinely new



You Are Already Using Machine Learning

- Have you create an OLS linear regression model?
 - Congrats; you're a data scientist.
- You may have also used more advanced machine learning algorithms without realizing it.
 - **EM imputation**
 - Utilizes an expectation-maximization algorithm to predict missing data.



Statistical Analysis vs. Machine Learning

- **Statistical Analysis**

- Focus on interpretability
- Assumption checking (i.e., integrity of mathematical approach)
- Interpretability of component parts/predictors
- "Given this theoretical model, how well do the data describe y ?"
- **Goal:** Draw conclusions about predictors

- **Machine Learning**

- Focus on generalizable prediction
- Intention to take an algorithm developed in one context and use it in another
- "Given these data, what algorithm will predict y most consistently in other datasets with similar generative characteristics?"
- **Goal:** Predict as strongly as possible equally well in the future

- *You will see many of the same predictive modeling techniques in both.*



Types of Machine Learning (by Process)

- **Supervised Learning** (the focus in Data Camp)
 - Regression models: continuous DVs
 - Classification models: discrete DVs
 - Decision tree models
 - Neural networks
- **Semi-supervised Learning**
- **Unsupervised Learning**
 - K-Means clustering (which you might already know)
- **Reinforcement Learning**



Problems Supervised Machine Learning Solves

- When you have many predictors, overall effect estimates like R^2 will be inflated
- In psychology, we usually use adjusted- R^2 to account for this
 - Predicts the amount of shrinkage in R^2 likely to be seen due to local overfitting
 - However, adjusted- R^2 will always reveal relatively poor prediction; sometimes extremely poor, and always poorer with more predictors
- Machine learning is designed to better predict "true" variance despite the noise created by a complex predictor space
- If N is orders of magnitude larger than k , you probably don't need machine learning (or rather, it won't get you much better prediction anyway)



Problems Supervised Machine Learning Creates

- There are many ways to model the relationship between y and a set of x
 - Researcher degrees of freedom
 - Model selection: linear regression, random forest, support vector machines, etc.
 - Parameter selection: variable selection (ntbcw: parameter estimation)
 - Hyperparameter selection: configuration options for the model
 - Model selection itself might be considered a hyperparameter
 - Most models are optimized to a *loss function*; hyperparameters
 - Hyperparameters can themselves be optimized
- If you're trying to eek out every last bit of true variance, regardless of where it comes from, you're going to need to get creative
 - Comes with a distinct interpretability vs. prediction tradeoff
- Every one of these algorithms has a literature at least as if not more complex than the full courses you've taken on ANOVA or regression



Linear Regression as Machine Learning

- You probably learned 1-predictor OLS linear regression as the solution to a mathematical formula

$$Y' = b_{yx}X + a_{yx}$$

$$b_{yx} = r_{xy} \frac{\sigma_y}{\sigma_x} = \frac{N \sum XY - (\sum X \sum Y)}{N \sum X^2 - (\sum X)^2}$$

$$a_{yx} = \bar{Y} - b_{yx} \bar{X}$$

- But what if you didn't know the formulas? What would you do?
 1. Guess the value of b , predict your data, and look at the mean squared residual
 2. Change b in one direction, predict again, and see if MSR changed
 3. If it went down, change b further that way; if it didn't, go the other way
 4. Repeat until you can't get MSR any smaller
- In machine learning terms, we call the MSR for regression the "cost function"
 - Iterative procedure to find a minimum cost called *stochastic gradient descent*



Why Use *caret*?

- What is *caret*?
 - Does not actually contain any machine learning algorithms
 - Provides a common framework/syntax to access many other packages that do contain machine learning algorithms
 - Centralizes tuning of hyperparameters across algorithms
 - Automates mathematical modeling that you'd normally need to do by hand (by code)
- You need to know how to use those packages to use their functions in *caret*; each one is slightly different



Supervised Learning with *caret*

- Basic Regression

- `model <- train(
 formula,
 data,
 method="lm",
 preProcess=c("center","scale","zv"),
 trControl=trainControl(method="cv", number=10, verboselster = T)
)`

- Basic Classification

- `model <- train(
 formula,
 data,
 method="glmnet",
 preProcess=c("center","scale","zv","conditionalX"),
 trControl=trainControl(method="cv", number=10, verboselster = T,
 summaryFunction = twoClassSummary, classProbs = T)
)`



Pre-Processing: Missing Values

- Add **preProcess = ""** to *caret* syntax
- A reminder about missing values
 - NMAR: Not missing at random
 - MAR: Missing at random
 - MCAR: Missing completely at random
- Imputation of missing values
 - **Median imputation:** Assumes MCAR, so just don't
 - **K-nearest neighbors:** Assumes MAR, so use if you're comfortable with that



Pre-Processing Options

- Centering and standardizing
 - **preProcess = c("center", "scale")**
- Box-Cox transformation for non-linearity
 - **preProcess = "boxcox"**
- Listwise deletion of non-varying predictors
 - **preProcess = "zv"** # or nearly zero with "nzv"
- Run a PCA and use components as predictors instead
 - **preProcess = "pca"**
- Remove highly correlated pairs of variables (by default, > .9)
 - **preProcess = "corr"**
- In classification, remove predictors if there is no variance within a class
 - **preProcess = "conditionalX"**



Machine Learning Models and Their Hyperparameters

- To add hyperparameters to *caret*, add
 - **tuneLength=3** # number of levels for default tuning parameter
 - **tuneGrid=expand.grid()** # change many tuning parameters
- **method="ranger"** # random forests
 - **tuneLength = 10** # changes mtry, which you could set by hand
- **method="glmnet"** # ridge and LASSO regression
 - **myGrid <- expand.grid(alpha = c(0,1),** # 0 = LASSO, 1 = ridge
lambda = seq(0.0001, 0.1, length = 10)) # complexity penalties
- Find a full list of methods with **names(getModelInfo())**
- Find explanations at <http://topepo.github.io/caret/train-models-by-tag.html> including permitted hyperparameters

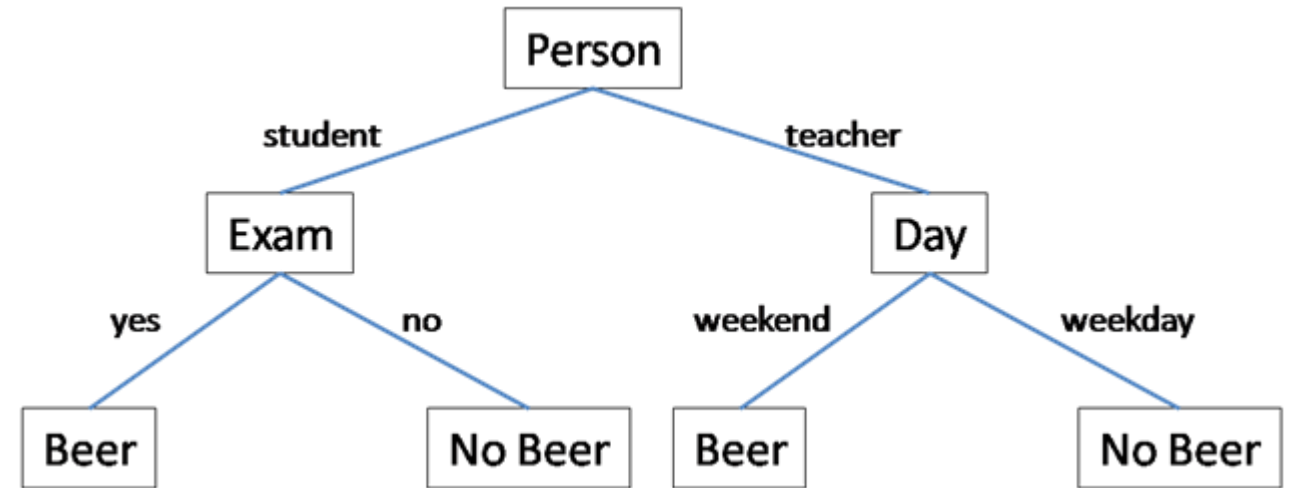
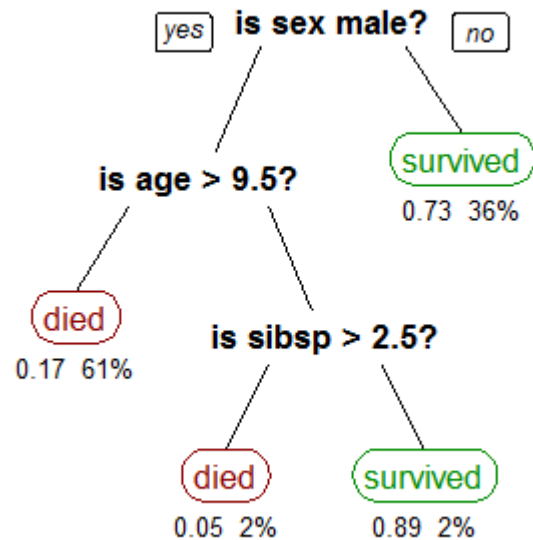


Machine Learning: Decision Trees

- Can be regression trees or classification trees (in general: CART)
- A very simple classification tree roughly works like this:
 - Look at y and each x to see which creates the two most homogeneous groups
 - This becomes the *root node*
 - For each classification created by the root node, repeat homogeneity test
 - If you get better overall prediction with the new classification, create a *split*
 - If you don't, stop following this path (i.e., this is a *leaf*)
 - Continue this process (recursively) until you hit a *stopping rule*, such as too little additional variance predicted
 - Refine the model by *pruning*, which removes leaves that don't contribute much to overall prediction
- Regression trees work similarly, but creating groups is more complex



Machine Learning: Decision Trees

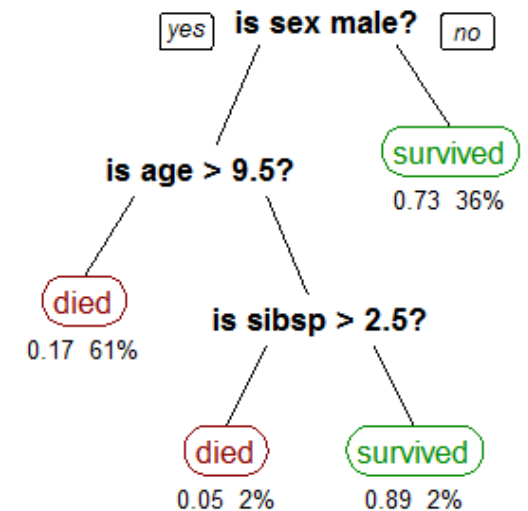


From <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>
and <https://computersciencesource.files.wordpress.com/2010/01/detresb.png>



Machine Learning: Random Forests

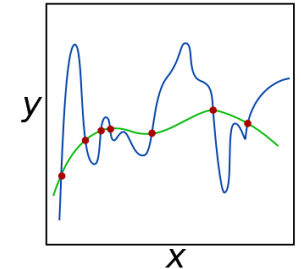
- Many decision trees; select models from random subsets of predictors to minimize the chance of overly influential cases (and thus overfitting)
- Because they involve later predictors conditional on earlier predictors, they by definition model interactions without explicit interaction terms
- Here, see an interaction between sibsp and age and sex plus their main effects
- Thus, transformations are less important here too
- Ultimately they use an "ensemble method" to combine trees; in this case, the mode





Machine Learning: LASSO/Ridge Regression

- LASSO and ridge combine automated predictor selection with regression
 - In ML, the goal is to minimize the results of the "cost" function
 - Remember: in lm , this is the residual mean square
 - In LASSO and ridge, this is the residual mean square plus a bias term
 - Gradient descent is used to create parameter estimates
 - LASSO performs "L1 regularization," in which the bias term = the sum of λ * the absolute values of the parameters, which can force zeros
 - Ridge performs "L2 regularization," in which the bias term = the sum of λ * the squared parameters, which makes all parameters shrink
 - Elastic-net performs "L1/L2 regularization", i.e., λ * the sum of both L1 & L2
- Hyperparameters
 - **alpha**: the balance between LASSO and ridge
 - **lambda**: the cost function used to either drop (LASSO) or down-weight (ridge) predictors
 - The power of ML comes from testing combinations of hyperparameters





Holdout vs. Cross Validation

- Validation does not tell you which model to choose; it gives you a generalizability estimate assuming consistent data generation.
- Holdout Validation
 - Randomly select some subset of data to use for model training (fitting) and testing (predicting). Accuracy is determined by comparing trained predictions and test values.
- k-fold Cross Validation
 - Randomly split the dataset into k datasets (folds) randomly which will be used as both training and test datasets. Each fold is used as a test set with all other folds as training sets. Accuracy is determined by comparing
 - 2-fold cross validation is not the same as holdout validation – why?
 - N-fold cross validation is also called leave-one-out cross validation



Quantifying Accuracy

- Regression
 - R^2 is easy and universal; just ask for model output (or plot)
- Classification
 - **Confusion matrices:** Like Type I and Type II errors at the case level
 - **confusionMatrix(predicted, true)**
 - Accuracy: Proportion of correct predictions
 - Sensitivity: Proportion of positive predictions out of all true positives
 - Specificity: Proportion of negative predictions out of all true negatives
 - **Receiving operating characteristic (ROC) curve**
 - **colAUC(X=predicted, y=actual, plotROC=TRUE)** from *caTools*
 - Area under the curve (AUC) ranges from 0 to 1 (ratio of Sensitivity to 1-Specificity)
- *caret* will generally select the best-performing hyperparameters for you, but you should know where they came from and what they do



Comparing Models

- If using n-fold cross validation, keep your fold composition the same within `trControl`
 - Inside a unique `trainControl()` definition that you run one time:
 - `index = createFolds(outcomevar, k = 10)`
- Use `resamples()` to compare output directly
 - `summary(resamples(list(model1, model2)))`
- Use plots to look at difference in AUC across models
 - `dotplot(resamples(list(model1, model2)), metric="ROC")`



Want to Practice Building Predictive Models?

- Many datasets to practice on here: <http://archive.ics.uci.edu/ml/index.php>
- Enter competitions on your ability to build high-quality predictive models here: <http://www.kaggle.com>
 - These competitions do not test your ability to use such models in a production environment, i.e., the real world