Regression Trees and Rule Based Models

Vishal Sood

13 May, 2015

Example Data

We will use the solubility data for illustrations. For cross-validation we need to set the folds, and a control to tell *caret::train* how to train the model. For creating data partitions caret provides functions,

1. *createDataPartition*: a series of test/training partitions
2. *createResample*: one or more bootstrap samples
3. *createFolds*: splits the data into k groups
4. *createTimeSlices*: creates cross-validation sample information to be used with time series data

To create the CV folds we will use *createFolds*, that creates 10 folds by default, returning the data as a list of vectors containing the sample positions corresponding to the data used during training. Remember that having 10 folds means that 1 of the ten folds will be held out from training for validation. The returned vectors will then contain 90% of the samples. This list will then be passed to *trainControl* to obtain a *control* object to be used with *caret::train*.

Basic Regression Trees

Packages for regression trees:

1. **rpart** splits using CART methodology, the *rpart* function, two commonly used control parameters that are used in training are the complexity parameter (cp) and maximum node depth (maxdepth), both accessible in *caret::train*. For *cp* set *method="rpart"*, for maxdepth, *method="rpart2".
2. **party** splits using conditional inference framework, the *ctree* function.

We make two CART models,

Tune the model

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

CART

951 samples 228 predictors

No pre-processing Resampling: Cross-Validated (10 fold)

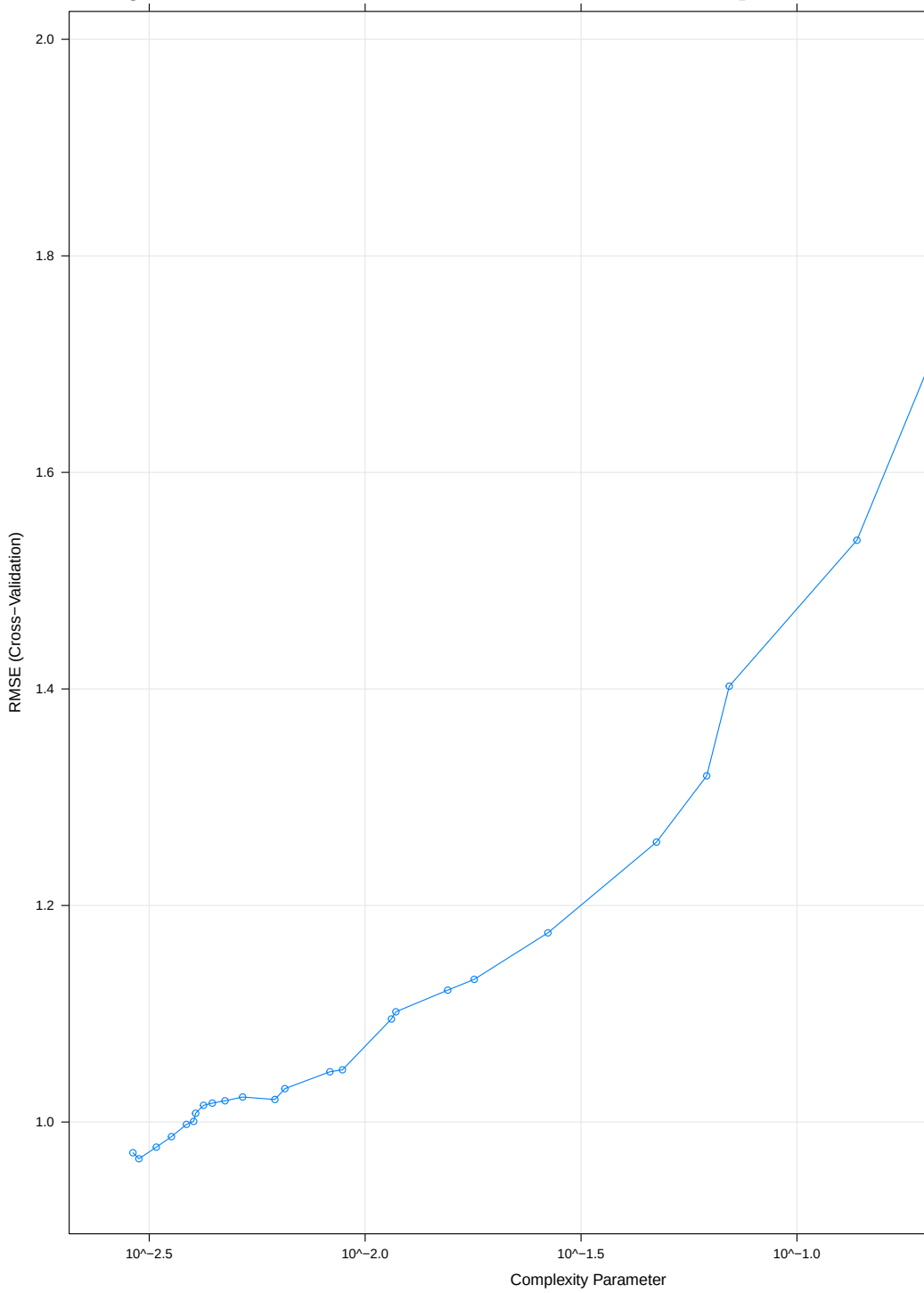
Summary of sample sizes: 856, 857, 855, 856, 856, 855, ...

Resampling results across tuning parameters:

cp RMSE Rsquared RMSE SD Rsquared SD 0.002899268 0.9716246 0.7752447 0.06541238 0.03897881 0.002992913 0.9660816 0.7779183 0.05330954 0.03562041 0.003284296 0.9767689 0.7727573 0.04958512 0.03322272 0.003560157 0.9864913 0.7681005 0.04915569 0.03456546 0.003857172 0.9978151 0.7627426 0.05749370 0.03635197 0.004007488 1.0005677 0.7614942 0.05130042 0.03486357 0.004050192 1.0080565

0.7573802 0.05214459 0.03849504 0.004224784 1.0154052 0.7534214 0.05669100 0.04255321 0.004427556 1.0174706 0.7524694 0.05439656 0.04070176 0.004737500 1.0196066 0.7514764 0.05532976 0.04411784 0.005204160 1.0230892 0.7499148 0.05409936 0.04451837 0.006181580 1.0207997 0.7495512 0.05958220 0.04837953 0.006518655 1.0308117 0.7448964 0.05745456 0.04726132 0.008288491 1.0464044 0.7369148 0.06873592 0.05097230 0.008861298 1.0482274 0.7357991 0.06448159 0.05027666 0.011501953 1.0951525 0.7125953 0.08962476 0.05412205 0.011781343 1.1017983 0.7085687 0.09360527 0.05754522 0.015535229 1.1218072 0.6972460 0.09949073 0.05991353 0.017892740 1.1317681 0.6906429 0.10606249 0.06810422 0.026503011 1.1747433 0.6680776 0.08700156 0.05948686 0.047291114 1.2585476 0.6174856 0.10318820 0.07279270 0.061802695 1.3197449 0.5802679 0.12398650 0.08602437 0.069715098 1.4025314 0.5265519 0.12300733 0.08459634 0.137700138 1.5373482 0.4375590 0.19960327 0.09333503 0.373005064 1.9564323 0.2781173 0.19891634 0.00920422

RMSE was used to select the optimal model using the smallest value. The final value used for the model was cp



= 0.002992913. And plot the tuning
Variable importance

rpart variable importance

only 20 most important variables shown (out of 228)

```
Overall
```

NumNonHBonds 0.9070 SurfaceArea2 0.7564 NumMultBonds 0.7453 NumCarbon 0.7257 MolWeight 0.6693 NumNonHAtoms 0.6093 NumBonds 0.5871 FP116 0.4788 NumOxygen 0.4706 NumRotBonds 0.3794 SurfaceArea1 0.3554 NumHydrogen 0.3522 FP081 0.2594 FP075 0.2435 FP077 0.1992 HydrophilicFactor 0.1164 FP203 0.0000 NumRings 0.0000 FP107 0.0000 FP053 0.0000 Test the model on the test-data

Conditional inference tree

Conditional Inference Tree

951 samples 228 predictors

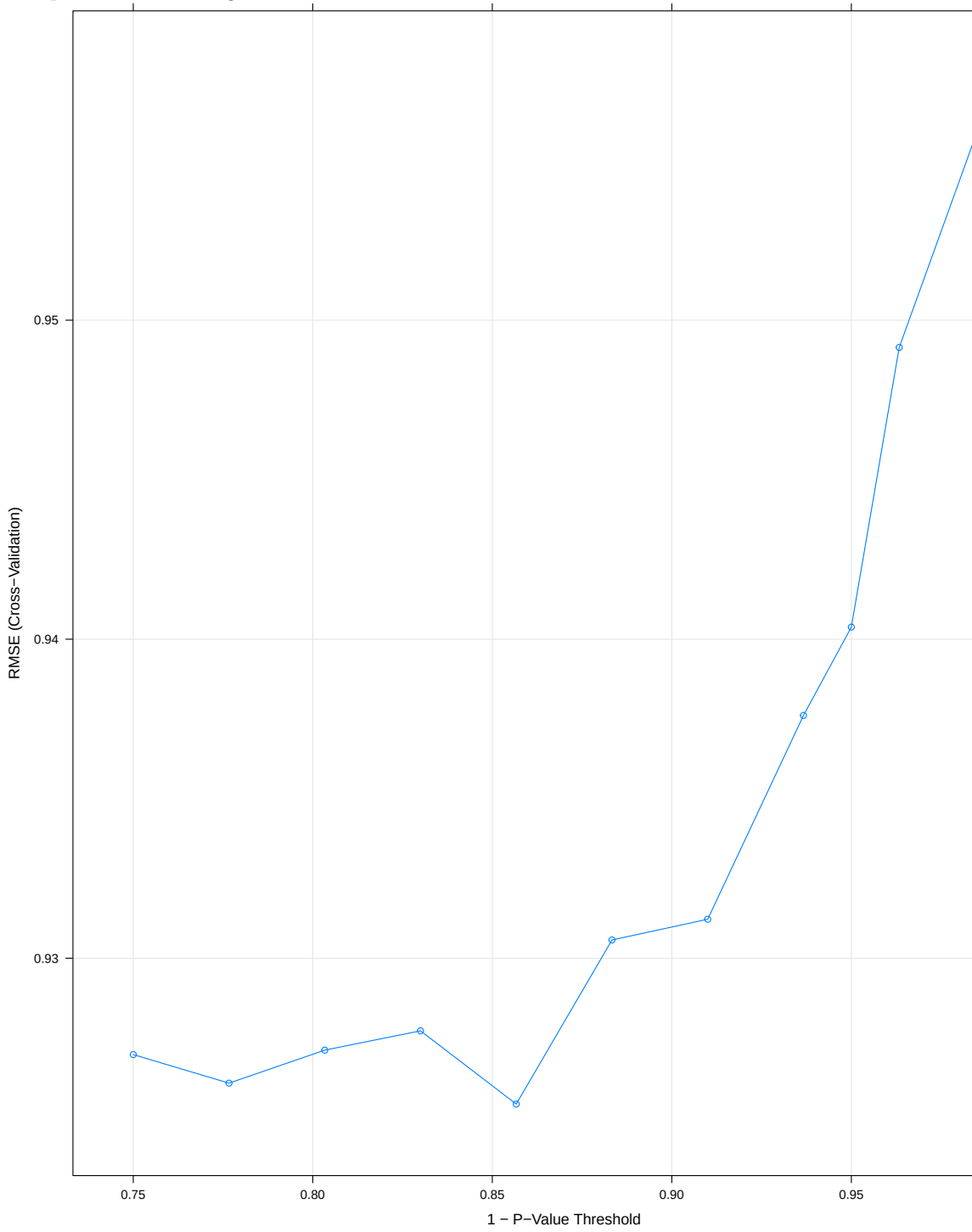
No pre-processing Resampling: Cross-Validated (10 fold)

Summary of sample sizes: 856, 857, 855, 856, 856, 855, ...

Resampling results across tuning parameters:

mincriterion RMSE Rsquared RMSE SD Rsquared SD 0.7500000 0.9269810 0.7903849 0.1072444 0.06710406 0.7766667 0.9260851 0.7905803 0.1103460 0.06810741 0.8033333 0.9271195 0.7901073 0.1108398 0.06829510 0.8300000 0.9277246 0.7906922 0.1049071 0.06333440 0.8566667 0.9254289 0.7913914 0.1115011 0.06598342 0.8833333 0.9305731 0.7892532 0.1120570 0.06569786 0.9100000 0.9312221 0.7891169 0.1066836 0.06480620 0.9366667 0.9376111 0.7860495 0.1047474 0.06470085 0.9500000 0.9403788 0.7847694 0.1038905 0.06301081 0.9633333 0.9491457 0.7806482 0.1075643 0.06510664 0.9900000 0.9574527 0.7767184 0.1105580 0.07000131

RMSE was used to select the optimal model using the smallest value. The final value used for the model was



mincriterion = 0.8566667.

Regression Model Trees

Rule-Based Models

Bagged Trees

Random Forests

Tree Correlation (Hastie et al 2008): If we start with a sufficiently large number of original samples and a relationship between predictors and response that can be adequately modeled by a tree, then trees from different bootstrap samples may have similar structures to each other (especially at the top of the trees) due to the underlying relationship. This can prevent bagging from optimally reducing variance of the predicted values.

So variance can be further reduced, Breiman's Random Forest:

1. m <- NumberOfModelsToBuild
2. for i = 1 to m do
3. Generate a bootstrap sample of the original data
4. Train a tree model on this sample
5. for each split do
6. `Randomly select k (< P) of the original predictors`
7. `Select the best predictor among the k predictors and partition the data`
8. end
9. Use typical tree model stopping critera to determine when a tree is complete (no pruning)
10. end

Tuning parameter: number of models to build.

Breiman: linear combination of many independent learners reduces the variance of the overall ensemble relative to any individual learners. Random forest achieves this by selecting strong, complex learners that exhibit low bias. Robust to noisy response, however the independence of learners can underfit data when the response is not noisy.

CART or conditional inference trees can be used as the base learner. Tuning parameter does not have a drastic effect on performance. Cross-validation RMSE can be very similar to out-of-bag error estimate.

Can we understand the relationship between the predictors and the response?

Breiman: Randomly permute the values of each predictor for the oob sample of one predictor at a time for each tree. The difference in predictive performance between the non-permuted sample and the permuted sample for each predictor is recorded and aggregated across the entire forest.

Or measure the improvement in node purity based on the performance metric for each predictor at each occurence of that predictor across the forest, aggregate across the forest to determine the overall importance for the predictor.

Problems with collinearty in determining predictor importances. Dillution effects.

Boosting

Originally for classification. From AdaBoost for classification to Friedman's stochastic gradient boosting machine. From AdaBoost for classification to Friedman's stochastic gradient boosting machine. From AdaBoost for classification to Friedman's stochastic gradient boosting machine.

1990s, Schapire 1990, 1999; Freund 1995, influence by learning theory. Weak classifiers are combined (boosted) to produce an ensemble classifier with a superior generalized misclassification error rate. AdaBoost is implementation (Schapire 1999)

Used widely with applications in gene expression (Dudoit 2002, Ben-Dor 2000), chemometrics (Varmuze 2003), music genre identification (Bergstra 2006).

(Friedman 2000): AdaBoost to statistical concepts of loss functions, additive modeling, and logistic regression. Boosting can be interpreted as a forward stagewise additive model that minimizes exponential loss. Resulted in a simple, elegant, and highly adaptable algorithm for different kinds of problems (Friedman 2001): gradient boosting machines. Both regression, classification.

Gradient Boosting Machines: given a loss function, and a weak learner, finds an additive model that minimized the loss function. Initialize with the best guess of the response. Calculate the gradient (eg residual), fit a model to the residuals to minimize the loss. Add the current model to the previous model. Iterate.

With trees as base learners, two tuning parameters: tree depth (interaction depth), number of iterations. Tree depth is also called interaction depth: think of each subsequential split as a higher-level interaction term with all of the other previous split predictors.

With squared error as the loss function, . let D = treeDepth . let K = numIterations . let L = learningRate . iteration(observed, currentPred) = currentPred + L*updatePred(observed, currentPred) . updatePred(residuals currentPred) = predict(T) where T = regressionTree(depth=D, response=residuals(observed, currentPred))

Difference from random forests: dependent on past trees, have minimum depth, and contribute unequally to the final model.

Can be susceptible to over-fitting. Learner optimally fits the gradient, so Boosting will select the optimal learner at each stage of the algorithm. The greedy strategy may not find the optimal global model, and may over-fit the training data. So regularization through λ, the learning rate. (Ridgeway 2007)

Friedman's **stochastic gradient boosting**: update using a randomly sampled fraction of the training data (around 0.5), the fraction becomes another tuning parameter.

Variable importance: function of the reduction in squared error – due to each predictor is summed within each tree in the ensemble. Averaged