Predictive Modeling with Caret

Mochan Shrestha

September 10, 2015

Standard Process for Data Mining

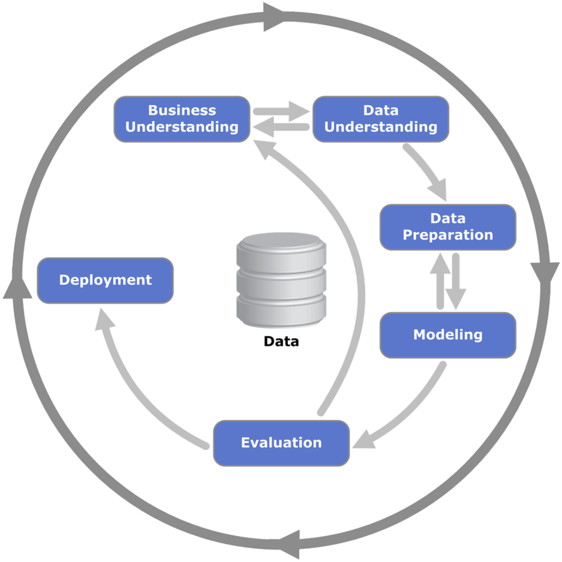


Figure: Cross Industry Standard Process for Data Mining

Data

- Caret works with data frames.
- We will use data frames already included in R packages.

Classification Data : Image Segmentation

- Classification data from caret package
- 2 factor Class PS(Poorly Segmented), WS(Well Segmented)

```
> data(segmentationData}
> training = subset(segmentationData, Case=="Train")
> testing = subset(segmentationData, Case=="Test")
```

- Data is about the quality of image segmentation of cells. For more information, use

```
> help("segmentationData")
```

Classification Data : Telecom Churn

Account information and two state churn - yes or no

```
> library(C50)
> data(churn)
```

Two data-frames, `churnTrain` and `churnTest`.

Regression Data : Kelly Blue Book Resale Data

Resale value data for cars in dollars: 2005 GM cars

```
> library(caret);
> data(cars);
```

Regression Data : Melting Point Data

Model the melting point of compounds from its chemical descriptors

```
> library(QSARdata)
> data(MeltingPoint)
> training = cbind(MP_Descriptors[MP_Data=="Train",],
  MP_Outcome[MP_Data=="Train"])
> testing = cbind(MP_Descriptors[MP_Data=="Test",],
  MP_Outcome[MP_Data=="Test"])
> colnames(training) = c(colnames(MP_Descriptors),
  "MeltingPt")
> colnames(testing) = c(colnames(MP_Descriptors),
  "MeltingPt")
```

Regression Data : Wage Data

- Wage data of workers by age, education, jobclass etc

```
> library(ISLR)
> data(Wage)
```

- For more information, use

```
> help(Wage)
```

Classification Data : Spam E-mail

Spam or not spam. Variables with frequency of words in the e-mail.

```
> library(kernlab)
> data(spam)
> help(spam)
```

Predictive Modeling Workflow

- Cleaning up the Predictors
- Test/Training data split
- Model Training, Tuning and Visualizations
- Evaluating Performance of Final Model

Example : Regression

Regression with Wage Data

Test-Training data split : Spending Data

- Spend data into **Training** and **Testing** sets
- Create a random split of the data so that 75% is in training and 25% in testing

```
> trainIndex = createDataPartition(y, p = .75,
  list = FALSE, times = 1)
```

- Select the data

```
> training = Wage[inTrain,];
> testing = Wage[-inTrain,];
```

Training a Linear Regression

- Create a model using the `train` function
- We predict wage. Predictors we are using are age, job-class and education
- Method is `lm` which is linear regression
- Returns an object that contains the model information for prediction

```
> modFit = train(wage ~ age + jobclass + education,
  method="lm", data=training);
```

Predicting the Test Set

- Apply the model from `train` to the testing data set.
- Returns the predicted values.

```
> pval = predict(modFit, testing);
```

Evaluating the Model

- Root mean squared error (RMSE):

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(Prediction - Truth)^2}$$

```
> RMSE(predict(modFit2, testing), testing$wage)
```

- R squared (R^2) : square of the correlation coefficient between the original and predicted values.

```
> R2(predict(modFit2, testing), testing$wage)
```

Using Neural Networks as the Model

- Bayesian Regularized Neural Networks
- Just change the method field

```
> modFit2 = train(wage ~ age + jobclass + education,
  method="brnn", data=training);
```

The list of methods for train are
http://topepo.github.io/caret/modelList.html

Tuning Parameters

- `brnn` lists a tuning parameter called `neurons`
- When there are tuning parameters, `train` runs multiple models to find the optimal set of parameters.

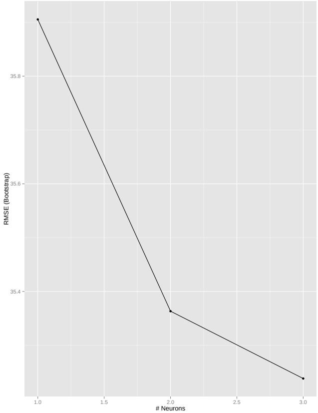
```
Define sets of model parameter values to evaluate;
for each parameter set do
    for each resampling iteration do
        Hold-out specific samples;
        Fit the model on the remainder;
        Predict the hold-out samples;
    end
    Calculate the average performance across hold-out predictions;
end
Determine the optimal parameter set;
```

Tuning Grid

- We can visualize the optimal parameter search using

```
> ggplot(modFit2);
```

- Used RMSE for the performance
- Uses bootstrapping for resampling
- "# Neurons" is the model parameter



Setting the Tuning Grid

- We can define a custom grid for the searching the optimal parameter
- `neurons` are searched in $\{1, 3, 5, 7, 9\}$

```
> grid = expand.grid(neurons = seq(1,10,by=2));
> modFit2 = train(wage ~ age + jobclass + education,
  method="brnn", data=training,
  tuneGrid = grid, verbose=FALSE);
```

K-Nearest Neighbors

- Use the training algorithm `knn` for k-nearest neighbor

```
> grid2 = expand.grid(k = seq(1,20,by=2));
> modFit3 = train(wage ~ age + jobclass + education,
  method="knn", data=training,
  tuneGrid = grid2, verbose=FALSE);
```

Pre-processing : Near Zero Variance

Remove predictors that are near zero in variance.

```
> nzcols = nearZeroVar(training);
> training = training[,-nzcols];
> testing = testing[,-nzcols];
```

Pre-processing : Correlated Predictors

Remove correlated predictors with correlation greater than 0.9

```
> descrCorr = cor(training);
> highCorr = findCorrelation(descrCorr, 0.90);
> training = training[,-highCorr];
> testing = testing[,-highCorr];
```

Pre-processing : Linearly Dependent Predictors

Find and remove linear combinations in the columns of the matrix

```
> comboInfo = findLinearCombos(training);
> training = training[,-comboInfo$remove];
> testing = testing[,-comboInfo$remove];
```

Pre-Processing : Scaling, Centering, Box-Cox Transformation

Can be added to the training function under the preProcess option.

```
> modFit = train(wage ~ age + jobclass + education,
  method="lm",
  preProcess = c("center", "scale", "YeoJohnson"),
  data=training);
```

Example : Classification

Regression with Image Segmentation Data

1. Training and testing data sets are made the same way
2. `Train` and `predict` also work the same way

Decision Trees : CART

Decision trees with single tuning parameter cp, complexity parameter.

```
> modelFit = train(Class ~ ., data=training,
  method="rpart",
  tuneLength=10);
```

- `tuneLength` is the number of points in the tuning parameter. Can see in the plot
- Note the performance is measured as accuracy (instances of correct classification)

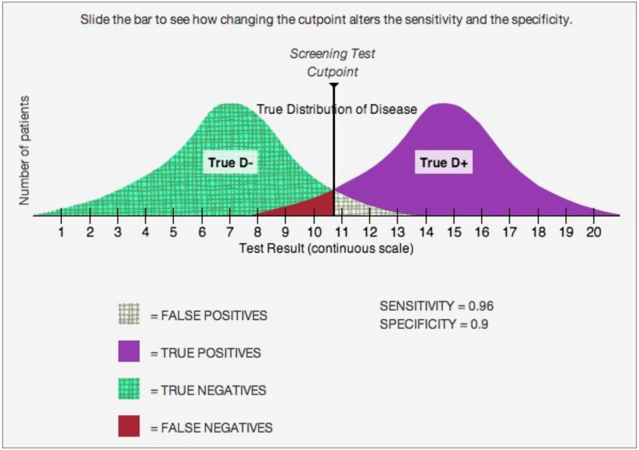
Confusion Matrix

- `predict` also works exactly the same way.
- With classification, we get the confusion Matrix and other statistics.

```
> rpartPred = predict(modelFit, testing);
> confusionMatrix(rpartPred, testing$Class);
```

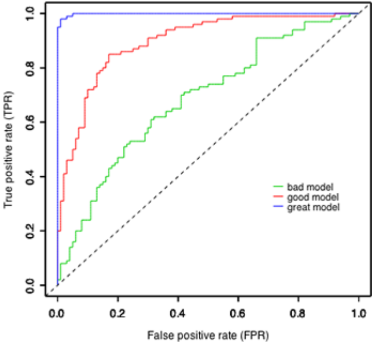
Sensitivity, Specificity

- `Sensitivity`: proportion of positives identified as positives
- `Specificity`: proportion of negatives identified as negatives



ROC

- The graph of sensitivity vs specificity for various cutoffs is called an ROC (receiver operating characteristic).
- Another metric is the area under the ROC curve



Training with ROC metric

- We do custom parameter tuning through `trainControl` so that we can get class probabilities for ROC metric (`classProbs = TRUE`).
- We use 3-fold cross validation (`repeatedcv` and `repeats = 3`).
- Method can be `boot` if we want bootstrapping.

```
> cvCtrl = trainControl(method = "repeatedcv",
  repeats = 3, summaryFunction = twoClassSummary,
  classProbs = TRUE);
> modelFit = train(Class ~ ., data=training,
  method="rpart", tuneLength=10,
  metric="ROC", trControl=cvCtrl);
```

Plot the model ROC

- Get the predictions as probabilities
- Plot the ROC for the testing set

```
> rpartPredp = predict(modelFit, testing,
  type="prob");
> library(pROC);
> rpartROC = roc(testing$Class, rpartPredp[,"PS"],
  levels=rev(levels(testing$Class)))
> plot(rpartROC)
```

Support Vector Machines

- Support vector machines training with radial kernel function

```
> svmTune = train(x=trainX, y=training$Class,
  method="svmRadial", tuneLength=9,
  preProc = c("center","scale"),
> metric="ROC", trControl = cvCtrl);
```