Tree-based Regression

葉建華

jhyeh@mail.au.edu.tw

http://jhyeh.csie.au.edu.tw/

The Problem

- We know there are many nonlinearities in real life. How can we expect to model everything with a global linear model?
  - Build a model for our data is to subdivide the data into sections that can be modeled easily
  - These partitions can then be modeled with linear regression
  - If we first partition the data and the results don't fit a linear model, then we can partition the partitions
  - Trees and recursion are useful tools for this sort of portioning
- CART: Classification And Regression Trees

Tree-based Regression

Tree-based regression

Pros: Fits complex, nonlinear data

Cons: Difficult to interpret results

Works with: Numeric values, nominal values

Decision Trees

- The algorithm we used to construct trees was ID3
  - Chooses the best feature on which to split the data and then splits the data into all possible values that the feature can take
- Problem of ID3
  - This type of splitting separates the data too quickly
  - Couldn't directly handle continuous features
- CART
  - Makes binary splits
  - Handles continuous variables

General Approach

General approach to tree-based regression

1. Collect: Any method.
2. Prepare: Numeric values are needed. If you have nominal values, it's a good idea to map them into binary values.
3. Analyze: We'll visualize the data in two-dimensional plots and generate trees as dictionaries.
4. Train: The majority of the time will be spent building trees with models at the leaf nodes.
5. Test: We'll use the R^2 value with test data to determine the quality of our models.
6. Use: We'll use our trees to make forecasts. We can do almost anything with these results.

Tree Node Structure

```
class treeNode():
    def __init__(self, feat, val, right, left):
        featureToSplitOn = feat
        valueOfSplit = val
        rightBranch = right
        leftBranch = left
```

Pseudo Code

Find the best feature to split on:

If we can't split the data, this node becomes a leaf node

Make a binary split of the data

Call createTree() on the right split of the data

Call createTree() on the left split of the data

Tree-building Code

Listing 9.1 CART tree-building code

```
from numpy import *

def loadDataSet(fileName):
    dataMat = []
    fr = open(fileName)
    for line in fr.readlines():
        curLine = line.strip().split('\t')
        fltLine = map(float,curLine)
        dataMat.append(fltLine)
    return dataMat

def binSplitDataSet(dataSet, feature, value):
    mat0 = dataSet[nonzero(dataSet[:,feature] > value)[0],:][0]
    mat1 = dataSet[nonzero(dataSet[:,feature] <= value)[0],:][0]
    return mat0,mat1

def createTree(dataSet, leafType=regLeaf, errType=regErr, ops=(1,4)):
    feat, val = chooseBestSplit(dataSet, leafType, errType, ops)
    if feat == None: return val
    retTree = {}
    retTree['spInd'] = feat
    retTree['spVal'] = val
    lSet, rSet = binSplitDataSet(dataSet, feat, val)
    retTree['left'] = createTree(lSet, leafType, errType, ops)
    retTree['right'] = createTree(rSet, leafType, errType, ops)
    return retTree
```

1 Map everything to float()

2 Return leaf value if stopping condition met

Tree-building Code

Listing 9.2 Regression tree split function

```
def regLeaf(dataSet):
    return mean(dataSet[:,-1])

def regErr(dataSet):
    return var(dataSet[:,-1]) * shape(dataSet)[0]

def chooseBestSplit(dataSet, leafType=regLeaf, errType=regErr, ops=(1,4)):
    tolS = ops[0]; tolN = ops[1]
    if len(set(dataSet[:,-1].T.tolist()[0])) == 1:
        return None, leafType(dataSet)
    m,n = shape(dataSet)
    S = errType(dataSet)
    bestS = inf; bestIndex = 0; bestValue = 0
    for featIndex in range(n-1):
        for splitVal in set(dataSet[:,featIndex]):
            mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
            if (shape(mat0)[0] < tolN) or (shape(mat1)[0] < tolN): continue
            newS = errType(mat0) + errType(mat1)
            if newS < bestS:
                bestIndex = featIndex
                bestValue = splitVal
                bestS = newS
    if (S - bestS) < tolS:
        return None, leafType(dataSet)
    mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
    if (shape(mat0)[0] < tolN) or (shape(mat1)[0] < tolN):
        return None, leafType(dataSet)
    return bestIndex,bestValue
```

1. Exit if all values are equal
2. Exit if low error reduction
3. Exit if split creates small dataset

Tree Pruning

- Trees with too many nodes are an example of a model overfit
- The procedure of reducing the complexity of a decision tree to avoid overfitting is known as *pruning*
- Postpruning
  - Use a test set to prune the tree
  - Use no user-defined parameters
  - Need to split data into test and training set

Pseudo Code

Split the test data for the given tree:

If the either split is a tree: call prune on that split

Calculate the error associated with merging two leaf nodes

Calculate the error without merging

If merging results in lower error then merge the leaf nodes

Tree Pruning Function

Listing 9.3 Regression tree-pruning functions

```
def isTree(obj):
    return (type(obj).__name__=='dict')

def getMean(tree):
    if isTree(tree['right']): tree['right'] = getMean(tree['right'])
    if isTree(tree['left']): tree['left'] = getMean(tree['left'])
    return (tree['left']+tree['right'])/2.0

def prune(tree, testData):
    if shape(testData)[0] == 0: return getMean(tree)
    if (isTree(tree['right']) or isTree(tree['left'])):
        lSet, rSet = binSplitDataSet(testData, tree['spInd'],
                     tree['spVal'])
    if isTree(tree['left']): tree['left'] = prune(tree['left'], lSet)
    if isTree(tree['right']): tree['right'] =  prune(tree['right'], rSet)
    if not isTree(tree['left']) and not isTree(tree['right']):
        lSet, rSet = binSplitDataSet(testData, tree['spInd'],
                     tree['spVal'])
        errorNoMerge = sum(power(lSet[:,-1] - tree['left'],2)) +\
            sum(power(rSet[:,-1] - tree['right'],2))
        treeMean = (tree['left']+tree['right'])/2.0
        errorMerge = sum(power(testData[:,-1] - treeMean,2))
        if errorMerge < errorNoMerge:
            print "merging"
            return treeMean
        else: return tree
    else: return tree
```

1 **Collapse tree if no test data**

Summary

- Regression is the process of predicting a target value similar to classification
  - Ridge regression is an example of a shrinkage method
- Another shrinkage method that's powerful is the lasso
- The lasso is difficult to compute, but stagewise linear regression is easy to compute and gives results close to those of the lasso
- Shrinkage methods can also be viewed as adding bias to a model and reducing the variance