CART in Haskell

Genji Ohara

June 13, 2023

1 Preamble

```
import Numeric.LinearAlgebra
import Prelude hiding ((<>))
import Text.ParserCombinators.Parsec
import Data.CSV

import DataSet
```

2 Data Type Definition

2.1 Data Space

Feature Space	$\mathcal{F} = \mathbb{R}^D$
Label Space	$\mathcal{L} = \{0, 1, \ldots, L-1\}$
Data Space	$\mathcal{D} = \mathcal{F} \times \mathcal{L}$

```
type DataSet = [DataPoint]
```

2.2 Data Space

```
data Literal = Literal{lFeatureIdx :: Int, lValue :: Double} deriving Show

data Split = Split {sLiteral :: Literal, sScore :: Double} deriving Show

instance Eq Split where
    (Split l s) == (Split l' s') = (Prelude.==) s s'

instance Ord Split where
    (Split l s) <= (Split l' s') = (Prelude.<=) s s'

data Tree = Leaf {label :: Int} |
            Node {literal :: Literal, left :: Tree, right :: Tree}
            deriving Show
```

3 Gini Impurity

$$\mathrm{Gini} : \mathcal{L}^n \to \mathbb{R}$$
$$\mathrm{Gini}(L) = 1 - \sum_{i=0}^{L-1} p_i(L)^2$$
$$p_i(L) = \frac{1}{|L|} \sum_{l \in L} \mathbb{I}[l = i]$$

```
gini :: [Label] -> Double
gini labels = 1.0 - (sum $ map (^ 2) $ pList labels)

pList :: [Label] -> [Double]
pList labels = map (/ labelSetSize) $ map fromIntegral $ cntList labels 0
    where labelSetSize = fromIntegral $ length labels

cntList :: [Label] -> Int -> [Int]
cntList labels trg =
    if trg == labelNum
        then []
        else [length $ filter (== trg) labels] ++ (cntList labels $ trg + 1)
```

4 Split Data

$$D_l(i,v) = \{(\boldsymbol{x}, y) \in D \mid x_i < v\}$$
$$D_r(i,v) = \{(\boldsymbol{x}, y) \in D \mid x_i \geq v\}$$

```
splitData :: DataSet → Literal → [DataSet]
splitData dataSet literal = [lData, rData]
    where
        lData = [data | data ← dataSet, (dFeature data) !! i <  v]
        rData = [data | data ← dataSet, (dFeature data) !! i ≥ v]
        i = lFeatureIdx literal
        v = lValue literal

scoreLiteral :: DataSet → Literal → Split
scoreLiteral dataSet literal = Split literal score
    where
        score = sum $ map (λx → (gini $ map dLabel x) ∗ (fromIntegral $ length x)
            / dataSize) splittedData
        dataSize = fromIntegral $ length dataSet
        splittedData = splitData dataSet literal

bestSplitAtGivenFeature :: DataSet → Int → Split
bestSplitAtGivenFeature dataSet featureIdx = maximum splitList
    where
        splitList = map (scoreLiteral dataSet) literalList :: [Split]
        literalList = map (Literal featureIdx) $ valueList
        valueList = map (λx → (dFeature x) !! featureIdx) dataSet

bestSplit :: DataSet → Split
bestSplit dataSet = maximum $ map (bestSplitAtGivenFeature dataSet) [0,1..
    featureNum-1]
```

5 Main

```
main = do
    rawDataSet ← parseFromFile csvFile "../data/iris/iris.data"
    let dataSet = either (λx → []) processData rawDataSet
    print $ bestSplit dataSet
```