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

$$\mathrm{Gini}(D) = 1 - \sum_{l=0}^{L-1} p_l(D)^2$$

$$p_l(D) = \frac{1}{|D|} \sum_{(x,y)\in D} \mathbb{I}[y = l]$$

```
gini :: DataSet → Double
gini points = 1.0 - (sum $ map (^ 2) pList)
    where
        pList       = map (/ dataSize) cntList
        dataSize    = fromIntegral $ length points
        cntList     = map fromIntegral
            [length $ filter (λx → (Prelude.==) (dLabel x) 0) points,
            length $ filter (λx → (Prelude.==) (dLabel x) 1) points,
            length $ filter (λx → (Prelude.==) (dLabel x) 2) points]
```

4 Split Data

```
splitData :: DataSet → Literal → [DataSet]
splitData dataSet literal = [lData, rData]
    where
        lCondition = λx → ((dFeature x) !! lFeatureIdx literal) <   (lValue
            literal)
        rCondition = λx → ((dFeature x) !! lFeatureIdx literal) ≥   (lValue
            literal)
        lData = filter lCondition dataSet
        rData = filter rCondition dataSet

scoreLiteral :: DataSet → Literal → Split
scoreLiteral dataSet literal = Split literal score
    where
        score = sum $ map (λx → (gini 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
```