Big Data Processing 2019-2020 Project Assignment

Implementing Decision Tree Learning in Spark

Joeri De Koster (Joeri.De.Koster@vub.be), Matteo Marra (Matteo.Marra@vub.be)

As a practical assignment, students are required to complete the project assignment described in this document. As part of this project assignment students are required to implement the ID3[1] decision tree learning algorithm in Spark. The implementation is required to work on our cluster configuration set up at the Software Languages Lab (Isabelle). The students are required to detail their findings in a report of up to 5 pages. The oral exam is an oral examination of the theory as well as an oral defence of the project. The project is to be completed individually by each student and submitted to Canvas.

Scenario

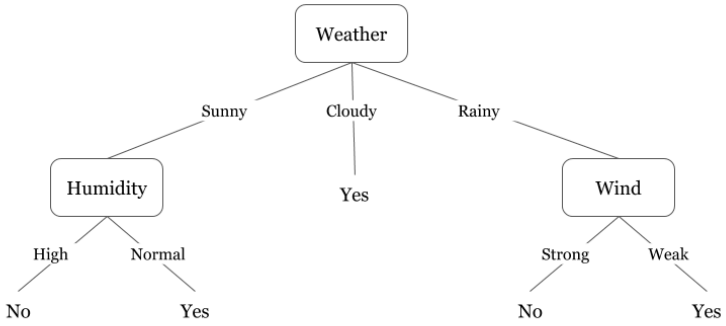
Prof. De Koster recently bought some items on Amazon and after having received the products he now wants to write a few helpful reviews. He is convinced he can provide some valuable feedback. However, other than the content of the accompanying text, he has no idea what the important qualities are of a helpful review. In order to answer that question he downloaded a dataset of amazon reviews and stored them in the following folder on Isabelle: `/data/amazon/`. This dataset is a collection of tab-separated text files where every line encodes a single review. More details on the information in each column can be found here.

In order to identify what other qualities a helpful review has, Prof. De Koster asks you to implement a decision tree learning algorithm (ID3) in Spark. The next section will give you some details on how ID3 works. The description in tasks 1 and 2 will give you some more information on how to apply this algorithm to our specific scenario.

[1] Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81–106

Decision Tree Learning

Decision Tree Learning is a general, predictive modelling technique that can be applied to classify datapoints based on a number of decision rules that split the data according to the value of their features. In their most basic form, decision rules operate on attributes with discrete values. As such, decision rules generally take the form of if-then-else statements.



Above is an example of a decision tree to decide based on the weather conditions whether we want to play outside or not. Finding the decision tree that optimises the accuracy of the model is an NP-complete problem. Therefore, decision tree learning algorithms typically do not guarantee an optimal solution. The basic algorithm used in decision trees is known as the ID3 algorithm. This algorithm gradually builds a decision tree starting from an empty tree with a single root node. It then uses a greedy strategy by iteratively selecting the best attribute to split the dataset. Each time an attribute is selected, a new branch for each value of that attribute is then added to the tree. In each iteration of the algorithm, a new attribute is selected and a new level is added to the tree until the tree correctly classifies all data points for some accuracy threshold.

In the above example, weather is the best discriminator for deciding whether we play outside or not so it starts as the root node of our tree. In the case the weather is Cloudy, all training data give a positive results, so for that branch we don't need to add additional levels to the tree. In the case the weather is Sunny, the best discriminator for splitting our dataset is Humidity as it correctly classifies all of those training examples. Similarly, in the case the weather is Rainy, the best discriminator for splitting our dataset is Wind as it correctly classifies all of those training examples.

Pseudocode for the algorithm is given below[2]:

```
type Dataset = RDD[Attributes]
type AttributeIdx = Int
def ID3(data: Dataset, target: AttributeId, attributes: Array[AttributeId]): DecisionTree = {
  // If one of three conditions hold, stop recursing:
  //   1) The set of examples that need to be classified is smaller than some threshold
  //      (this prevents overfitting)
  //   2) The list of predictive attributes is empty
  //   3) All examples in data have the same value for the target attribute (zero entropy)
  // Return a single-node tree with as a label the most common value of the target attribute.
  if (data.count < THRESHOLD || attributes.isEmpty || H(data, target) == 0) {
    return new leafTree(mostCommonValue(data, target))
  } else {
    // Find the attribute that best classifies the data (maximum information gain)
    val ig = attributes.map(IG(data, target, _))
    val A = attributes.find(IG(data, target, _) == ig.max)
    // Construct a new decision tree with root attribute A
    val tree = new decisionTree(A)
    // For every possible class/value v of A, add a new branch with a subtree that classifies
    // all datapoints in data for which attribute A has value v.
    for (v <- possibleValues(A)) {
       val filtered = data.filter(_.getAttribute(A) == v)
       val subtree = ID3(filtered, target, attributes.filter(_ != A))
       tree.addSubTree(v, subtree)
    }
    return tree
  }
}
```

The entropy, $\mathrm{H}(S)$, is a measure for the amount of uncertainty for the values of the target attributes in S. The value of $\mathrm{H}(S)$ is given by the following formulae:

$$\mathrm{H}(S) = \sum_{x \in X} -p(x)\log_2 p(x)$$

Where:

- S is the dataset of target attributes for which the entropy is being calculated.
- X is the set of classes (or values in the case of discrete attributes) of the target attribute.
- $p(x)$ is the proportion of the number of elements in class x to the total number of elements in S.

[2] Please note that any skeleton code that is included in this document serves a purely illustrative purpose and you are not forced to follow it. For example, all of the code assumes an implementation using RDDs while you are also allowed to use Dataframes or Datasets.

The information gain, $\mathrm{IG}(S, A)$, is a measure for the difference in entropy before and after the set S is split according to attribute A. In other words, this value gives a measure for how well the resulting decision tree would classify our target attribute if we add branches for every possible class or value of attribute A to the root node. The value of $\mathrm{IG}(S, A)$ is given by the following formulae:

$$\mathrm{IG}(S, A) = \mathrm{H}(S) - \sum_{t \in T} p(t)\mathrm{H}(t)$$

Where:

- T the subsets created from splitting S by attribute A such that $S = \bigcup_{t \in T} t$
- $p(t)$ is the proportion of elements in t to the number of elements in S
- $\mathrm{H}(S)$ and $\mathrm{H}(t)$ are the entropy of the entire dataset S and the subset t respectively.

Task 1: Feature Selection & Categorical Attributes

Feature Selection. Your first job is selecting around 4~5 features from the dataset that could potentially help with classifying whether a review is helpful or not. For example, whether or not a review came from a user that made a verified purchase might be a good indicator of how much trust we put in that review.

In order to complete this first step you are required to implement the `parseReview` method in the following code snippet.

```
case class Review(verifiedPurchase: Boolean,
                  starRating: Int,
                  body, String,
                  ???,
                  helpfulVotes: Int)

def parseReview(review: String): Review = ???
val reviews = sc.textFile("/data/amazon/*").map(parseReview)
```

Categorical Attributes. Decision trees are constructed from discrete or categorical values. This means your next job is to transform any attribute that has continuous values into a categorical values. For example, the length of the review body might be a good indicator of how helpful a review is. Too little text could imply too little information for the reader of the review, while too much text might discourage readers from reading the review. However, the total length of the review body is a continuous value. In order to transform this value we might replace the length of the review with the quartile it belongs to (comparing it with all other review lengths). Also note that the target attribute (number of helpful votes) is also not a categorial value and will need to be transformed as well.

The following code snippet gives some example skeleton code on how this could be implemented.

```
abstract class Attribute[T] {
    def possibleValues(): Array[T]
}
class Rating(val n: Int) extends Attribute[Int] {
    def possibleValues() = Array(1,2,3,4,5)
}
case class Attributes(verifiedPurchase: ???,
                      starRating: Rating,
                      body: ???
                      ???,
                      helpfulVotes: ???)

def extractAttributes(reviews: RDD[Review]): RDD[Attributes] = ???
val reviewAttributes = extractAttributes(reviews)
```

Task 2: Decision Tree Learning (ID3)

Finally, in this task, we can start with the implementation of our decision tree learning algorithm. First we need to implement a function to calculate the entropy of our dataset given a certain target attribute:

```
def H(data: RDD[Attributes], target: AttributeId): Float = ???
```

And a function to calculate the total information gain when selecting a certain attribute to classify our dataset:

```
def IG(data: RDD[Attributes],
       target: AttributeId,
       A: AttributedId): Float = ???
```

Next we can implement our decision tree learning algorithm.

```
def ID3(data: RDD[Attributes],
        target: AttributeId,
        attributes: Array[AttributeId]): DecisionTree = ???
```

Once you have implemented the decision tree learning algorithm you will have to report on the results. You can experiment with different combinations of features when running the algorithm and compare the results. In order to compare results split the data set in a training set and a test set.

Task 3: Answering Questions

Please include a section in your report in which you answer the following questions:

- Have you persisted some of your intermediate results? Can you think of why persisting your data in memory may be helpful for this algorithm?
- In which parts of your implementation have you used partitioning? Did it help for performance? If so, why?
- Did you use RDDs, DataFrames or Datasets for your implementation? Why?

Practical Info

- Student are required to implement the assignment individually. Collaboration between students is not allowed.
- The report should include an explanation of the design decisions made for implementing this assignment, a section about the obtained results from running the experiments in terms of performance and a section about the questions detailed in task 3.
- The deliverables for this project include a single scala file with your code and a pdf with the report (max 5 pages). These can be submitted as a single zip file through Canvas.
- The oral exam is an oral defence of your project in combination with an examination of the theory.
- For running your experiments on a local machine it is advised to construct a smaller sample dataset, you can find all of the datasets here.
- Once you have finalised your implementation you are required to run some experiments on Isabelle with the full dataset. Each student will get two 2h slots in order to run their experiments. Matteo Marra (matteo.Marra@vub.be) will contact you about reserving a slot.