Building a Decision Tree.

* Describe the general method for building a Decision Tree and the 2 key steps: Induction and Pruning.
* Describe the modified C4.5 algorithm.
* Explain one method of Pruning to avoid Overfitting.
* Building Decision Trees involves 2 steps:
* :
  + .
  + .
  + .
    - .

|  |  |  |  |
| --- | --- | --- | --- |
| Induction Phase | & | | Pruning Phase |
|  | | | |
| **Induction** 🡪 Means that you are building the initial Tree. | | **Pruning**🡪 Replace Subtrees with the **Majority Label**.  it goes through and does the replacements. | |
| So, you might initially have the tree on the left, and then when you do **Pruning** you get a *much smaller* tree (right one). | | | |
|  | | This tree should be *more* **Generalizable** and *less* **Overfit**. | |
| **The Induction Phase.**  It is where you are actually Creating the Initial Tree.  🡪 This is done Recursively top down. | | **The Pruning Phase.**  Where you are replacing Subtrees in your Decision Tree, that do **not** substantially impact performance.  Goal: Is to Generalize the tree.  🡪 So that you are ***not*** being so specific with all the branches.  So, you are favoring *smaller* Trees.  But !!  You want to do so, without compromising on **Accuracy** too much.  (Or whatever our performance measure is). | |
| **Induction**, requires us to have (some kind of) an **Evaluation Measure**. | | It helps **Overfitting**. | |
| **The C4.5 Algorithm.**  🡪 Is a very popular Algorithm for **Generating Decision Trees**.  There are many others.  Including improvements on **C4.5**.  Such as the one called **C5**.  The version we did, is a modification of **C4.5**.  It is a slight simplification which *only* deals with **Numeric Features**.  *(The full/real version handles* ***Nominal Features*** *too).*  **C4.5**  🡪 Uses **normalized Information Gain** for (as) its **Evaluation Measure**. | | There are several ways to **Prune**:  The one we did is an approach that goes from top to down:  We try to replace a Subtree with the **Majority Class**. (Or at least of the Classes bellow it).  And we **only** do it if it does **not** affect **Accuracy** too much. | |
| *How much is too much?*  This is one of the knobs we can turn with Machine Learning programs.  We could try making **epsilon**: 10%, 5%, 2%...  We could try many values and see if it helps. | |
| *Why Accuracy?*  We chose **Accuracy** because:  🡪 It is agnostic of the number of Classes that we have.  It just wants to know 🡪 How many of the observations we guessed the Label correctly.  It does **not** favor one class over the other.  *What else could we use?*  There are many others we could use.  But 🡪 That would be a very *specific case* of a Decision Tree.  It would **not** be a **Generalized Decision Tree Pruner**.  (& this is okay too). | |

Induction.

* Induction 🡪 Build an initial tree.
  + It is where you actually create the trees.
  + You take the Training data, it goes through 🡪 The initial Induction Algorithm and generate 🡪 A Tree.
* Induction is a Recursive Algorithm.
  + We are trying to go as far down as we can 🡪 Like a Depth-First Search.
  + And then we kind of bubble back up.

|  |  |
| --- | --- |
| Induction | |
|  | |
| (1) We pick a Feature for our Root Node 🡪 Sepal Length.   * Here we are doing it in order of left to right. * In reality, we will use a particular method to select the Feature that we are going to choose for our Root Node and each of our internal nodes. * We then order all of our observations by Sepal Length.   (2) We need to select a threshold, that is going to do a good job slitting my data 🡪 5.4   * This gives me a split of my data. ( 3 and 4).   (3) We consider the split on the top.   * These are all observations that are less than or equal to 5.4. | (1) Creates the Root Node with:   * Sepal Length as the Feature. * n = 22 🡪 Means there are 22 observations in the full dataset.   (3) This creates on the tree 2 branches:   * Left: For all observations that are ≤ to 5.4 * Right: For observations that are > than 5.4. * All these observations share the same Label.   (B) is a Leaf Node because:   * When looking at the observations that will go down the left branch (3), they all share the same Label. * n:7 🡪 The split (3) contains 7 observations. * As there is only 1 class represented in portion 3, I do not need to do more splitting.   This is a Terminal node, and I can say that: Anything that follows this path is going to be Setosa. |
|  | |
| (1) 🡪 This split of the data is done.  (2) We select the next Feature that we are going to look at 🡪 Sepal Width.  (3) Is the remainder of our data. We now sort only this portion, based on Sepal Width values.  (4) We pick a threshold 🡪 2.5. | (C) node represents:  The Feature we just selected 🡪 Sepal Width  n:15 🡪 Represents the 15 instances we have in portion (3). |

|  |  |
| --- | --- |
|  |  |
|  | |
| * We look at portion (1), which are the observations that would go down [c-1]. * Observations in (1) only have 1 Label (3) 🡪 Versicolor.   + Thus, we wind up with a Leaf Node = (D) | (D) is the Leaf Node representing portion (1), which contains its 3 observations. |

|  |
| --- |
| Feature Selected to Examine 🡪 PETAL LENGTH |
|  |
| * We look at the next split that we had: (1). * As we have a mixture of Classes in split (1): 🡪 We need to select another Feature to examine (2). * (2) – We select **Petal Length** to examine now.   + And, we order (1) based on said Feature. * (3) – We pick a threshold 🡪 4.8 * (4) – Is the set that is ≤ to 4.8. *(This is the set we will look at next – always the left branch)*. * (5) – Is the set that is > than the selected threshold. |

|  |  |
| --- | --- |
| Feature Selected to Examine 🡪 Petal Width | |
|  | |
| * Now we examine split (1):   + Again, we’ve got more than 1 Class 🡪 So we need to come up with a split. * (2) – We select Petal Width to examine. * (3) – We are going to use the threshold of 🡪 0.4. | |
| And, if I go left down [f-1] 🡪 I only have Setosa (2). == And, if I look at set (1) 🡪 I only have Setosa (2). | |
| * As in split (1) we only have Setosa 🡪 That means we create a Leaf Node.   + Split (1) only contains 2 observations,   + Thus, 🡪 n:2 . | |
| And if we go right down [f-2] 🡪 We only have Versicolor. | |
|  | |
|  |  |
|  |  |
|  |  |

C4.5 Algorithm

* C4.5 Algorithm 🡪 For building a Decision Tree.
  + This is a variation of the C4.5 Algorithm.
  + It is a very classic Algorithm that was introduced in the 90s, it is still widely used.
  + There are a couple of newer versions, or improved versions, of it.
  + This is kind of the workhorse of the Decision Tree world.

|  |  |  |  |
| --- | --- | --- | --- |
| C4.5 Algorithm – For building a Decision Tree | | | |
| We are only going to consider Numeric Features  It starts of as a Training Set. And as we go through the Algorithm we are going to be looking at: Smaller and smaller slices of our data. | | | |
| BuildTree(observations) | 1 | observations 🡪 the Set of observations we are currently looking at. | This function is recursive.  So, we will end up calling **BuildTree()** again later toward the end.  It starts of with a Training Set as input.  And as we go through the Algorithm we are going to be looking at: Smaller and smaller slices of our data. |
| if observations consists of a single class label C: | 2 |  |  |
| return LeafNode( { C: size(observations) } ) | 3 | C 🡪 the Class Label. | We will create a new LeafNode instance.  Here, Hank is using json dictionary notation. |
| size(observations) 🡪 the number of observations we have |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Some helper Classes:  These are just to keep track of some information. | | |
| class InternalNode |  | As we are only going to consider Numeric Features 🡪 All of our Internal Nodes will have 2 children:   * S |
| class LeafNode | labelDistribution | We are actually going to keep track of the distribution of Labels. |
| class FeatureInfo |  |  |

Prune Algorithm.

* Collapsing 🡪 Substituting a Subtree with its Majority Label.

|  |  |  |
| --- | --- | --- |
| Pruning a SMALL Tree. | | |
| This is the tree we generated with the initial **Induction Algorithm**. | The problem with this tree, is that it is already fairly simple.  Thus, in this small Tree, it does not actually do us much good to do Collapsing.  Another problem, is that we only have 22 observations.  Any changes to 22 observations in term of the Classification, is going to drastically alter the Accuracy, in terms of percentage. | |
| Looking at the whole tree,  If we use a Majority Class of all of our Leaf Nodes.  In this case, we have a tie:   * Versicolor = 9. * Setosa = 9. * Virginica = 4.   We have to pick one Class. If we do it alphabetically 🡪 Setosa would end up winning as the Majority Class. | | |
| If we use the original Training data, from where all Leaf Nodes are built from, then:  Our Accuracy is 100% .  So, if we end up Classifying everything as Setosa:  (thus, eliminating the other Classes)  We are going to get 9 of them correct, and there are 22 total:  9/22 ≈ 0.4090…  Our Accuracy dips below 50%.  This is too much. So, we are **not** going to do that.  We are **not** going to use this whole tree. | | Important!  This is on our Training Set.  But, when doing Pruning, we could also use:   * Some sample of our Training Set, or, * A little help out piece of data as well. |
| So, now we look at our 2 Subtrees:  On the left [a-1], there is not much of a Subtree, so I do not need to worry about that.  On the right, [a-2], I could do the same calculations. | | |
| For the [a-2] Subtree, I look at the majority Label. Which is 🡪 Versicolor.  72.7% is a little bit better.  But it is still too big of a gap, from 100%, so we might not want to do this either. | | |
| So we basically keep going down.  What we are doing is, basically, recomputing the Accuracy if we were to substitute our Subtree with the Majority Label.  And, if that Accuracy differs too much from our original Accuracy 🡪 We are not going to substitute that Subtree.  aka. We are not going to do Collapsing. | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| Prune Algorithm | | |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
|  | |
|  | |
|  |  |
|  |  |