CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Machine Learning

Machine Learning

Lecture: K-D Trees

Ted Scully

When to use k-Nearest Neighbour

- Drawbacks:
  - Can only accommodate a **moderate number of features** (typically 20 or less)
  - Problems with **irrelevant features**. By default all features contribute equally to the similarity metric.
  - Can be very sensitive to **imbalanced datasets**.
  - It is called instance-based because it constructs hypotheses directly from the training instances themselves. This means that the hypothesis complexity can grow with the size of the data. Therefore it can be **slow in classification/regression if there is a large dataset.**

Curse of Dimensionality

- After over-fitting, the biggest problem in machine learning is the "curse of dimensionality".

- The curse of dimensionality refers to various issues that arise when analyzing data in high-dimensional spaces (potentially with hundreds or even thousands of dimensions) and is particularly relevant to instance based learners.

- The common root of these difficulties lies in the fact that **as dimensionality increases, the volume of space increases so fast that the available data becomes sparse.**
  - This raises particular problems for ML techniques that rely on detecting areas where objects form groups with similar properties.

Curse of Dimensionality

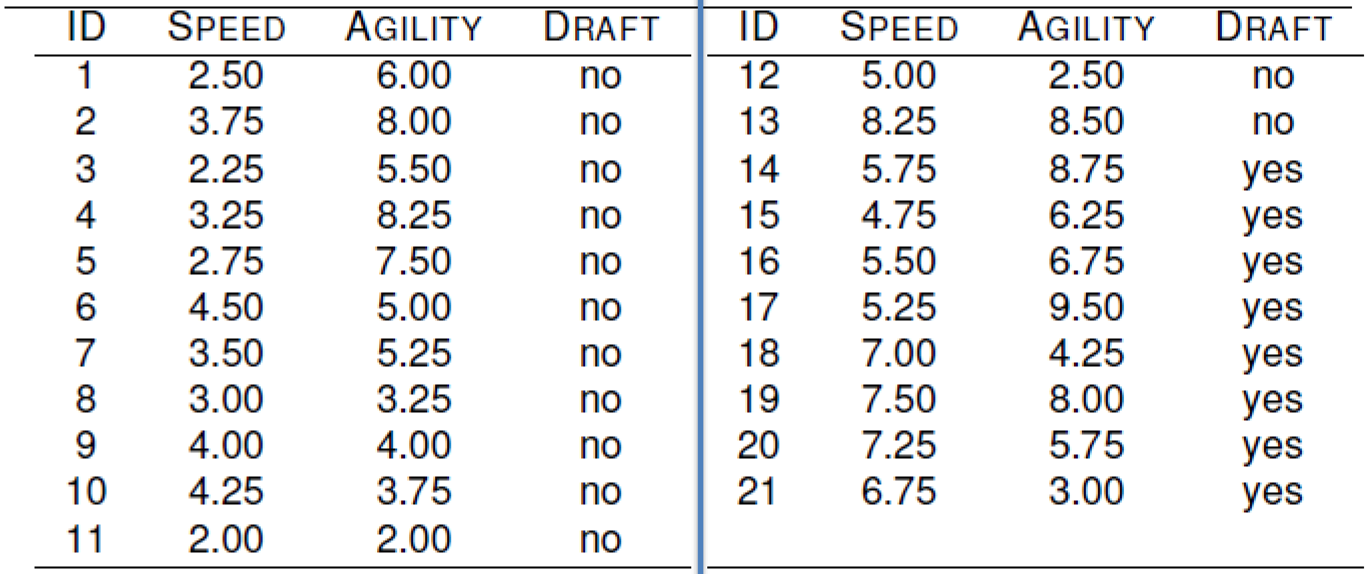
- Many algorithms that work fine in low dimensions become intractable when the input is high-dimensional.
- Generalizing correctly becomes exponentially harder as the dimensionality (number of features) of the examples grows, because **a fixed-sized training set covers a dwindling fraction of the input space**.

When to use k-Nearest Neighbour

- Drawbacks:
  - Can only accommodate a **moderate number of features** (typically 20 or less)
  - Problems with **irrelevant features**. By default all features contribute equally to the similarity metric.
  - Can be very sensitive to **imbalanced datasets**.
  - It is called instance-based because it constructs hypotheses directly from the training instances themselves. This means that the hypothesis complexity can grow with the size of the data. Therefore it can be **slow in classification/regression if there is a large dataset.**

Data Structures for Improving Runtime Performance

- The fact that kNN algorithms store the entire training set in memory has a very negative impact on the time complexity of these algorithms.
- Structures and methods have been developed that can be used to help alleviate this issue (assuming the dataset is relatively stable).
- One such structure is called a **k-d tree**, which is a balanced binary tree.
- We will use the following example to help illustrate the ideas of a k-d tree.

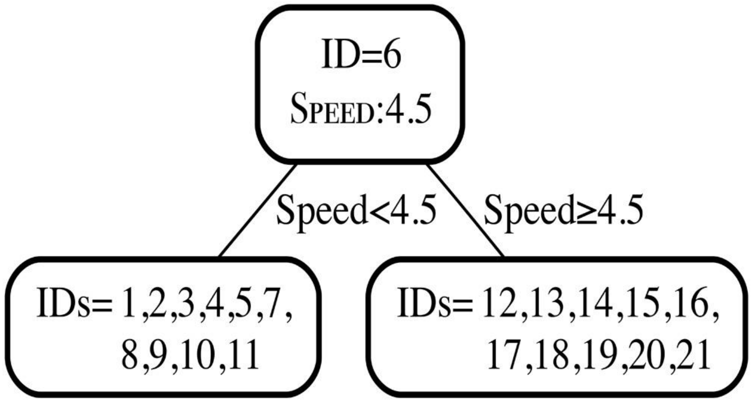


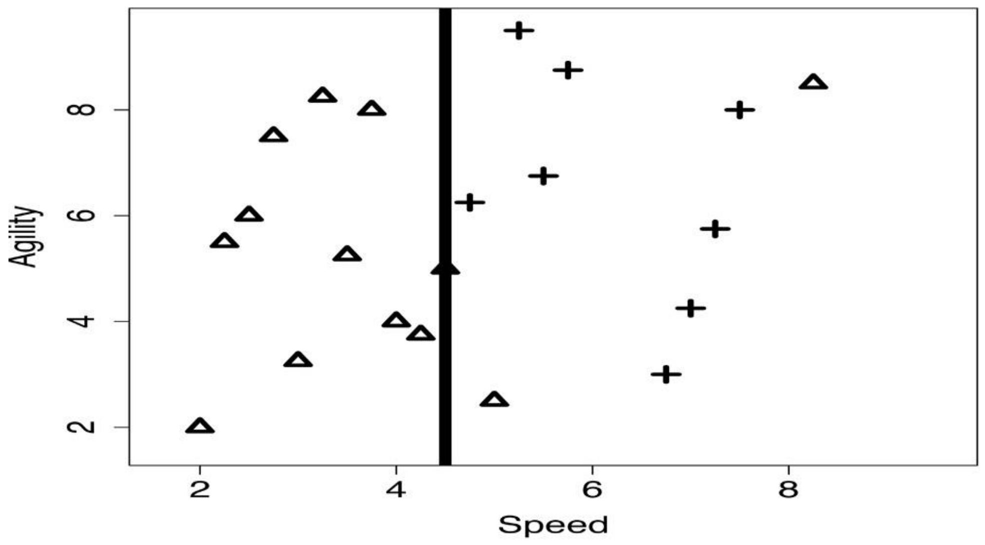
ID	SPEED	AGILITY	DRAFT
1	2.50	6.00	no
2	3.75	8.00	no
3	2.25	5.50	no
4	3.25	8.25	no
5	2.75	7.50	no
6	4.50	5.00	no
7	3.50	5.25	no
8	3.00	3.25	no
9	4.00	4.00	no
10	4.25	3.75	no
11	2.00	2.00	no
12	5.00	2.50	no
13	8.25	8.50	no
14	5.75	8.75	yes
15	4.75	6.25	yes
16	5.50	6.75	yes
17	5.25	9.50	yes
18	7.00	4.25	yes
19	7.50	8.00	yes
20	7.25	5.75	yes
21	6.75	3.00	yes

Machine Learning for Predictive Analytics – Kelleher et al.

Data Structures for Improving Runtime Performance

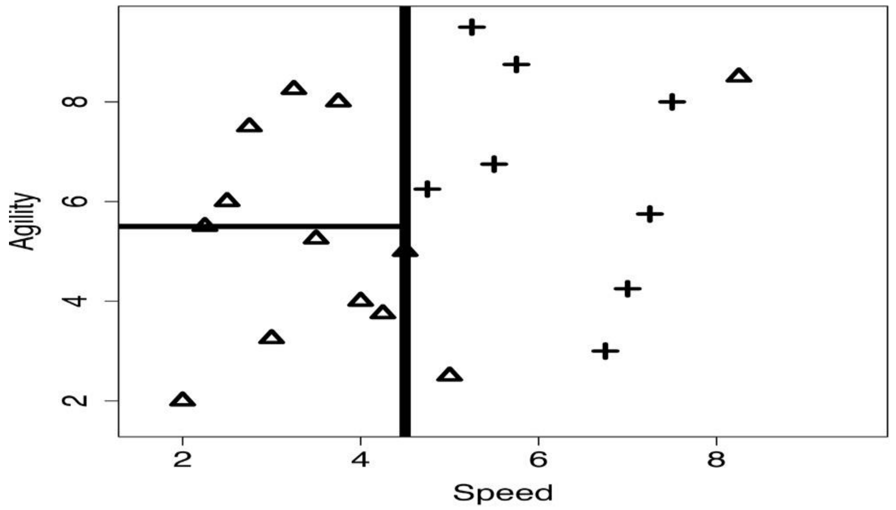
- To construct a k-d tree we arbitrarily put all **features into a list**.
- We pick the first feature and split the data using the median value of that feature.
- The example below shows a simple dataset that contains a speed and agility feature.
- We initially split by speed using the median value from the data set.

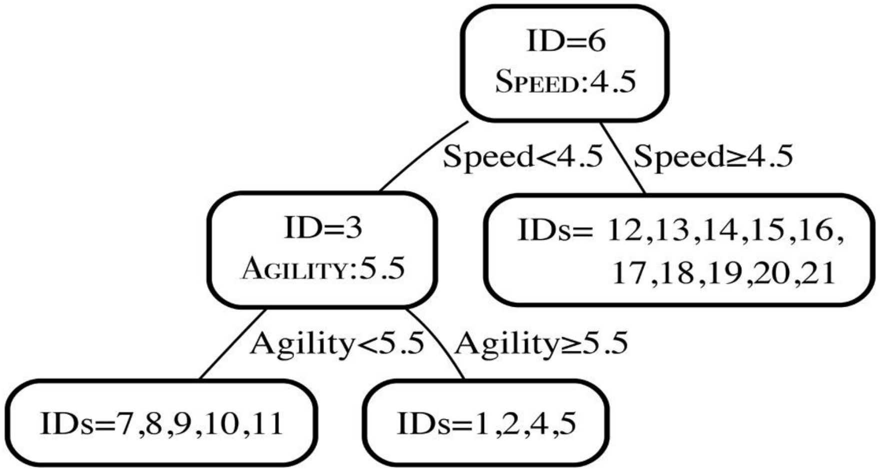

ID=6
SPEED:4.5
Speed<4.5
Speed≥4.5
IDs= 1,2,3,4,5,7, 8,9,10,11
IDs= 12,13,14,15,16, 17,18,19,20,21

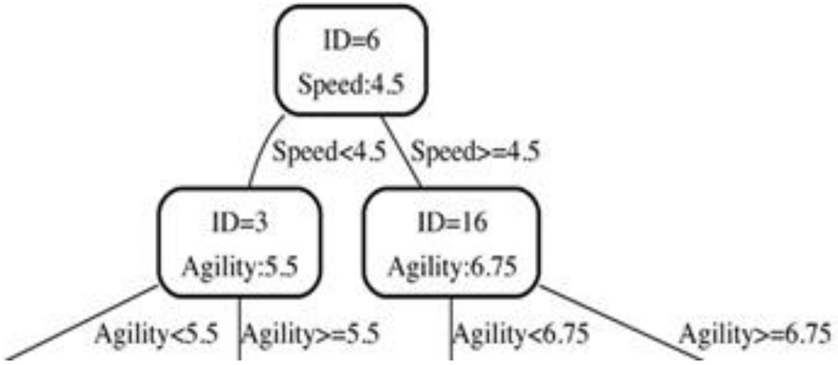


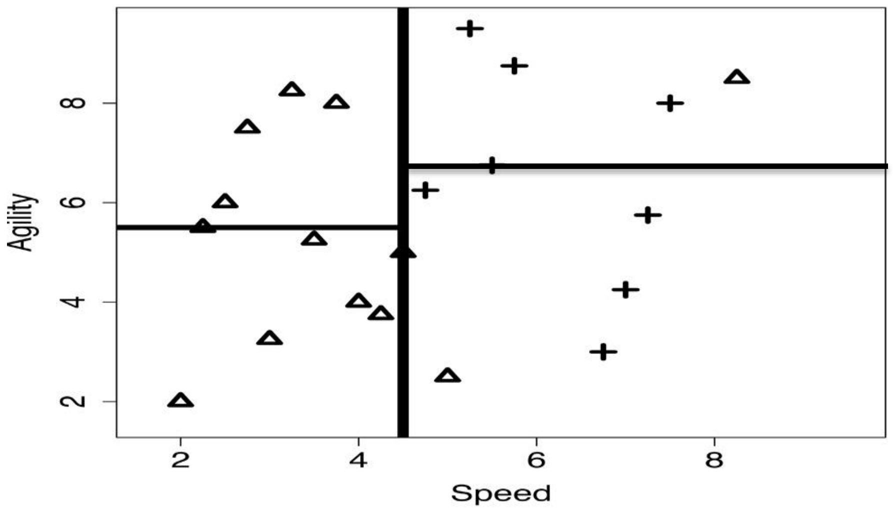
Data Structures for Improving Runtime Performance

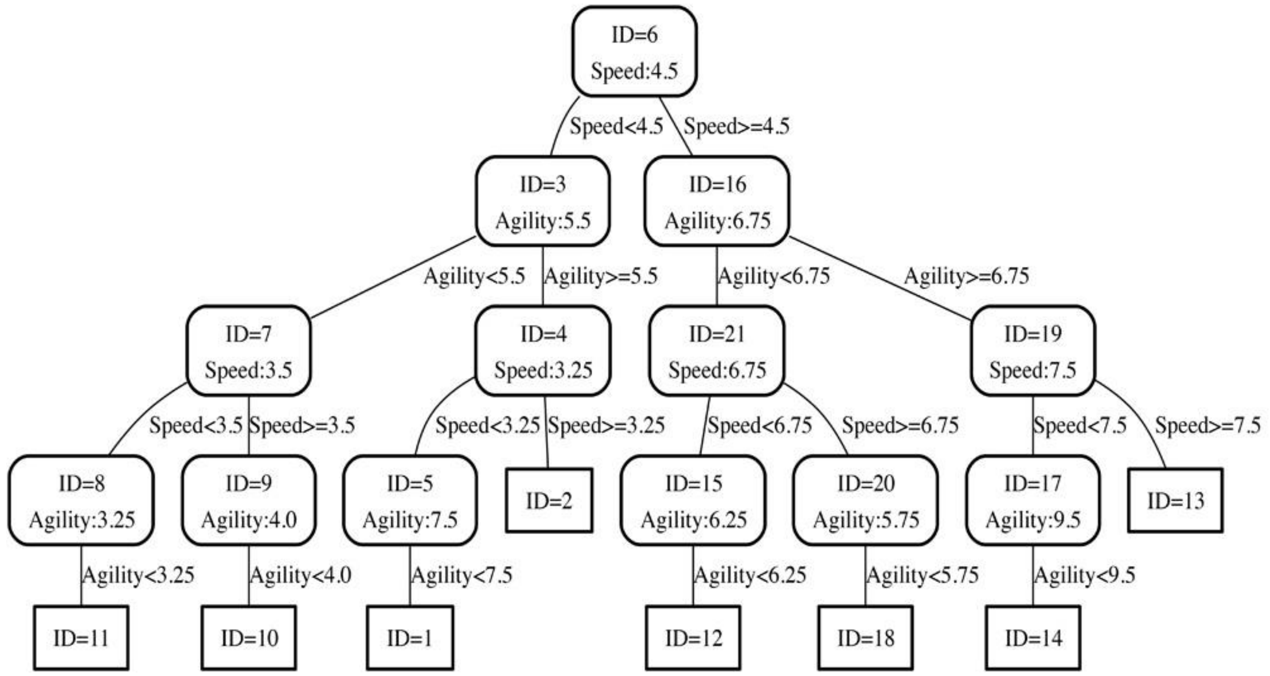
- We select the **next feature** in the list and split according to the median of that feature.
- Every time we partition the data, we add **a node with two branches to the tree**. The node represents the single instance that is the median value of the feature.
- The left branch holds all instances that have a value less then the median of the feature and on the right branch all instances greater than or equal to the median.
- We continue this process until we end up with single data instances along each branch. The single data instances then becomes the leaf nodes of the tree.

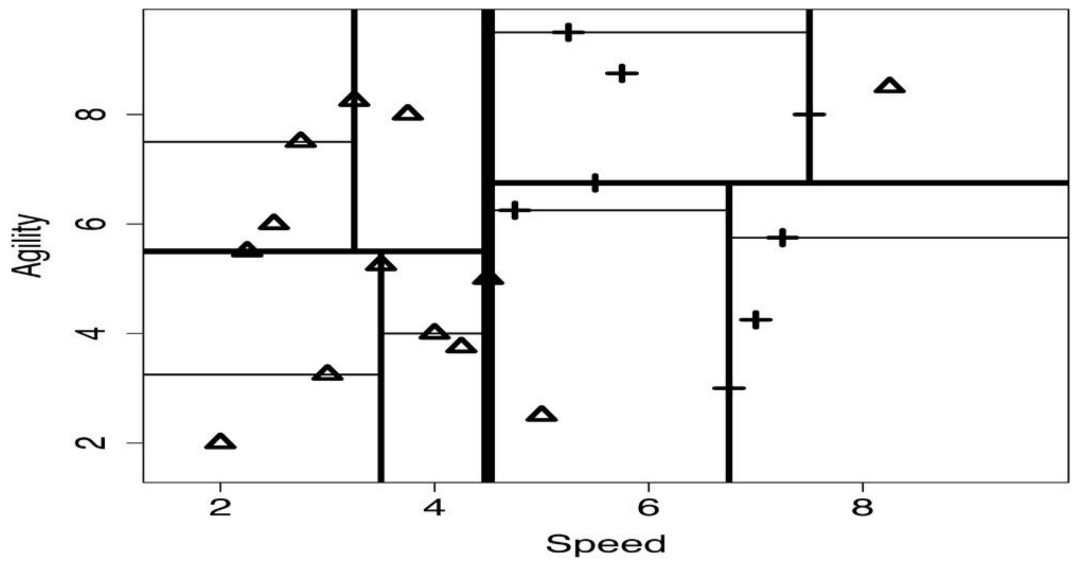




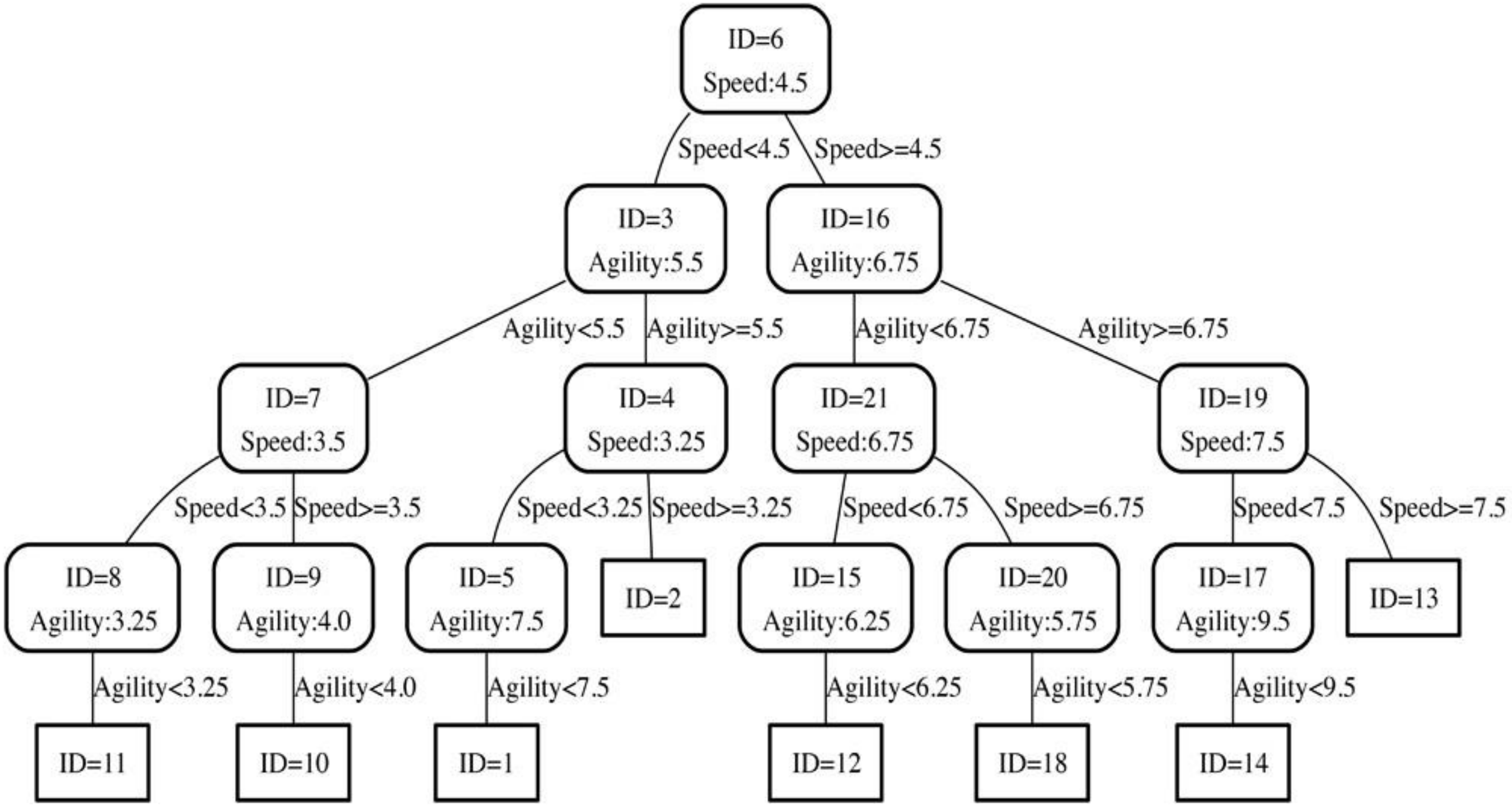

ID=6
Speed:4.5
Speed<4.5
Speed>=4.5
ID=3
Agility:5.5
ID=16
Agility:6.75
Agility<5.5
Agility>=5.5
Agility<6.75
Agility>=6.75


Agility
Speed
2
4
6
8

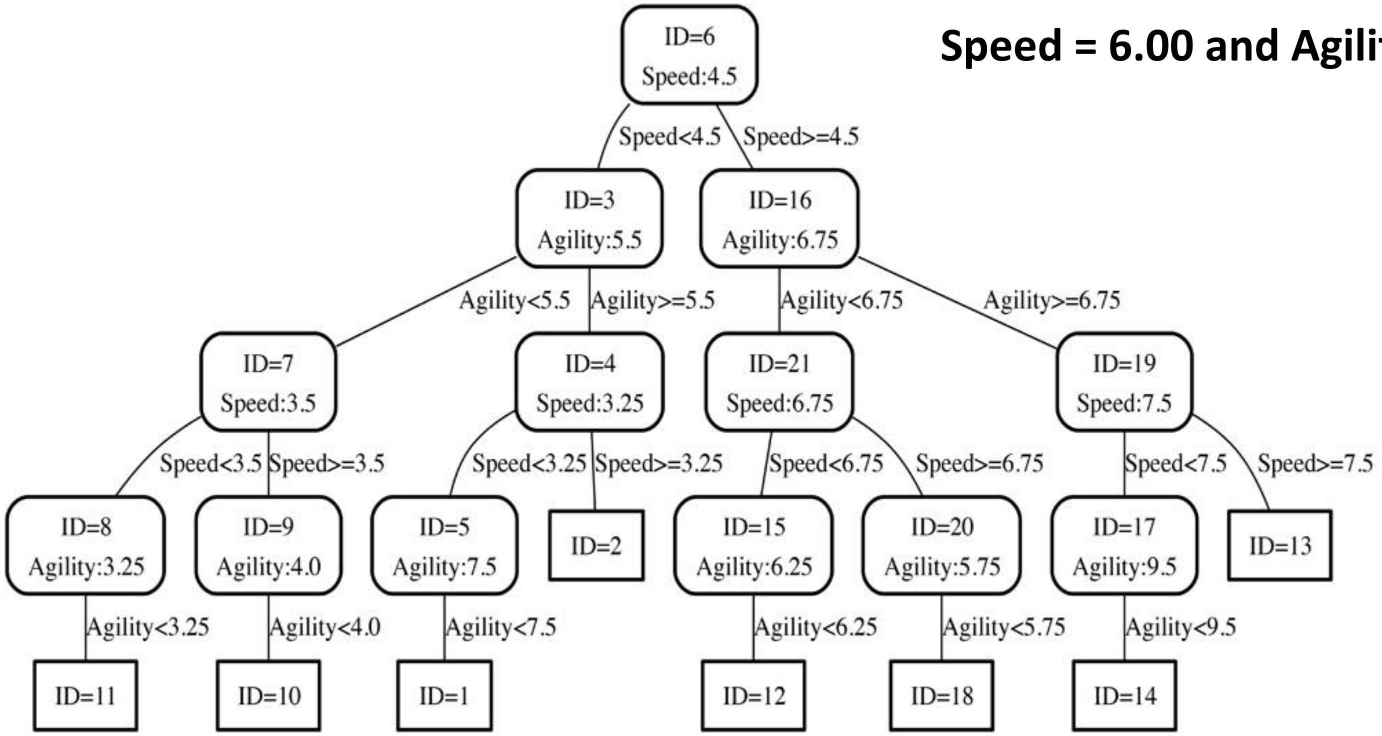

ID=6
Speed:4.5
Speed<4.5
Speed>=4.5
ID=3
Agility:5.5
ID=16
Agility:6.75
Agility<5.5
Agility>=5.5
Agility<6.75
Agility>=6.75
ID=7
Speed:3.5
ID=4
Speed:3.25
ID=21
Speed:6.75
ID=19
Speed:7.5
Speed<3.5
Speed>=3.5
Speed<3.25
Speed>=3.25
Speed<6.75
Speed>=6.75
Speed<7.5
Speed>=7.5
ID=8
Agility:3.25
ID=9
Agility:4.0
ID=5
Agility:7.5
ID=2
ID=15
Agility:6.25
ID=20
Agility:5.75
ID=17
Agility:9.5
ID=13
Agility<3.25
Agility<4.0
Agility<7.5
Agility<6.25
Agility<5.75
Agility<9.5
ID=11
ID=10
ID=1
ID=12
ID=18
ID=14

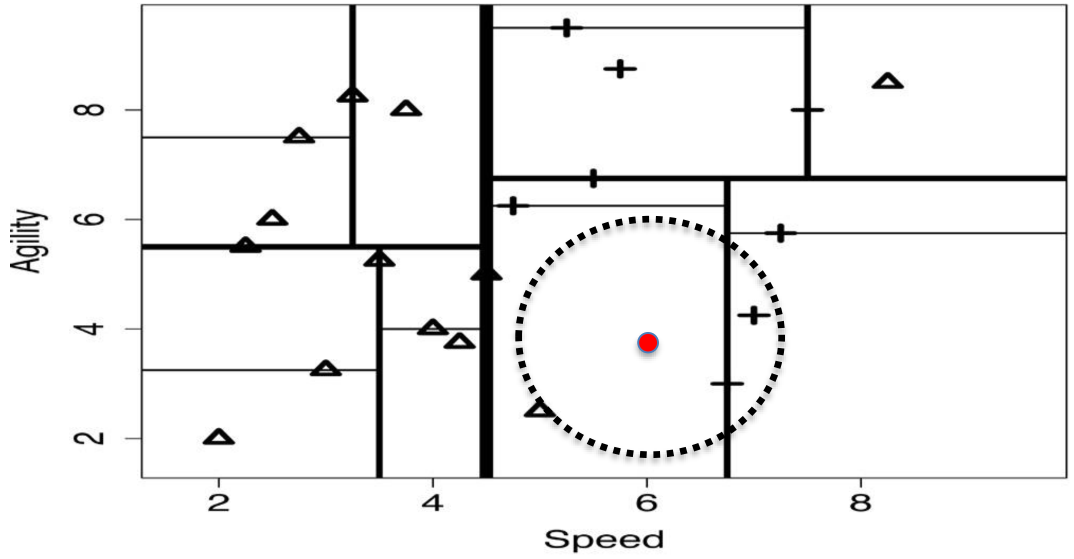

Agility
2
4
6
8
Speed

- Once we have stored the instances in a dataset in the tree we can use the tree to retrieve the nearest neighbour for a query instance.
- The first step of this process is to take the query instance and follow it's path through the tree (take a query instance where **Speed = 6.00 and Agility = 3.50**)



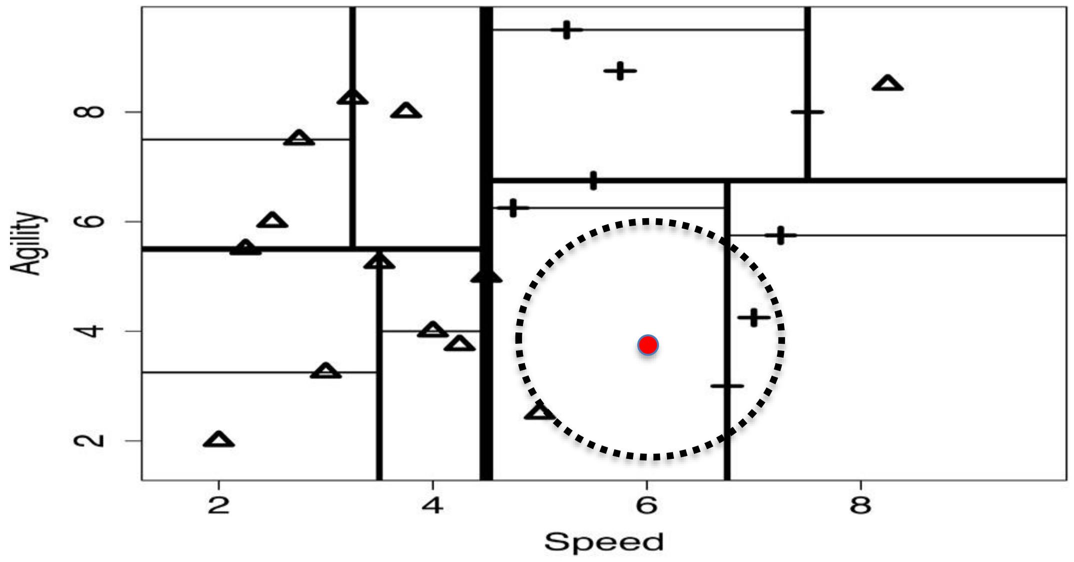
Speed = 6.00 and Agility = 3.50

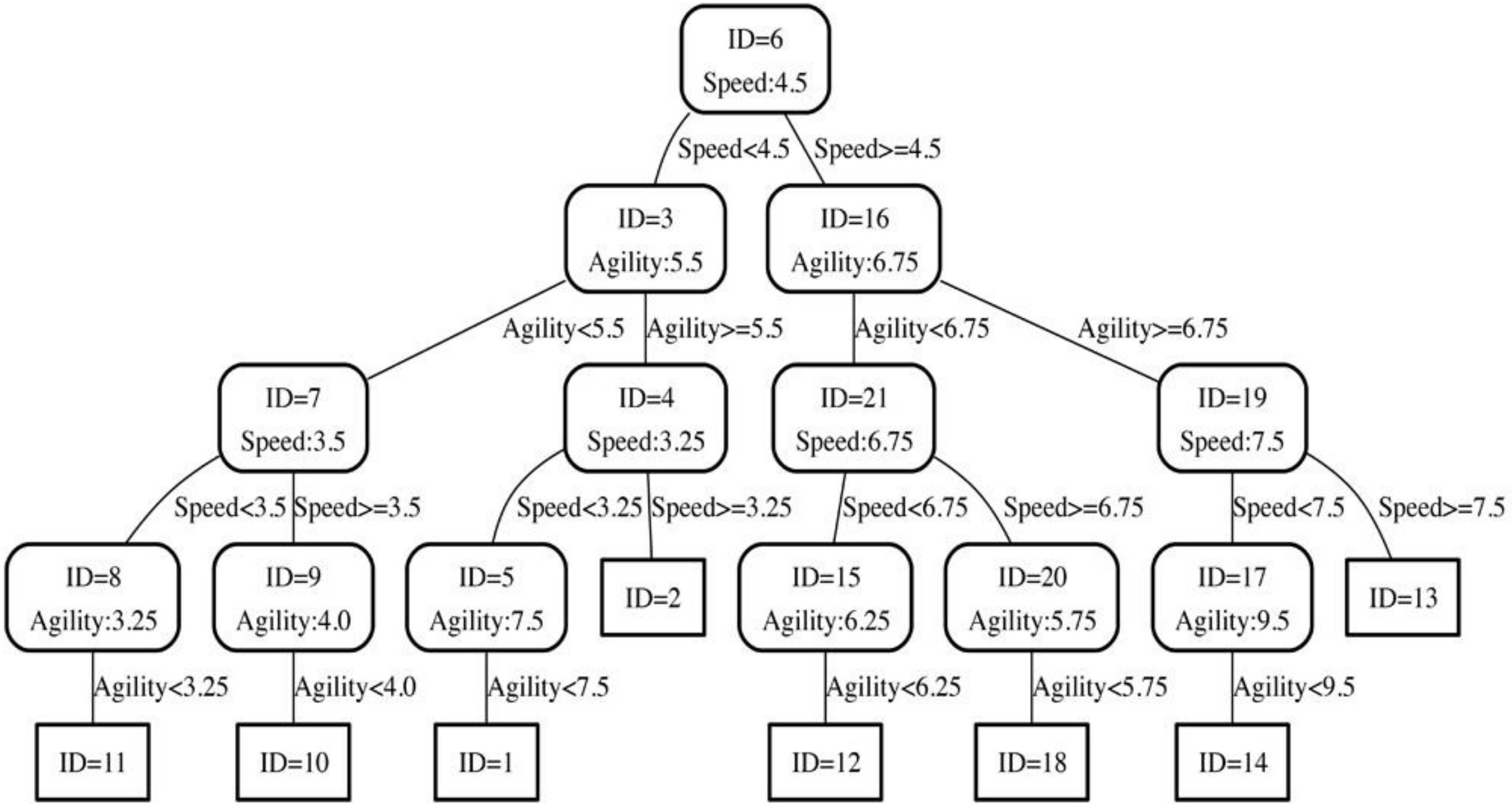




Speed = 6.00 and Agility = 3.50

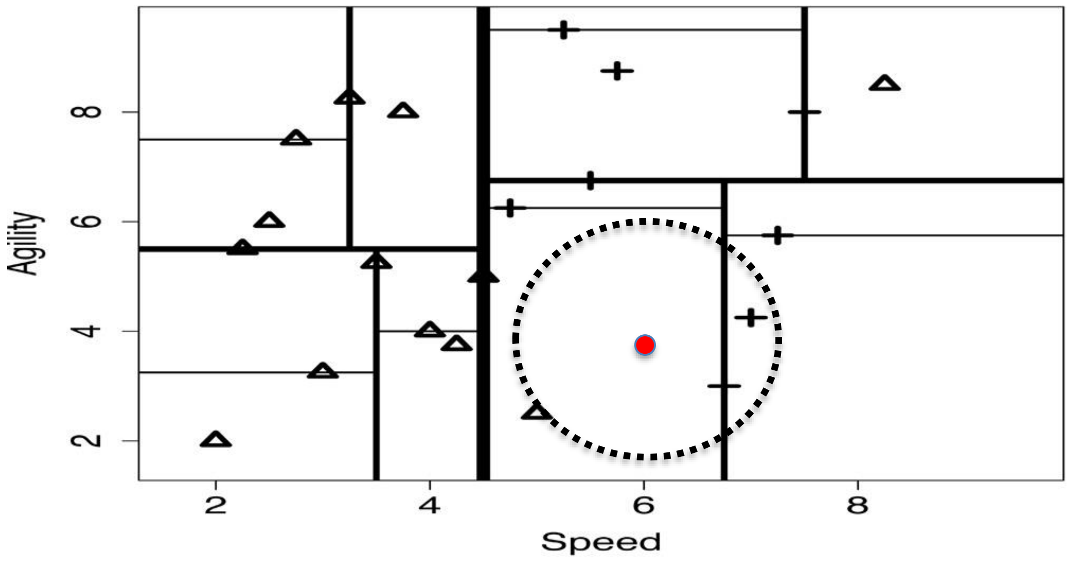
- The red point specifies the query point in space and the dashed circle shows the Euclidean distance from the point to the training instance with Id=12.
- At this point we know that the true nearest neighbour is located on the edge of the dashed circle or within it.
- The **leaf node is not guaranteed to be the nearest neighbour**. For example we can see from the figure that the node with id 12 is not the true nearest neighbour to the query.
- So the search process begins to **iteratively move back up the tree** stopping at each parent node to determine **if the node and the associated boundary is closer than the current closest node**.

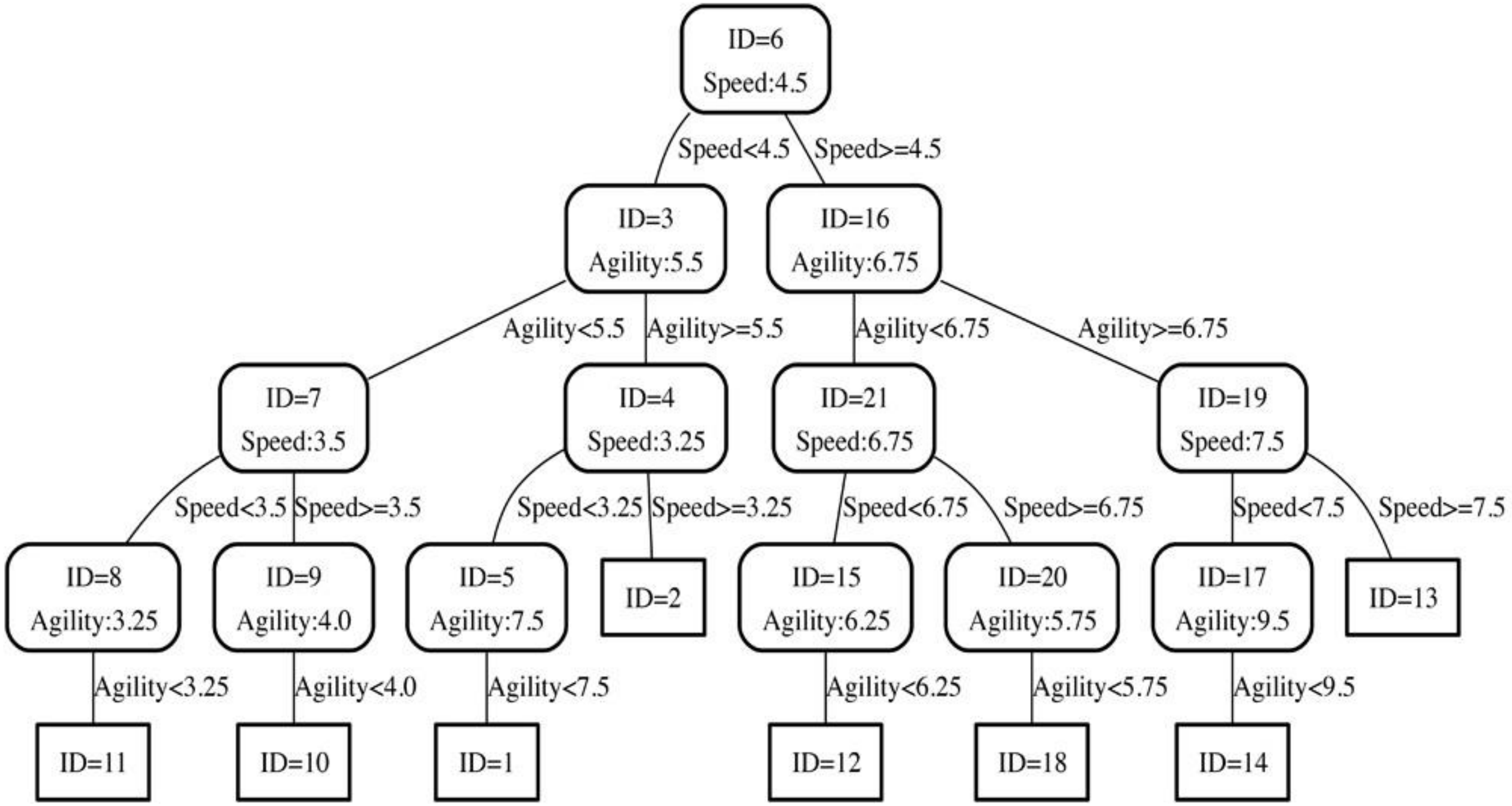



ID=6
Speed:4.5
Speed<4.5
Speed>=4.5
ID=3
Agility:5.5
ID=16
Agility:6.75
Agility<5.5
Agility>=5.5
Agility<6.75
Agility>=6.75
ID=7
Speed:3.5
ID=4
Speed:3.25
ID=21
Speed:6.75
ID=19
Speed:7.5
Speed<3.5
Speed>=3.5
Speed<3.25
Speed>=3.25
Speed<6.75
Speed>=6.75
Speed<7.5
Speed>=7.5
ID=8
Agility:3.25
ID=9
Agility:4.0
ID=5
Agility:7.5
ID=2
ID=15
Agility:6.25
ID=20
Agility:5.75
ID=17
Agility:9.5
ID=13
Agility<3.25
Agility<4.0
Agility<7.5
Agility<6.25
Agility<5.75
Agility<9.5
ID=11
ID=10
ID=1
ID=12
ID=18
ID=14

Speed = 6.00 and Agility = 3.50

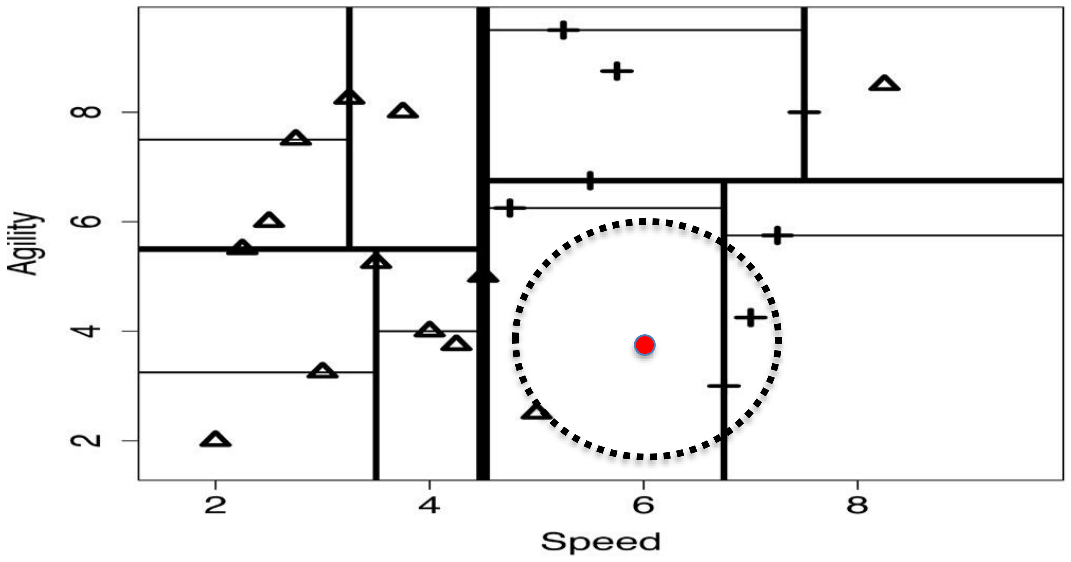
- The next node is **ID 15**, which is not any closer.
- Therefore, next we check to see if the distance between the query and the boundary represented by ID15 is less than best distance. It isn't. If it was then we would update our nearest neighbour.
- Next we check to see if the boundary represented by ID15 is less than the current best distance. It isn't there we move up the tree again.
- (If the distance to boundary had been less than best distance then it could be that there were instance on the other side of the boundary, down the right hand branch that could be closer)




ID=6
Speed:4.5
Speed<4.5
Speed>=4.5
ID=3
Agility:5.5
ID=16
Agility:6.75
Agility<5.5
Agility>=5.5
Agility<6.75
Agility>=6.75
ID=7
Speed:3.5
ID=4
Speed:3.25
ID=21
Speed:6.75
ID=19
Speed:7.5
Speed<3.5
Speed>=3.5
Speed<3.25
Speed>=3.25
Speed<6.75
Speed>=6.75
Speed<7.5
Speed>=7.5
ID=8
Agility:3.25
ID=9
Agility:4.0
ID=5
Agility:7.5
ID=2
ID=15
Agility:6.25
ID=20
Agility:5.75
ID=17
Agility:9.5
ID=13
Agility<3.25
Agility<4.0
Agility<7.5
Agility<6.25
Agility<5.75
Agility<9.5
ID=11
ID=10
ID=1
ID=12
ID=18
ID=14

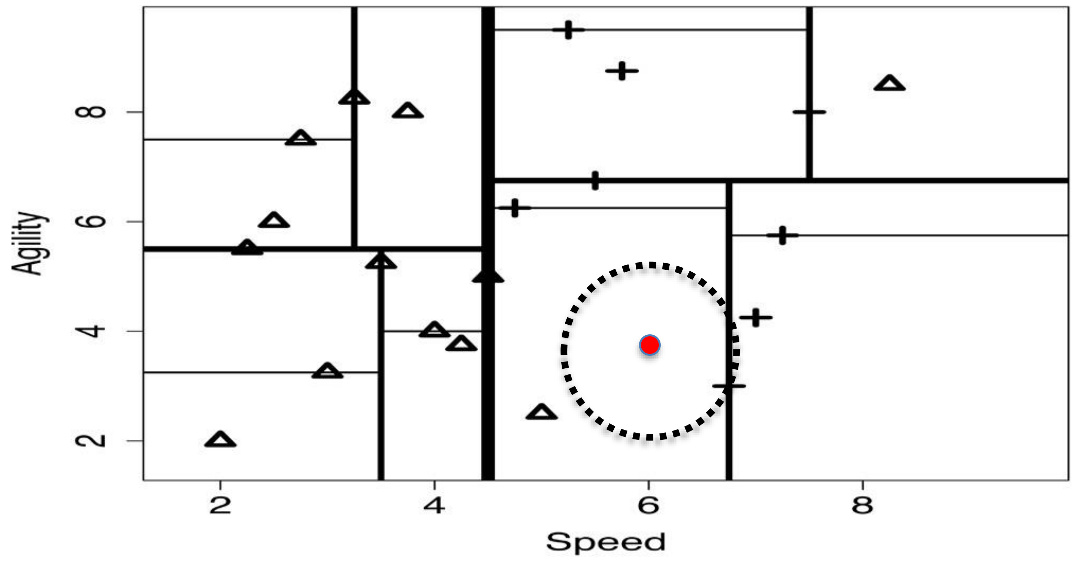
Speed = 6.00 and Agility = 3.50

- The node with **ID 21**, is closer to the query and this now becomes the new best distance and the current nearest neighbour.



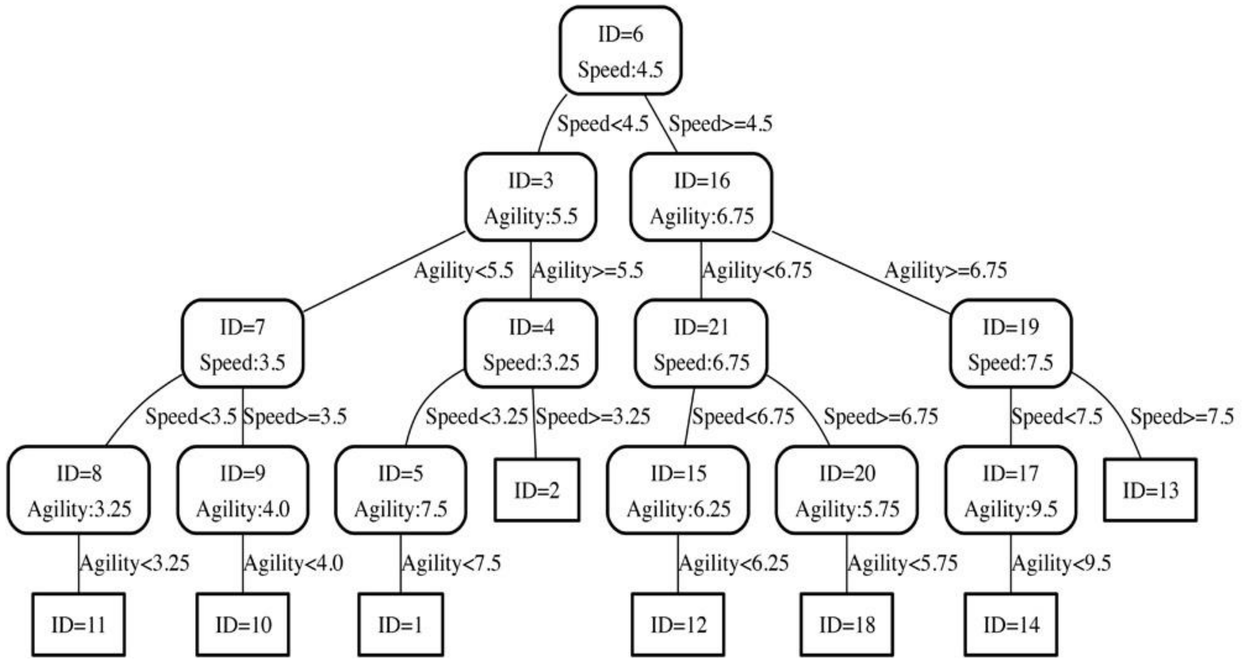
Speed = 6.00 and Agility = 3.50

- The node with **ID 21**, is closer to the query and this now becomes the best distance and the current nearest neighbour.
- Again we check to see if the boundary associated with this node (ID 21) is closer than best distance. It is, therefore, we must move down the right hand side of this node to check if there are any closer nodes. However, there are not any closer nodes.
- We continue to move back up the tree until we reach the root node and the node ID21 will be returned as the closest node.



Data Structures for Improving Runtime Performance

- Data structures such as KD trees can save us from calculating quite a lot of distance metrics.
- In the example below we only had to calculate the distance between the query and six instances in the training set (7 out of 21 instances).
- KD-trees are reasonably efficient when there are a lot **more instances than features**. Typically you should be looking for 2^m instances for *m* features at least. The performance of a KD-tree can suffer significantly is this is not the case.


ID=6
Speed:4.5
Speed<4.5
Speed>=4.5
ID=3
Agility:5.5
ID=16
Agility:6.75
Agility<5.5
Agility>=5.5
Agility<6.75
Agility>=6.75
ID=7
Speed:3.5
ID=4
Speed:3.25
ID=21
Speed:6.75
ID=19
Speed:7.5
Speed<3.5
Speed>=3.5
Speed<3.25
Speed>=3.25
Speed<6.75
Speed>=6.75
Speed<7.5
Speed>=7.5
ID=8
Agility:3.25
ID=9
Agility:4.0
ID=5
Agility:7.5
ID=2
ID=15
Agility:6.25
ID=20
Agility:5.75
ID=17
Agility:9.5
ID=13
Agility<3.25
Agility<4.0
Agility<7.5
Agility<6.25
Agility<5.75
Agility<9.5
ID=11
ID=10
ID=1
ID=12
ID=18
ID=14