

# GOML2: ML in Go

Because...why not

# Today

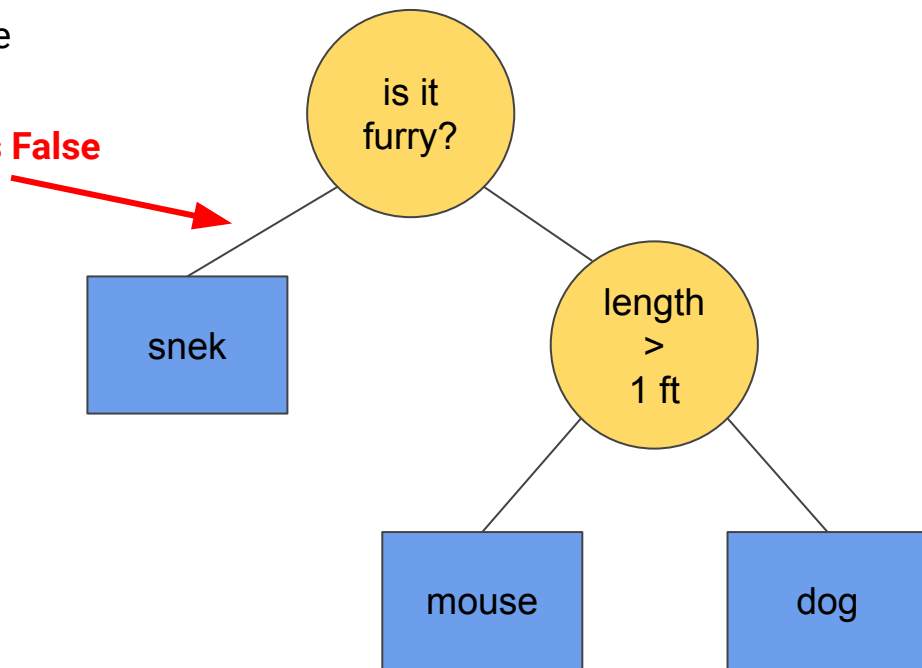
- **Classification trees**
- **Bias vs variance**
- **Cost-Complexity pruning**
- **Project timeline for GOML2**



# Classification

Our decision tree

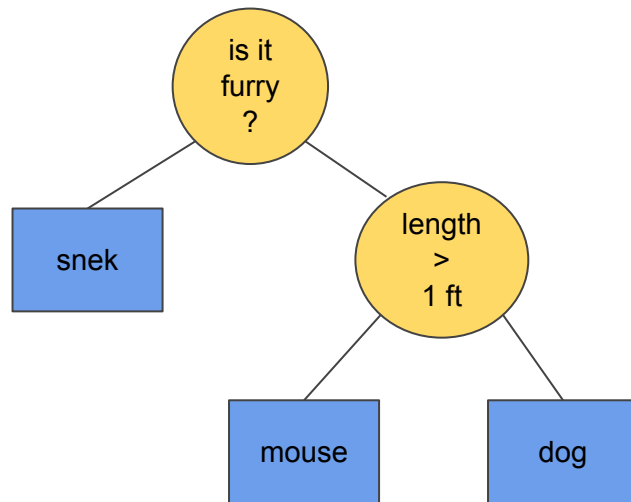
**assume left is False**



# Classification

Some data - let's suppose our tree was trained on this data

Furry?	Length	Class
X	3.0 ft	snek
✓	0.4 ft	mouse
✓	2.1 ft	dog
X	2.4 ft	snek
✓	3.0 ft	dog
✓	0.5 ft	dog
✓	0.6 ft	mouse



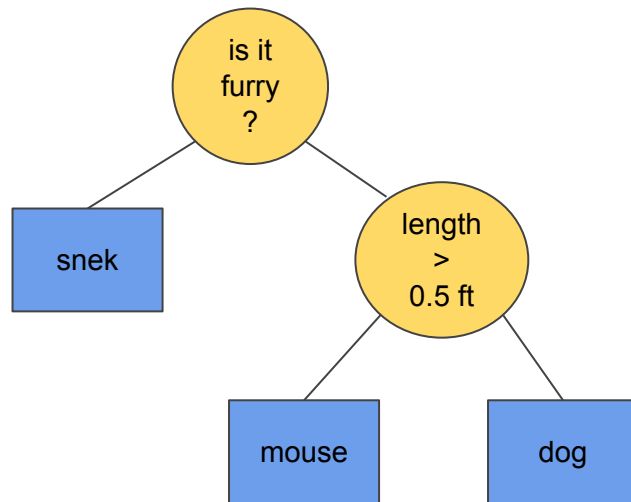
# Classification

There are data points that are classified incorrectly

Furry?	Length	Class
X	3.0 ft	snek
✓	0.4 ft	mouse
✓	2.1 ft	dog
X	2.4 ft	snek
✓	3.0 ft	dog
✓	0.5 ft	dog
✓	0.6 ft	mouse

!!!

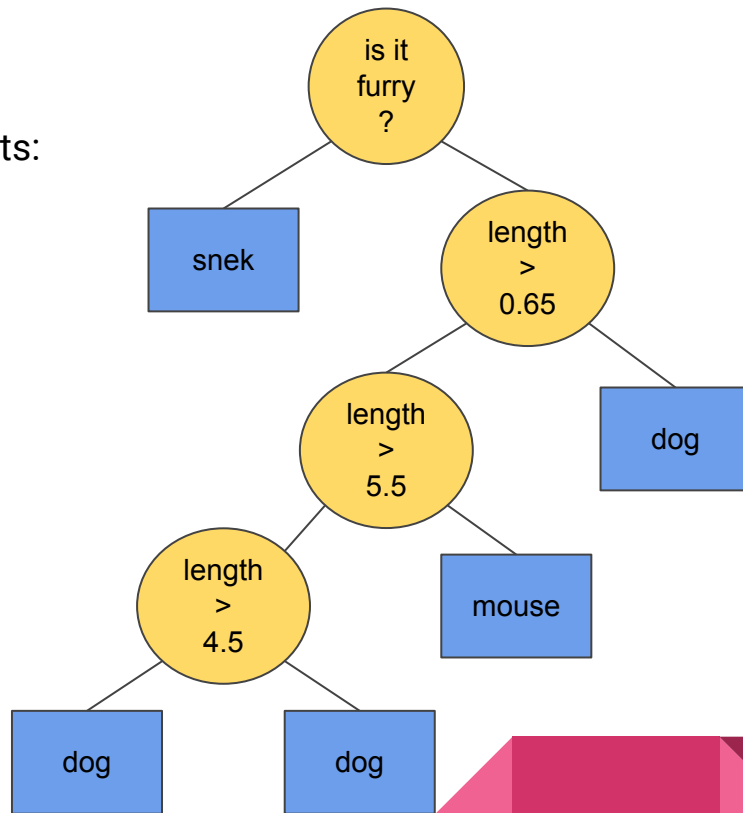
!!!



# Classification

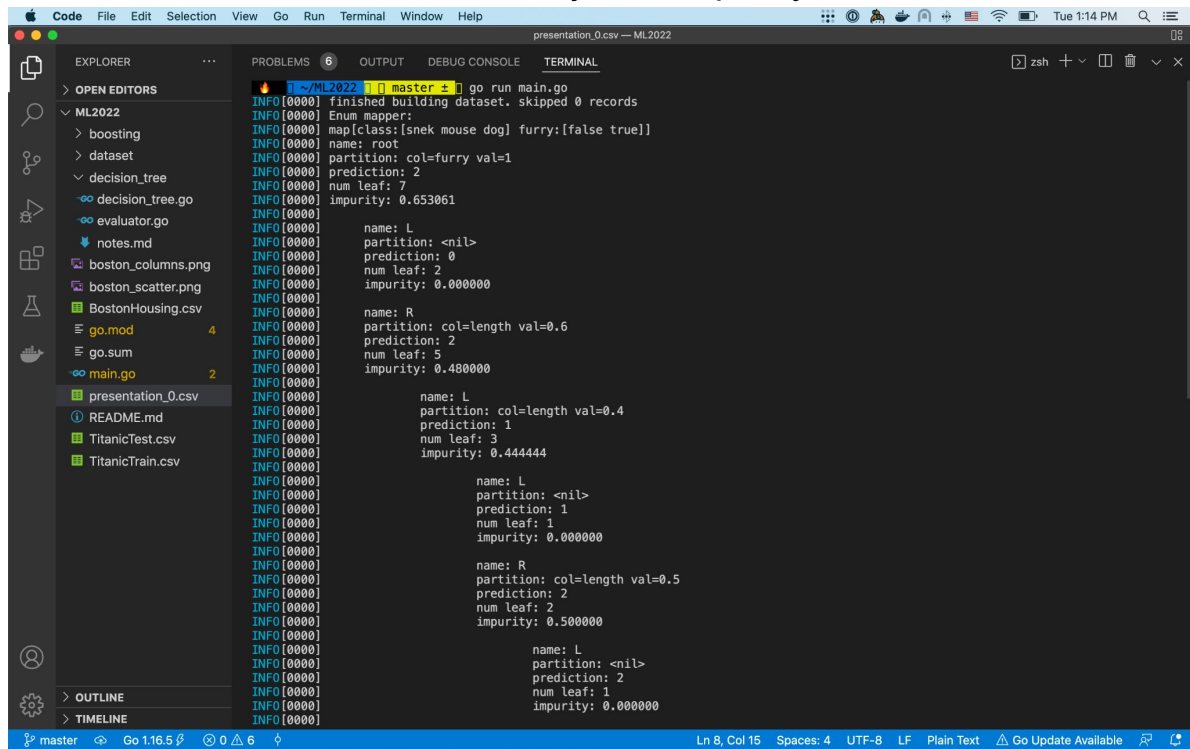
We could build a tree like this to get all of the data points:

Furry?	Length	Class
X	3.0 ft	snek
✓	0.4 ft	mouse
✓	2.1 ft	dog
X	2.4 ft	snek
✓	3.0 ft	dog
✓	0.5 ft	dog
✓	0.6 ft	mouse



# Classification

How to do this with GOML2... Sorry, not as pretty 🙄



The screenshot shows a VS Code editor window with a Go project. The Explorer panel on the left shows the file structure, including a directory named 'ML2022' and files like 'presentation\_0.csv'. The Terminal panel on the right shows the output of running 'go run main.go'. The output displays the results of a classification task using GOML2, showing the number of records, the number of leaves, and the impurity for each partition. The output is as follows:

```
INFO[0000] finished building dataset. skipped 0 records
INFO[0000] Enum mapper:
INFO[0000] map[class:[snek mouse dog] furry:[false true]]
INFO[0000] name: root
INFO[0000] partition: col=furry val=1
INFO[0000] prediction: 2
INFO[0000] num leaf: 7
INFO[0000] impurity: 0.653061
INFO[0000]
INFO[0000] name: L
INFO[0000] partition: <nil>
INFO[0000] prediction: 0
INFO[0000] num leaf: 2
INFO[0000] impurity: 0.000000
INFO[0000]
INFO[0000] name: R
INFO[0000] partition: col=length val=0.6
INFO[0000] prediction: 2
INFO[0000] num leaf: 5
INFO[0000] impurity: 0.480000
INFO[0000]
INFO[0000] name: L
INFO[0000] partition: col=length val=0.4
INFO[0000] prediction: 1
INFO[0000] num leaf: 3
INFO[0000] impurity: 0.444444
INFO[0000]
INFO[0000] name: L
INFO[0000] partition: <nil>
INFO[0000] prediction: 1
INFO[0000] num leaf: 1
INFO[0000] impurity: 0.000000
INFO[0000]
INFO[0000] name: R
INFO[0000] partition: col=length val=0.5
INFO[0000] prediction: 2
INFO[0000] num leaf: 2
INFO[0000] impurity: 0.500000
INFO[0000]
INFO[0000] name: L
INFO[0000] partition: <nil>
INFO[0000] prediction: 2
INFO[0000] num leaf: 1
INFO[0000] impurity: 0.000000
```

## Bias vs variance

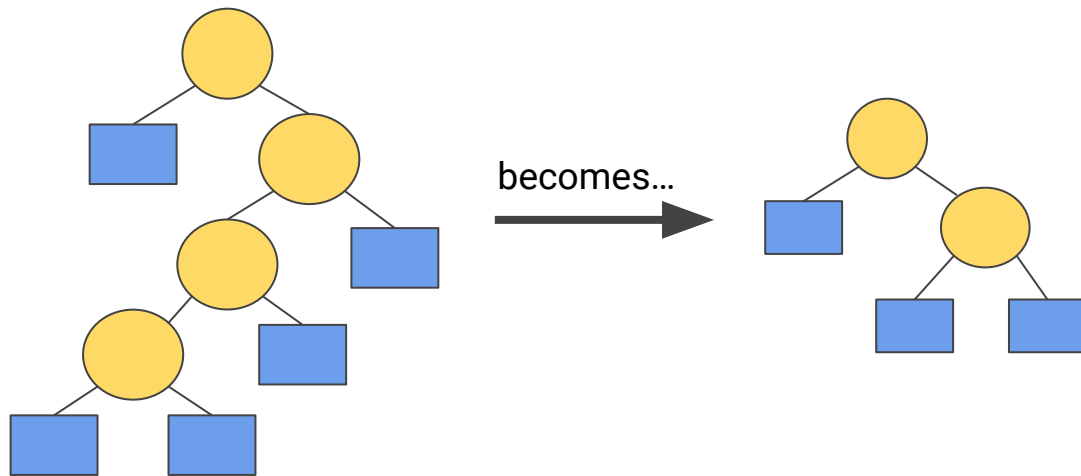
In most circumstances we want *some* error in our model to help decrease variance.





# Pruning

By removing steps we fix the bias (which is already low) and decrease variance



# Cost-complexity pruning

The best tree is the one that solves minimization problem:

$$R_{\alpha}(T) = R(T) + \alpha |f(T)|$$

$R(T)$  training error

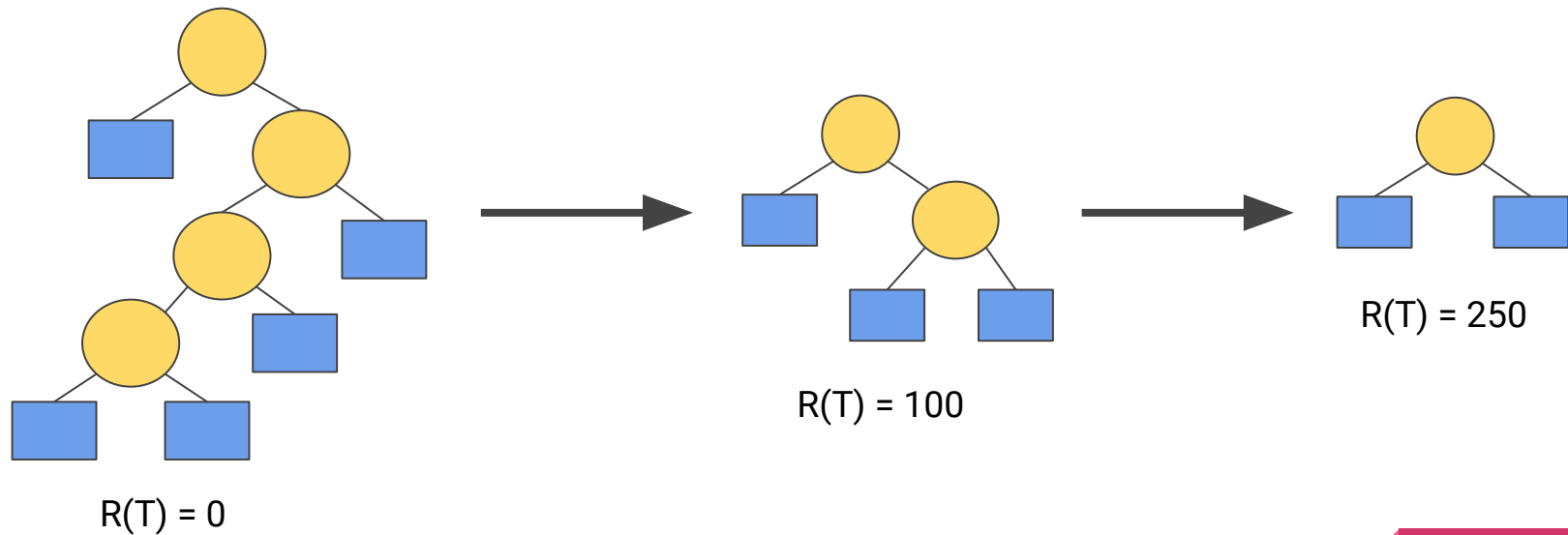
$\alpha$  hyperparameter

$f(T)$  returns set of leaves



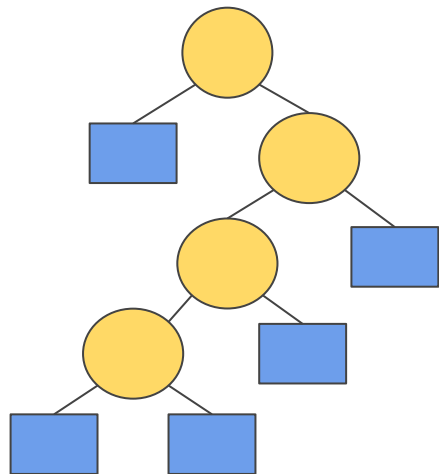
# Cost-complexity Pruning

Let's suppose each tree has the following error



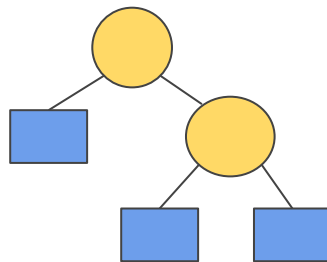
# Cost-complexity Pruning

If  $\alpha$  is fixed at 100, we get the following values for  $R_\alpha(T)$ :



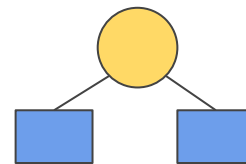
$$R(T) = 0$$
$$R_\alpha(T) = 0 + 100 * 5$$

$$= 500$$



$$R(T) = 100$$
$$R_\alpha(T) = 100 + 100 * 3$$

$$= 400$$



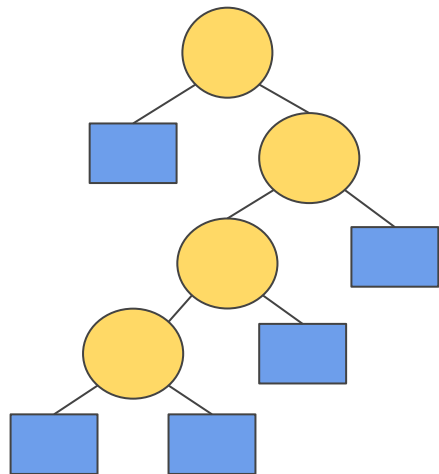
$$R(T) = 250$$
$$R_\alpha(T) = 250 + 100 * 2$$

$$= 450$$



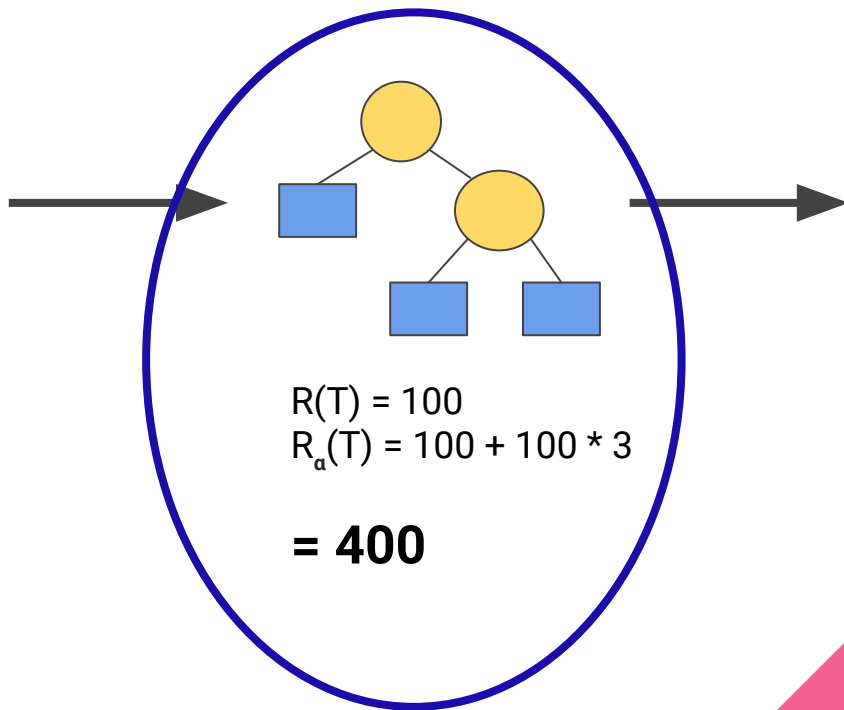
# Cost-complexity Pruning

The lowest value for  $R_a(T)$  is the winner!



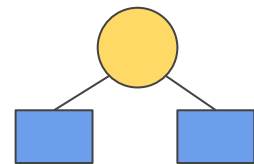
$$R(T) = 0$$
$$R_a(T) = 0 + 100 * 5$$

$$= 500$$



$$R(T) = 100$$
$$R_a(T) = 100 + 100 * 3$$

$$= 400$$

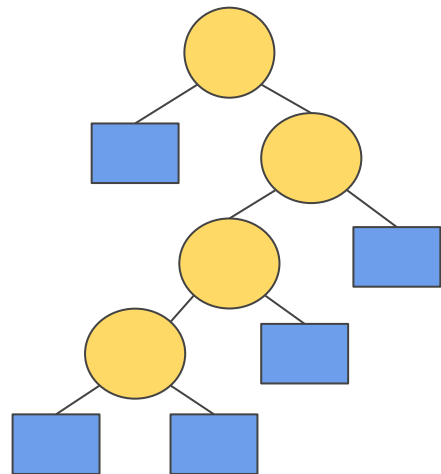


$$R(T) = 250$$
$$R_a(T) = 250 + 100 * 2$$

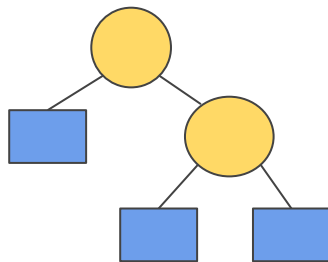
$$= 450$$

# Cost-complexity Pruning

But... any tree could be a winner, depending on alpha



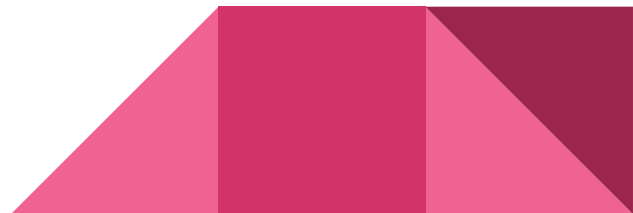
winner when  $\alpha = 0$



winner when  $\alpha = 100$



...



# Cost-complexity pruning

How do we know which alphas and subtrees to consider?



# Cost-complexity pruning

How do we know which alphas and subtrees to consider?

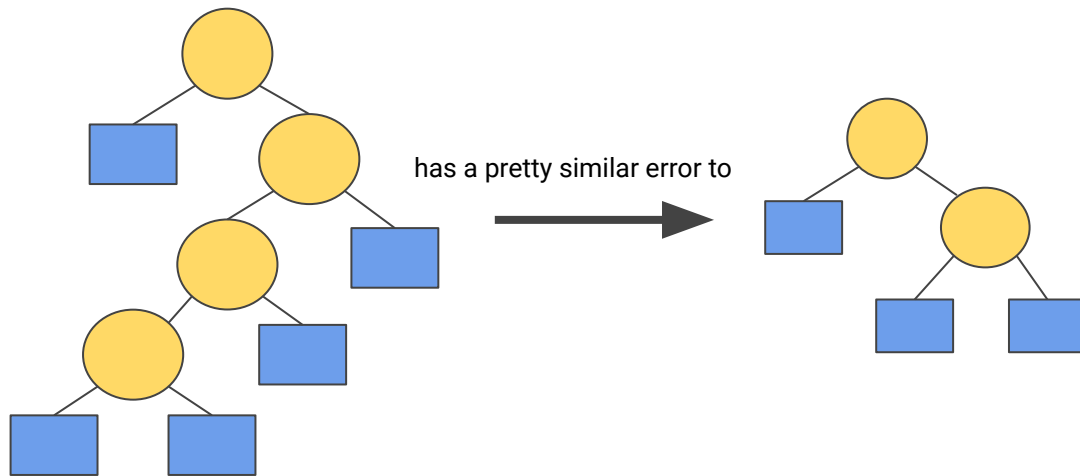
**Answer lies in the *other* name for cost-complexity pruning, “weakest-link pruning”**





## Cost-complexity pruning

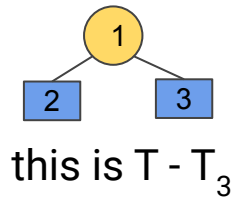
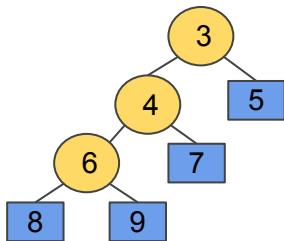
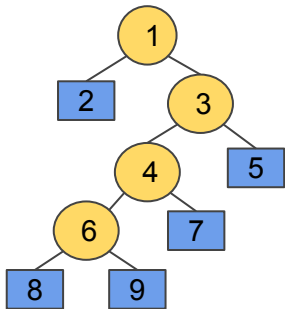
## Intuition..



# Cost-complexity pruning

Turning this into a minimization problem we get:

$$\min [R_{\alpha}(T - T_t) - R_{\alpha}(T)]$$



# Cost-complexity pruning

To find the value of  $\alpha$  for weakest-link subtree (rooted at  $t$ ):

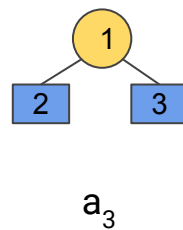
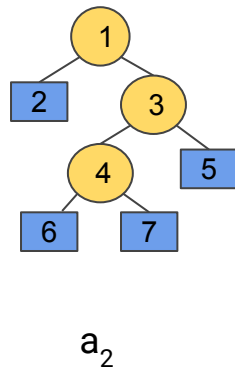
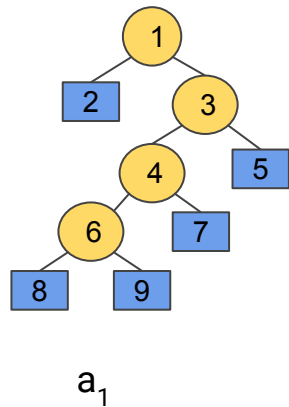
$$\begin{aligned} & \min [R_\alpha(T - T_t) - R_\alpha(T)] \\ &= \min [R(T - T_t) - R(T) + \alpha (|f(T - T_t)| - |f(T)|)] \\ &= \min [R(t) - R(T_t) + \alpha (1 - |f(T)|)] \end{aligned}$$

$$\alpha' = R(t) - R(T_t) / |f(T_t')| - 1$$



# Cost-complexity pruning

All of this to get a set of alphas and subtrees. Which do we choose?



# K-fold cross validation

Suppose one row represents all observed data points



Furry?	Length	Class
x	3.0 ft	snek



# K-fold cross validation

Generally we would split into two distinct sets, a **training** and a **testing** set.



# K-fold cross validation

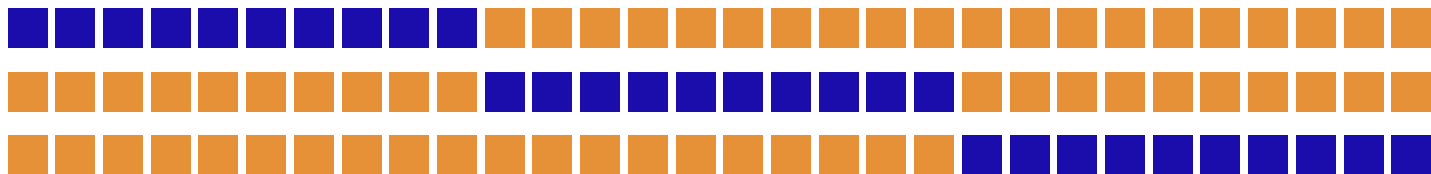
But doing so reduces the amount of data that the model was trained on



Furry?	Length	Class
x	3.0 ft	snek

# K-fold cross validation

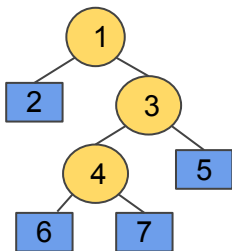
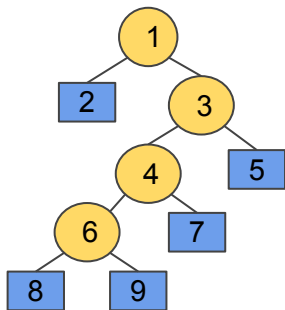
Instead we split the data points into K training / testing sets



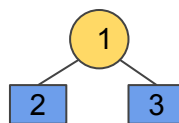


# K-fold cross validation

We build new subtrees using the alphas that we found

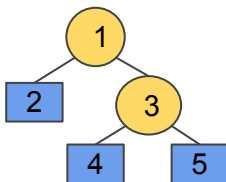
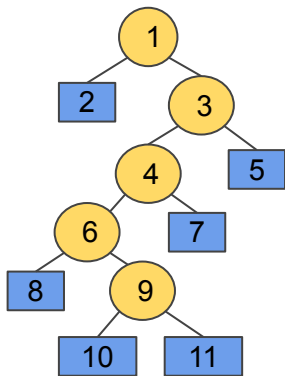


winner:  $a_1'$

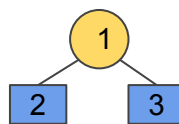


# K-fold cross validation

Here's a new set of trees for the next training set

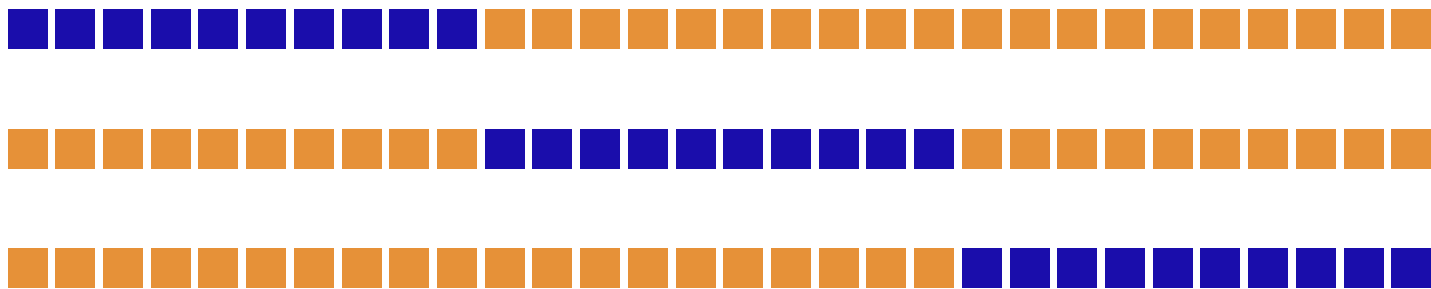


winner:  $a_2'$



# K-fold cross validation

Instead we split the data points into K training / testing sets



$$\frac{a'_1 + a'_2 + \dots + a'_K}{K} = a'$$

**this is the best value for  
alpha using entire dataset.**

# Current Scope

- **Classic ML**

- Datasets
  - Visualization
- Unsupervised learning
- Supervised learning
  - Decision trees (regression/classification)
  - Ensemble learning
    - Gradient tree boosting
- Model selection and validation
  - K-fold cross validation

- **Deep learning**

- I few years ago I built simple NN w/ backprop from scratch 😊
  - at [www.robertkotcher.me](http://www.robertkotcher.me)

# References

- StatQuest: <https://www.youtube.com/watch?v=D0efHEJsHo&t=779s>
- MLWiki: [http://mlwiki.org/index.php/Cost-Complexity\\_Pruning](http://mlwiki.org/index.php/Cost-Complexity_Pruning)
- ScikitLearn source: <https://github.com/scikit-learn/scikit-learn>

