Decision Trees

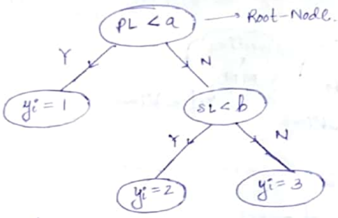
K-NN, NB, $\underbrace{\text{log. reg, lr. reg, SVM}}_{\text{(geometric, hyperplane)}}$. DT → (if.. else).

K-NN → instance based method; NB → probabilistic method

DT → simply a nested if else condition classifier. – programmatic interpretation

Ex → x_i = < SL, PL, SW, PW >

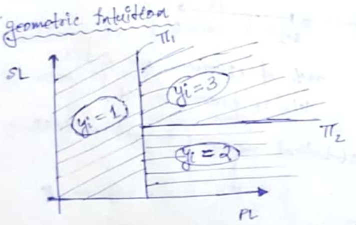
model
```
if PL < a:
    yi = 1
else
    if SL < b:
        yi = 2
    else
        yi = 3
```



* All leaf nodes are decision (classifn to a class).

* At all internal nodes, we make a decision.

geometric Intuition



DT — a set of axis parallel hyperplanes that divide the entire region into hypercubes & hypercuboids.

Building a DT : Entropy

say Y is a random variable that can take 'k' values – $y_1, y_2 \dots y_k$.

entropy
$$\boxed{H(Y) = \sum_{i=1}^{k} P(y_i)\log_b(P(y_i))}$$
b = 2 or b = e.
lg   ln

$$\boxed{P(y_i) = P(Y = y_i)}$$

say Y: play tennis — + : 9, – : 5. $P(y^+) = 9/14$, $P(y^-) = 5/14$.

$$H(Y) = -\left\{\frac{9}{14}\log\left(\frac{9}{14}\right) + \left(\frac{5}{14}\right)\log\left(\frac{5}{14}\right)\right\}.$$

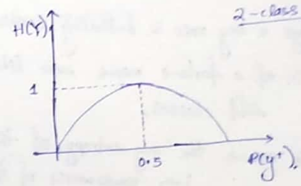
Properties of entropy (for 2-class classificatn).

case 1: $y_+ = 99\%$, $y_- = 1\%$ } $H(Y) \simeq 0.0801$

case 2: $y_+ = 50\%$, $y_- = 50\%$ } $H(Y) = 1.0$

case 3: $y_+ = 0\%$, $y_- = 100\%$ } $H(Y) = 0.0$

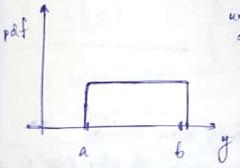
2-class case

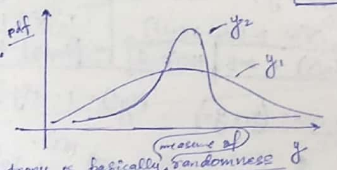


extending to multi-class →

* Y = $y_1, y_2 \dots y_k$. If all are equi-probable, H(Y) = 1 (max).
* If any 1 is most probable & others have prob. ≃ 0, then H(Y) lows.

$$\boxed{H(Y_1) > H(Y_2)}$$





* entropy is basically measure of randomness of y.

Information Gain

* Entropy of a table means basically w.r.t. the class label.
* we basically break the table on the basis of a feature, then the information gain of that feature is defined as

$$\boxed{IG(\text{feature}) = H(\text{original table}) - \text{weighted Avg. } H \text{ of all the constituent tables}}$$

Let D be the original table.

|D| = no of entries in D.

Let feature f_0 breaks D to D_1, D_2 & D_3.

Then,

$$\boxed{IG(f_0) = H(D) - \left\{ \frac{|D_1| * H(D_1) + |D_2| * H(D_2) + |D_3| * H(D_3)}{|D|} \right\}}$$

IG plays a key role in building decision trees.

* More IG of a feature means more likely is the feature in separating diff classes.
* More IG means the avg entropy of the constituent tables is less, i.e. randomness of the constituent tables is less.

Gini Impurity ~ similar to entropy.

$$\boxed{I_G(Y) = 1 - \sum_{i=1}^{k} (P(y_i))^2}$$

Case I: $P(y_+) = P(y_-) = 0.5$.

$I_G(Y) = 1 - (0.25 + 0.25)$

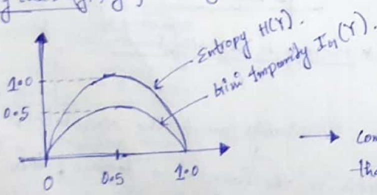
$\boxed{I_G(Y) = 0.5}$, $\boxed{H(Y) = 1}$

Case II: $P(y_+) = 1$, $P(y_-) = 0$.

$I_G(Y) = 1 - (1 + 0) = 0$.

$\boxed{I_G(Y) = 0}$, $\boxed{H(Y) = 0}$

2-category case:- (y_+, y_-) $\quad P(y^+) = 1 - P(y_-)$.



So, both have similar behaviour only. So, why study $I_G(Y)$?

→ Computing "log" takes more time than computing "squares".

* So, I_G is computationally efficient.

Constructing a DT

* Recursively at each level, select the feature with maximum IG. to partition the table.

Eventually we may end up in one of the following cases

① pure node → stop growing the node.

② very few pts of the other class. So, further growing tree may lead to overfitting. — so stop growing.

③ we are too deep down the tree. — stop growing.

If depth is small ⟹ underfit.

DT :- hyperparameter ⟹ depth.

Splitting numerical features

f_1	y
2.2	1
2.6	1
3.5	0
3.8	0
4.6	1
5.3	0

. For numerical features

① sort the entries in ascending order of that numerical feature.

② serially take each value in the feature as threshold τ & split on the decision

$$f_1 < \tau$$

③ select the τ with maximum IG.

It is very time consuming as we have to go through all n entries as threshold.

Feature Standardization

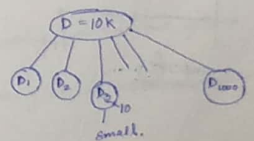
In DT, we don't depend on actual values of the features, but rather depend on their relative order.

So, we don't need to perform any kind of feature standardization.

Categorical features with many categories

pincode/ zipcode — 1000s of classes.

So, splitting to these many classes may cause some classes to have very few data pts.



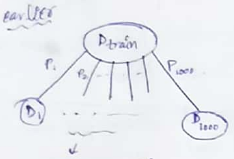
So, instead of using pincode as a categorical feature, we make use of a feature engineering hack to use it as a numerical feature.

We replace every P_j (pincode j) with $P(y_i=1 | P_j)$. i.e.,

probability of $y_i = 1$ given pincode j. $= \dfrac{\# y_i=1 \text{ and } P_j}{\# P_j}$

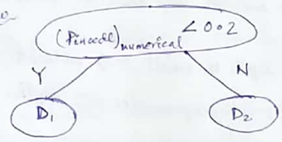
So, now we have a improved situation.

earlier



Most of these will be useless.

now



So, we get rid of problem of data sparsity.

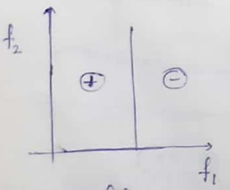
Overfitting and Underfitting

As depth ↑ → possibility of having very few pts @ a leaf node ↑ (sparsity).

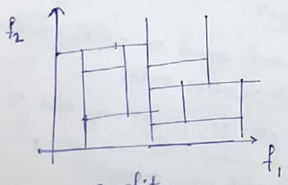
Also interpretability ↓ due to lots of constraints.

→ we maybe overfitting to noise.

depth determined via CV (cross validation)



underfit



overfit

Train & Run-time Complexity

train ~ $O(n \log n\, d)$ — $n \log n$: sorting; d: evaluating

n = # pts in D_{train}
d = dim.

Since d is a direct factor in time complexity, for large dimensions, DT mayn't be good.

After training :— storing D_{tree} ⇒ nested if-else conditions.

space = O(internal nodes) + O(leaves).

Runtime complexity = $O(k)$ k = max depth of any leaf node.

D.T. — large data, dimension small ✓
low latency. used by Amazon, Google and many diff internet applications.
↳ (RF, GBDT).

Regression using decision tree

Instead of entropy / Gini Impurity, we use MSE (mean squared error) or MAE (median absolute error).

Just like we find IG by the difference of entropy and weighted avg entropy, here also everything remains same.

* Prediction at each level $\hat{y}_i$ = mean(y_i) in that table.

Real world cases

① Imbalanced data → balance it. (upsampling / weights etc.). Impacts entropy/MSE calcn.

② large 'd':— @ each node, split & check for max. IG feature.
So, time complexity to train ↑.

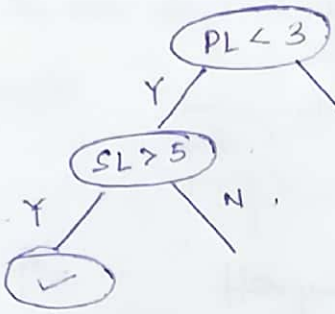
* For categorical feature, avoid one-hot encodings. as 1 feature gets converted to many features. Instead convert to numerical features just as explained earlier.

(3) Similarity Matrix — DT can't work.
Need to give features explicitly.

(4) Multi-class classifⁿ — Naturally can be extended, don't require One Vs Rest.

(5) Decision Surfaces — Non-linear axis parallel hyperplanes.

(6) Feature interactions — logical feature interactions are inbuilt in DT.



(PL < 3) AND (SL > 5).

(7) Outliers — depth ↑, outliers will impact.
Tree unstable.

(8) Interpretability — very much interpretable. unless depth not too much.

(9) Feature Importance:-

* normalised sum up reductions in H / I_G due to f_i — Measure of feature importance.
  * Basically how much does f_i helps in reducing the entropy.
    or how much does f_i helps in building D.T.

So, computing feature importance is straight forward.

Scanned by CamScanner