UNIT - III

1) Feaulture Generation and Feature Selection:

1 Extracting Meaning from Data:

How do Companies extract meaning from the data they have? In this chapter we hear from 2 people with very different approaches to that question - namely William Cukierski from kaggle & David Huffaker from Google.

(i) William Cukierski:

Will went to Cornell for a BA in physics & to Rutgers to get his PhD in biomedical engineering. He focused on cancer research, studying Pathology images. While working on writing his dissertation, he got more & more involved in Kaggle Competitions, finishing very near the top in multiple competitions, & now works for Kaggle.

After giving us some background in data science competitions & crowdsourcing, Will will explain how his Company works for the participants in the platform as well as for the larger community.

Will will then focus on feature extraction & feature selection. Quickly feature extraction refers to taking the raw dump of data u have & curating it more carefully, to avoid "garbage in, garbage out" scenario u get if u just feed raw data into an alg without forethought. Feature selection is the process of constructing a subset of the data | func's of the data to be the predictors or var's for ur models & alg's.

(ii) Background: Data Science Competitions:

There is a history in the machine learning community of data science competitions - where individuals or teams compete over a period of several weeks & months to design a prediction alg. what it predicts depends on the particular dataset, but some eg's include whether or not a given person will get in a car crash, or like a particular film. A training set is provided, an evaluation metric determined up front, & some set of rules is provided abt, for eg, how often competitors can submit their predictions, whether or not teams can merge into larger teams & soon

Some remarks abt data science competitions r warranted. First, data science competitions r part of the data science ecosystem - one of the cultural forces at play in the current data science landscape, & so aspiring data scientists ought to be awar them.

Second, creating these competitions puts one in a position to de data science, & define its scope. By thinking abt the challenges & that they've issued, it provides a set of eg's for us to explore central question: what is data science? This is not to say tha we will unquestionably accept such a def, but we can atleast use it as a starting point: what attributes of the existing compe -ons capture data science, & what aspects of data science r missin

Finally, competitors in the various competitions get ranked, & so o metric of a "top" data scientist could be their standings in ther competitions. But notice that many top data scientists, especially women, dont compete. In fact, there r few women at the top, & we think this phenomenon needs to be explicitly thought thru wh we expect top ranking to act as a proxy for data science talent

(iii) The kaggle Model:

"Being a data scientist is when u learn more & more abt more & more, until u know nothing abt everything" - Will Cukierski

Kaggle is a company whose tagline is, "we're making data science a sport" kaggle forms relationships with companies & with data scientists. For a fee, kaggle hosts competitions for businesses that essentially want to crowdsource to solve their data problems. kaggle provides the infrastructure & attracts the data science talent.

They also have inhouse a bunch of top-notch data scientists, including Will itself. The ~~...~~ companies r their paying customers & they provide datasets & data problems that they want solved. kaggle crowdsources these problems with data scientists around the world. Anyone can enter.

A Single Contestant:

In kaggle competitions, u r given a training set, & also a test set where y's r hidden, but the x's r given, so u just use ur model to get ur predicted x's for the test set & upload them into the kaggle system to see ur evaluation score. This way u don't share ur actual code with kaggle unless u win the prize. Note that even giving out just the x's is real inf - in particular it tells u, for eg, what sizes of x's ur alg should optimize for. Also for the purposes of the competition, there is a third hold-out set that contestants never have access to. You don't see the x's or the y's - that is used to determine the competition winner when the competition closes.

On kaggle, the participants r encouraged to submit their models up to 5 times a day during the Competitions, which last a few weeks. As Contestants submit their predictions, the kaggle leaderboard updates immediately to display the contestant's current evaluation metric on the hold-out test set.

With a sufficient no of Competitors doing this, we see a "leapfrogging" b/w them. It also establishes a band of accuracy around a problem that u generally don't have - in other words, given no other inf, with nobody else working on the problem u r working on, u don't know if ur 75% accurate model is the best possible.

This leapfrogging effect is good & bad. It encourages people to squeeze out better performing models, possibly at the risk of overfitting, but it also tends to make models much more complicated as they get better. One reasons u don't want Competitions lasting too long is that, after a while, the only way to inch up performance is to make things ridiculously complicated. For eg, the original Netflix prize lasted 2 yrs & the final winning model was too complicated for them to actually put into production.

Their Customers:

So why would Companies Pay to work with kaggle? The role that Kaggle is filling is the following: there's a mismatch b/w those who need analysis & those with skills. Even though companies desperately need analysis, they tend to hoard data; this is the biggest obstacle for success for those companies that even host a kaggle Competition. Many companies don't host Competitions at all, for similar reasons. Kaggle's innovation is that it Convinces businesses to share proprietary data with the benefit that their large data problems will be solved for them by crowdsourcing kaggle's ten thousands of datasciences around the world.

When Facebook was recently hiring data scientists, they hosted a Kaggle competition, where the prize was an interview. There were 422 Competitors. We think it's convenient for facebook to have interviewees for data scientists positions in such a posture of gratitude for the mere interview. Cathy thinks this distracts data scientists from asking hard questions abt what the data policies r and the underlying ethics of the Company.

2) Feature Selection:

The idea of feature selection is identifying the subset of data / transformed data that u want to put into ur model.

Prior to working at Kaggle, Will placed highly in Competitions (which is how he got the job), so he knows first hand what it takes to build effective predictive models. Feature selection is not only useful for winning Competitions - it's an imp part of building statistical models & alg's in general. Just bcoz u had data doesn't mean it all has to go into the model.

For eg, it's possible u have many redundancies & correlated var's in ur raw data, & so u don't want to include all those var's in ur model. Similarly u might want to construct new var's by transforming the vars with a logarithm, say, or turning a continuous var into a binary var before feeding them into the model.

Different branches of academia use different terms to describe the same thing. Statisticians say "explanatory data var's" & "dependent var's" & "predictors" when they're describing the subset of data that is the i/p to a model. Computer scientists say "features."

Feature extraction & selection r the most imp but under-rated steps of machine learning. Better features r better than better alg's

Is it possible, Will muses, that Norvig really wanted to say we have better features? u see, more data is sometimes just more data, but for the more interesting problems that Google faces, the feature landscape is complex/rich/nonlinear enough to benefit from collecting the data that supports those features.

Why? we r getting bigger & bigger datasets, but that's not always helpful. If the no of features is larger than the no of observations, or if we have a sparsity problem, then large isn't necessarily good. And if the huge data just makes it hard to manipulate bcoz of computational reasons without improving our signal, then that's a net -ve.

To improve the performance of ur ~~per~~ predictive models, u want to improve ur feature selection process.

Example: Users Retention: (holding back customers & trying to sell them more)

Let's give an eg for u to keep in mind before we dig into some possible methods. Suppose u have an app that u designed, let's call it Chasing Dragons (Game), & users pay a monthly subscription fee to use it. The more users u have, the more money u make. Suppose u realize that only 10% of new users ever come back after the 1st month. So u have 2 options to increase ur revenue: find a way to increase the retention rate of existing users, & acquire new users. Generally it costs less to keep an existing customer around than to market & advertise to new users.

But setting aside that particular cost-benefit analysis of acquistion & retention, let's choose to focus on ur user retention situation by building a model that predicts whether or not a new user will come back next month based on their behavior this month. u could build such a model in order to understand ur retention situation, but let's focus instead on building an alg that is highly accurate at predicting. u might want to use this model to give a free month to users who u predict need the extra incentive to stick around, for eg

A good, crude, simple model u could start out with would be logistic regression. This would give u the prob the user returns their second month conditional on their activities in the 1st month. You record each user's behavior for the first 30 days after sign-up. u could log every action the user took with timestamps: user clicked the button that said "level 6" at 5:22 am, user slew a dragon at 5:23 a.m, user got 22 points at 5:24 am, user was shown an ad for deodorant at 5:25 am. This would be the data collection phase. Any action the user could take gets recorded.

Notice that some users might have thousands of such actions, and other users might have only a few. These would all be stored in time stamped event logs. You'd then need to process these logs down to a dataset with rows & col's, where each row was a user & each col was a feature. At this point, u shouldn't be selective; u're in the feature generation phase. So ur data science team (game designers, s/w eng's, stat's, & marketing folks) might sit down & brainstorm features. Here r some eg's:

-> No of days the user visited in the 1st month
-> Amt of time until second visit
-> No of points on day j for j = 1, ..., 30 (this would be 30 seperate features)

→ Total no of points in 1st month (Sum of the other features)
→ Did user fill out Charting Dragon profile (binary 1 or 0)
→ Age & gender of user
→ Screen size of device.

f we buit new set of features from the original feature set

3) **Feature Generation & Feature Extraction:**

This process we just went thru of brainstorming a list of featu for charting Dragons is the process of feature generation or feature ~~selection~~ extraction. This process is as much of an art as a science. It's good to have a domain expert around for this process, but it's also good to use ur imagination.

In today's technology environment, we're in a position where u can generate lots of features thru logging. Contrast this with other contexts like surveys, for eg - u r lucky if u can get a survey respondent to answer 20 questions, let alone hundreds

But how many of these features r just ~~alon~~ noise? In this environment, when u can capture a lot of data, not all of it might be actually useful inf

Keep in mind that ultimately you're limited in the features u have access to in 2 ways: whether or not it's possible to even capture the inf, & whether or not it even occurs to u at all to try to capture it. u can think of inf as falling into the following buckets.

(i) Relevant & useful, but its impossible to capture it!

u should keep in mind that there's a lot of inf that ur not capturing abt users - how much free time do they actually have? what other apps have they downloaded? Are they unemployed? some of this inf might be more predictive of whether or not they return next month. There's not much u can do abt this, except that it's possible that some of the data u r able to capture serves as a proxy by being highly correlated with these unobserved pieces of inf e.g. if they play the game every night at 3 am they might suffer from insomnia, or they might work the night shift.

(ii) Relevant & useful, possible to log it, & u did:

Thankfully it occured to u to log it during ur brainstorming session. It's great that u chose to log it, but just bcoz u chose to log it doesn't mean u know that it's relevant or useful, so that's what you'd like ur feature selection process to discover.

Scanned by TapScanner

Feature Selection Algorithms:

Filters: Filter type methods select var's regardless of the model. They r based only on general features like the correlation with the var to predict. Filter methods suppress the least learning var's. The other var's will be a part of a classification or a regression model used to classify & to predict data. These methods r particularly effective in computation time & robust to overfitting.

However filter methods tend to select redundant var's b/w (no longer useful) they do not consider the relationships b/w var's. Therefore, they r mainly used as a pre-process method. Filters order possible features with respect to a ranking based on a metric & statistic, such as correlation with the outcome var.

However, the problem with filters is that u get correlated features. In other words, the filter doesn't care abt redundancy. And by treating the features as independent, u r not taking into a/c possible interactions.

This isn't always bad & it isn't always good. On the one hand, 2 redundant features can be more powerful when they r both used; & on the other hand, something that appears useless alone could actually help when combined with another possibly useless looking feature that an interaction would capture.

Set of features → Selecting best subset → M learning alg → Performance

2) Wrappers: Wrapper feature selection tries to find subsets of features, of some fixed size, that will do the trick. However, as anyone who has studied combinations & permutations knows, the no of possible size k subsets of 'n' things, called $\binom{n}{k}$, grows exponentially. So, there's a nasty opportunity for overfitting by doing this.

There r 2 aspects to wrappers that we need to consider:
1) Selecting an alg to use to select features.
2) Deciding on a selecting criteria & filter to decide that ur set of features is good.

Set of all features → Generate subsets → ML alg → Performance (Selecting best subset)

(i) Selecting an algorithm:
Let's first talk abt a set of alg's that fall under the category of stepwise regression, a method for feature selection that involves selecting features according to some selection criterion by either adding & substracting features to a regression model in a

systematic way there r 3 primary methods of stepwise regression:

→ Forward Selection
→ Backward elimination
→ Combined approach (forward & backward)

→ Forward Selection:
In fwd selection u start with a regression model with no features & gradually add one feature at a time according to which feature improves the model the most based on a selection criterion. Th looks like this: build all possible regression models with a single predictor. Pick the best. Now try all possible models that include the best predictor & a second predictor. Pick the best of those. u keep adding one feature at a time, & u stop when ur selection no longer improves, but instead gets worse.

→ Backward elimination:
In backward elimination, u start with a regression model that includes all the features, & u gradually remove one feature at a time according to the feature whose removal makes the biggest improvement in the selection criterion. u stop removing features when removing the feature makes the selection criterion get worse.

→ Combined approach:
Most subset methods r capturing some flavor of min redundancy - max - relevance. So, for eg, u could have a greedy alg that starts with the best feature, takes a few more highly ranked, removes the worst, & so on. This is hybrid approach with a filter method.

3) Decision Trees: A decision tree is a decision support tool that uses a tree-like graph & model of decisions & their possible consequences, including chance event outcomes, resource costs, & utility.

A decision tree is a structure that includes a root node, branches, & leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, & each leaf node holds a class label. The topmost node in the tree is the root node.

It breaks down a data set into smaller & smaller subsets while at the same time an associated decision tree is developed. The final result is a tree with decision nodes & leaf nodes. Leaf node represents a classification or decision.

Relevant & useful, possible to log it, but u didn't:

It could be that u didn't think to record whether users uploaded photo of themselves to their profile, & this action is highly predictive of their likelihood to return. You're human, so sometimes you'll end up leaving out really important stuff, but this shows that ur own imagination is a constraint in feature selection. One of the key ways to avoid missing useful features is by doing usability studies, to help u think thru the user experience & what aspects of it you'd like to capture.

iv) Not relevant or useful, but u don't know that & log it:

This is what feature selection is all abt – you've logged it, but u don't actually need it & you'd like to able to know that.

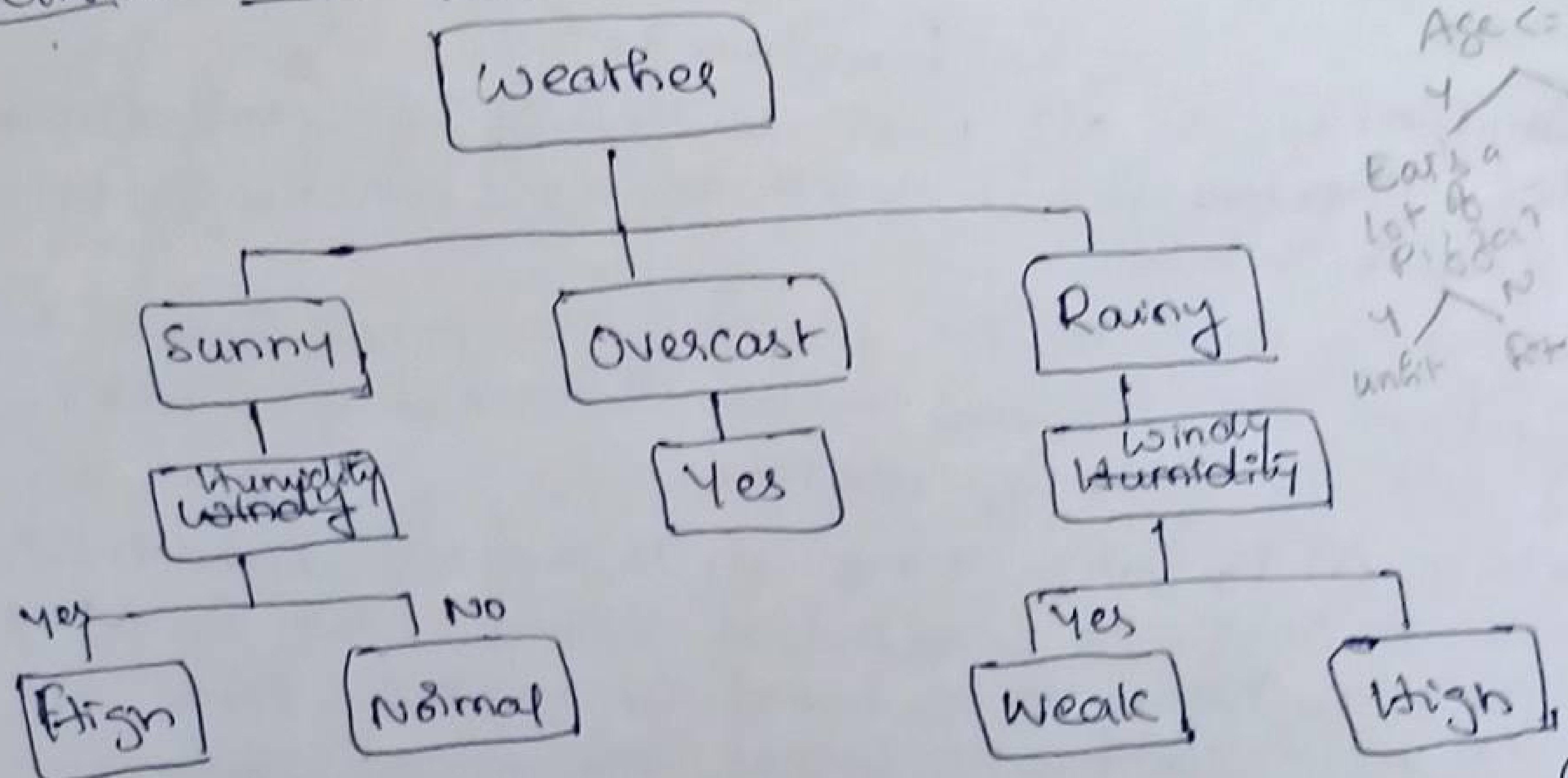
v) Not relevant or useful, & u either can't capture it or it didn't occur to you:

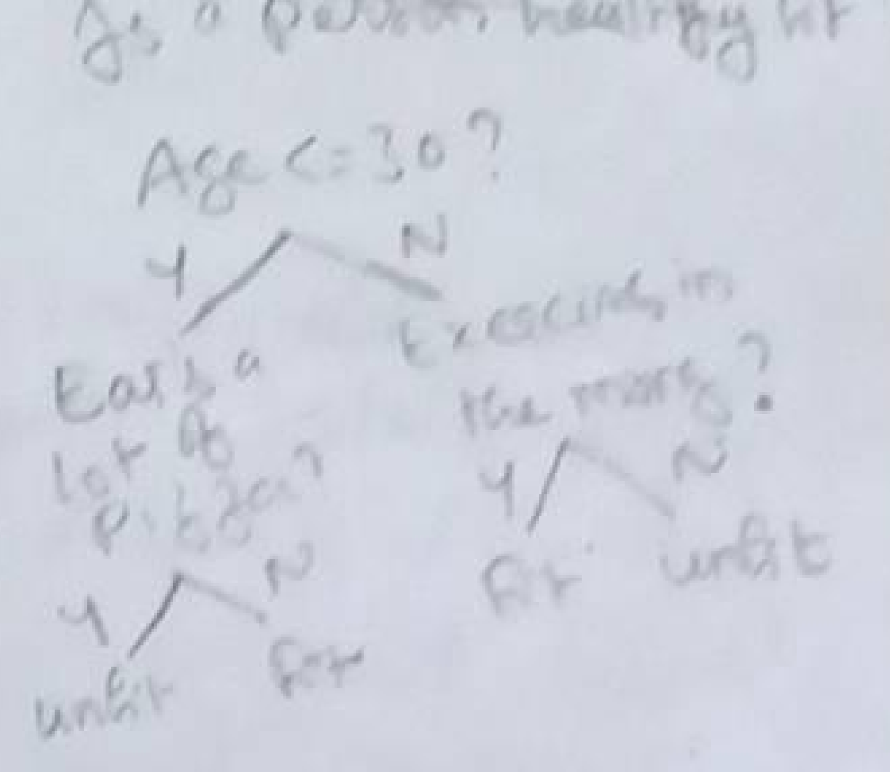
That's OK! It's not taking up space, & u don't need it.

4) Feature Selection Algorithms:

a) Filters: Filters order possible features with respect to a ranking based on a metric or statistic, such as correlation with the outcome var. This is sometimes good on a first pass over the space of features, bcoz they then take account of the predictive power of individual features. However, the problem with filters is that u get correlated features. In other words, the filter doesn't care abt redundancy. And by treating the features as independent, you're not taking into a/c possible interactions.

Decision tree (cont):





The core algorithm for building decision tree is ID3 & J48 (SLIQ, C4.5 &

A decision tree can easily be transformed to a set of rules by mapping from the root node to the leaf nodes one by one.

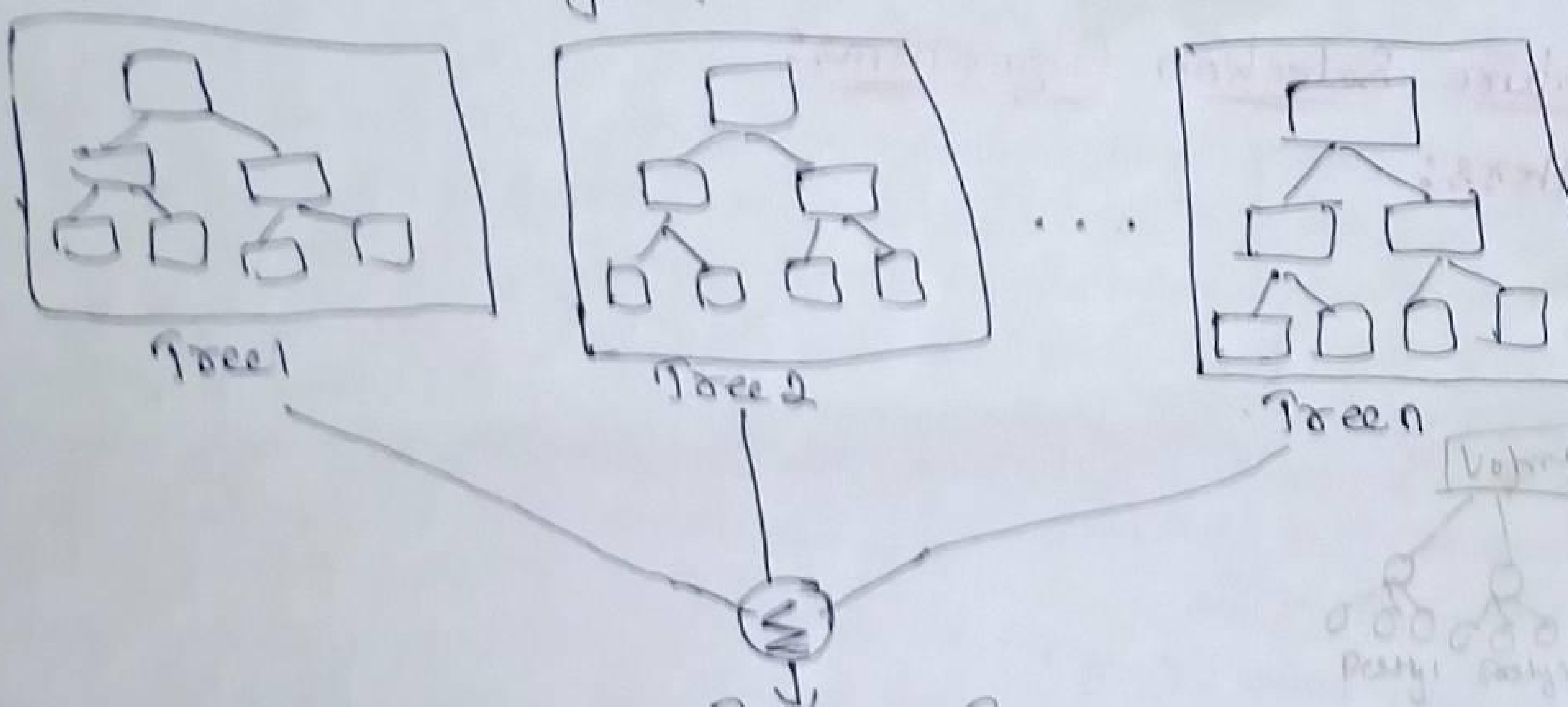
R_1: If (Outlook Weather = Sunny) & (Humidity = Yes) Then HIGH

i. Random Forest: Random forest is a trademark term for an ensemble classifier that consists of many decision trees & o/p's the class that is the mode of the classes o/p by individual trees.

Random forests r collections of trees, all slightly different. It randomize the alg, not the training data. How u randomize depends on the alg, for e.g's: don't pick the best, pick randomly from the k best options.

It generally improves decision trees decisions. Unlike single decision trees which r likely to suffer from high variance or high bias random forests use avg'ing to a natural balance b/w the 2 extremes.

The Random forest is one of the best among classification alg's able to classify large amts of data with accuracy. Random forest alg is used by data scientists at Rose Data Science Professional Practice group.



Random forest.

The Random forest alg was developed by Leo Breiman & Adle Cutler. Random forest grows many classification trees. Each tree is grown as follows:

1) If the no of cases in the training set is N, Sample N cases at random, from the original data. This sample will be the training set for growing the tree.
2) If there r M i/p var's, a no 'm' is specified such that at each node, m var's r selected at random out of 'M' & the best split on these m is used to split the node. The value of 'm' is held constant during the forest growing.
3) Each tree is grown to the largest extent possible.

R, Python & WEKA have robust packages to implement Random forests.