466 Final Presentation

Jack Kelly and Panya Ou

Topic : Machining Learning using an ID3 Decision Tree from a dataset of Dapp usage

Main Goals:

- Learning more about this specific Machine Learning algorithm
- Implementation of this ID3 algorithm from scratch
- Building a decision tree based on datasets from the real world

Dataset Attributes

-11 Dapps included
-User dapp use
-wallet balances(target variable)

Our topic - User patterns on the Aptos Blockchain

What we want to explore / Our questions

- Can we use a subset of dapps to predict what other dapps a user uses
- Can we predict the account balance of a user based off what dapps they use?

Our approach - Gathering Data

Data Gathered

- 30k user accounts(wallet addresses)
- The Addresses of the most popular applications

Data used:

- Protocol whitelists
- Use on-chain data to gather users recent transactions

Account Addresses

```
0x9ee9892d8600ed0bf65173d801ab75204a16ac2c6f190454a3b98f6bcb99d915,
0x6d19623096d04c409b180b936f23a47d48aa4c3a48ebc21909d65308b1c7ccee,
0x81c87115a1739d83789e1ffaa78b6d039992042890a208c20b62fc1b95596b66,
0xb149debde4265c518efb8915c2ee0ddef2880701d5b0b16fd0adebfd0015b8cd,
0x61c277a88918d1a095a951b9b2d3e0126baaaecb8e4341ad65c7205f1f8d58bb,
0xe41046a66e65afc778d76fe8044c8d8732f1d0b23c98c9f931d84064a5a8f494,
0xf1c2bde3622781fb91d176c15b22078a915ef86737706310bb033036b27c5327,
0x5735b68ad8f9d165fb05bf4e17c48a35c60d17a7a26734fdfc99511e15c0a8c5,
```

Some context And Terminology

What is Aptos?

- New Layer 1 Blockchain
- Led by facebook's former Libra team

Why is Aptos interesting?

- Move based Blockchain
- High TPS
- Low Blockfinality time

```
module liquidswap_lp::coins {
    use std::signer;
    use std::string::utf8;

    use aptos_framework::coin::{Self, MintCapability, BurnCapability};

    /// Represents test USDT coin.
    struct USDT {}

    /// Represents test BTC coin.
    struct BTC {}

    /// Storing mint/burn capabilities for `USDT` and `BTC` coins under user
    struct Caps<phantom CoinType> has key {
        mint: MintCapability<CoinType>,
        burn: BurnCapability<CoinType>,
    }

    /// Initializes `BTC` and `USDT` coins.
    public entry fun register_coins(token_admin: &signer) {
        let (btc_b, btc_f, btc_m) =
            coin::initialize<BTC>(token_admin,
                utf8(b"Bitcoin"), utf8(b"BTC"), 8, true);
        let (usdt_b, usdt_f, usdt_m) =
            coin::initialize<USDT>(token_admin,
                utf8(b"Tether"), utf8(b"USDT"), 6, true);

        coin::destroy_freeze_cap(btc_f);
        coin::destroy_freeze_cap(usdt_f);

        move_to(token_admin, Caps<BTC> { mint: btc_m, burn: btc_b });
        move_to(token_admin, Caps<USDT> { mint: usdt_m, burn: usdt_b });
    }
```

Some Move Code

Terminology

- DeFi - Decentralized finance (ex. lending/borrowing, Liquidity pools)
- Dapp - Decentralized app

  This is how users interact with the chain
- NFTs - domain names, art, utility tokens

Our process

Verify the accounts are valid + the same for the dapp addresses

Load and clean user account data

these users will be split up into training and testing sets this list of users was pulled from pontem's space pirate whitelist. Pontem is a dapp that allows users to earn 'interest' on their crypto holdings by providing liquidity through their dex (sets of liquidity pools).

```python
import pandas as pd
#load the users csv file (
users = open('users.csv').readlines()
user_temp = []
for user in users:
    if len(user)>=66:
        user_temp.append(user.split(',')[0])
print("ALL user ADDR COUNT:",len(users))
users = user_temp

print("PROPPER ADDR COUNT:",len(users))
```

Using aptos API

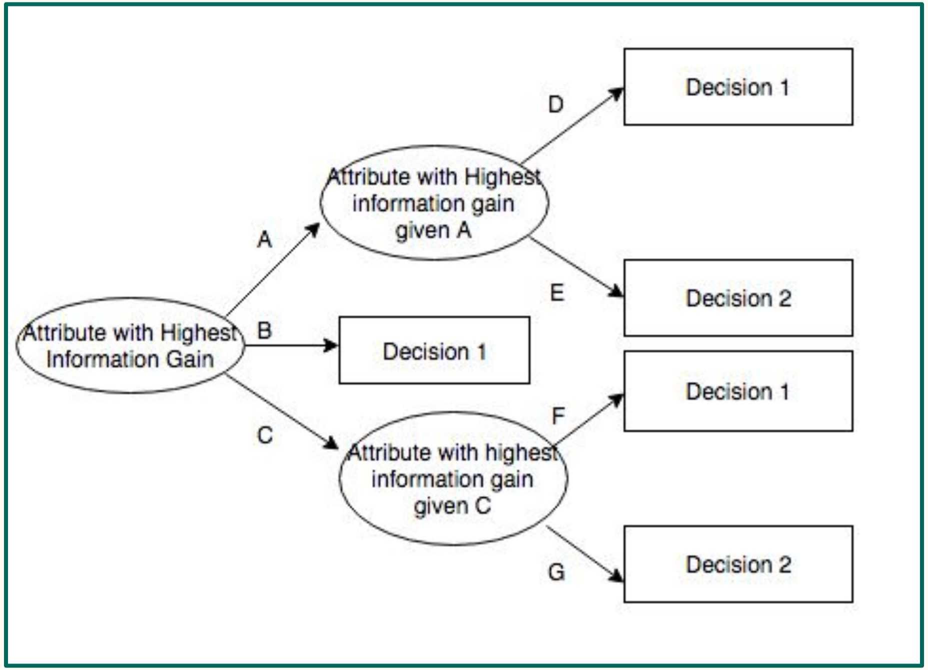
```python
# from flask import jsonify
import requests
import random
NODE_URL = "https://fullnode.mainnet.aptoslabs.com/v1"
def get_account_transactions(address):
    user_dapps = {}
    response = requests.get(
        f"{NODE_URL}/accounts/{address}/transactions",
        params={"limit": 50}
        )
    txs = response.json()
    # print(len(txs))
    txns_temp = []
    for tx in txs:
        try:
            txn = {}
            payload = tx['payload']
            to = payload['function'].split('::')[0]
            func = payload['function'].split('::')[2]
            mod = payload['function'].split('::')[1]
            txn['function'] = func
            txn['module'] = mod
            if to in dapp_dic.keys():
                dapp_name = dapp_dic[to]
                if dapp_name in user_dapps.keys():
                    user_dapps[dapp_name] = 1 + user_dapps[dapp_name]
                else:
                    user_dapps[dapp_name] = 1
```

What is the ID3 Decision Tree Algorithm ?

ID3 stands for Iterative Dichotomiser 3 and is named that way because the algorithm iteratively (repeatedly) divides (dichotomizes) features into two or more groups at each step.

It was invented by Ross Quinlan and uses a top-down approach.

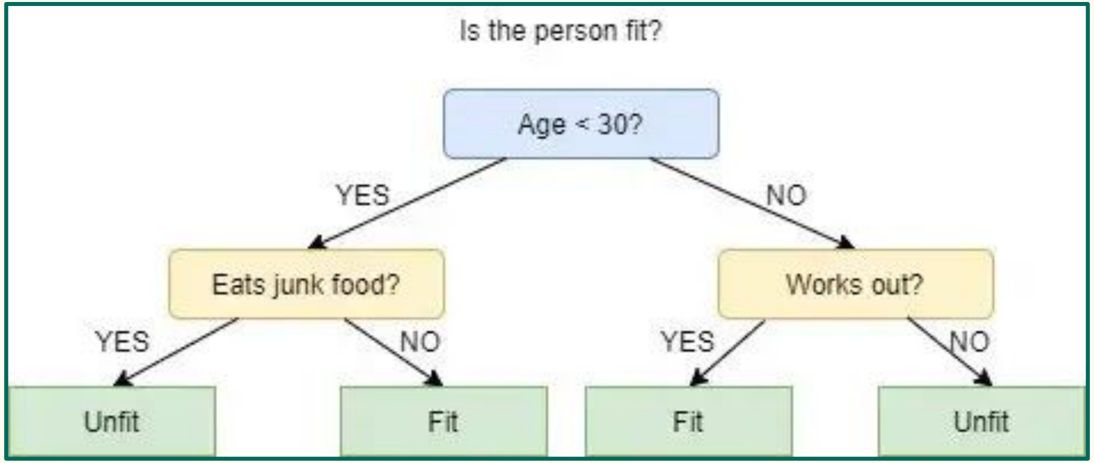
That just means we start building the tree from the top and build our way down to the leaf nodes.



What is a Decision Tree?

A decision tree is a structure that contains nodes and edges and is built from a dataset in our specific example.

Each node is used to **make a decision** or **represent an outcome (This is known as a leaf node)**



What our decision tree does

Given our dataset It will predict if a user is a Big-Shark/Whale based on the dapps that the used.

(A big-shark is just an individual who holds a lot of tokens)

However, the algorithm I created does not only apply to my own inputs, it can apply to any given input given the correct text format.

Our Dataset

KanaLabs	BlueMove	Hippo-Labs	Souffl3	Aries	liquidSwap	AnimeSwap	Topaz	Argo	HoustonSwap	Cetus	Big_Shark
L-Ks	L-Be	M-Hs	M-S3	N-As	N-lp	L-Ap	N-Tz	N-Ao	N-Hp	N-Cs	Yes
N-Ks	N-Be	N-Hs	N-S3	N-As	N-lp	H-Ap	N-Tz	N-Ao	N-Hp	N-Cs	Yes
N-Ks	M-Be	N-Hs	L-S3	N-As	L-lp	H-Ap	N-Tz	N-Ao	N-Hp	N-Cs	No
N-Ks	N-Be	N-Hs	H-S3	N-As	L-lp	N-Ap	N-Tz	N-Ao	N-Hp	N-Cs	Yes
N-Ks	N-Be	N-Hs	H-S3	N-As	N-lp	N-Ap	N-Tz	N-Ao	N-Hp	L-Cs	No
N-Ks	N-Be	N-Hs	M-S3	N-As	L-lp	L-Ap	N-Tz	N-Ao	N-Hp	N-Cs	No
N-Ks	L-Be	N-Hs	L-S3	N-As	N-lp	N-Ap	N-Tz	N-Ao	N-Hp	N-Cs	Yes
N-Ks	H-Be	N-Hs	M-S3	N-As	N-lp	N-Ap	L-Tz	N-Ao	N-Hp	N-Cs	No
N-Ks	N-Be	N-Hs	H-S3	N-As	N-lp	H-Ap	N-Tz	N-Ao	N-Hp	N-Cs	No
N-Ks	N-Be	N-Hs	H-S3	N-As	N-lp	N-Ap	L-Tz	N-Ao	N-Hp	N-Cs	No
N-Ks	N-Be	L-Hs	N-S3	N-As	N-lp	N-Ap	N-Tz	N-Ao	N-Hp	N-Cs	Yes
N-Ks	N-Be	N-Hs	L-S3	N-As	N-lp	N-Ap	N-Tz	N-Ao	N-Hp	N-Cs	Yes
N-Ks	N-Be	N-Hs	N-S3	N-As	H-lp	N-Ap	N-Tz	N-Ao	N-Hp	N-Cs	Yes
N-Ks	N-Be	L-Hs	N-S3	L-As	H-lp	H-Ap	N-Tz	N-Ao	N-Hp	N-Cs	No
N-Ks	N-Be	N-Hs	L-S3	N-As	N-lp	N-Ap	N-Tz	N-Ao	N-Hp	N-Cs	No
N-Ks	M-Be	N-Hs	N-S3	N-As	H-lp	N-Ap	N-Tz	N-Ao	N-Hp	N-Cs	No
N-Ks	L-Be	L-Hs	N-S3	N-As	N-lp	L-Ap	N-Tz	N-Ao	N-Hp	N-Cs	Yes

BlueMove	Souffl3	liquidSwap		
1	4	0	0	Yes
0	0	0	0	No
4	2	1	0	No
0	11	1	0	Yes
0	11	0	0	No
0	4	1	0	No
2	2	0	0	Yes
5	3	0	0	No
0	15	0	0	No

N: no use
L: $<$ 2 uses
M: $<$5 uses
H : $>$ 5 uses

The Process - What is Calculated to build the DT?

Calculations of Information gain and Entropy are used to build the Decision Tree.

Entropy: Entropy is the measure of disorder within a dataset.

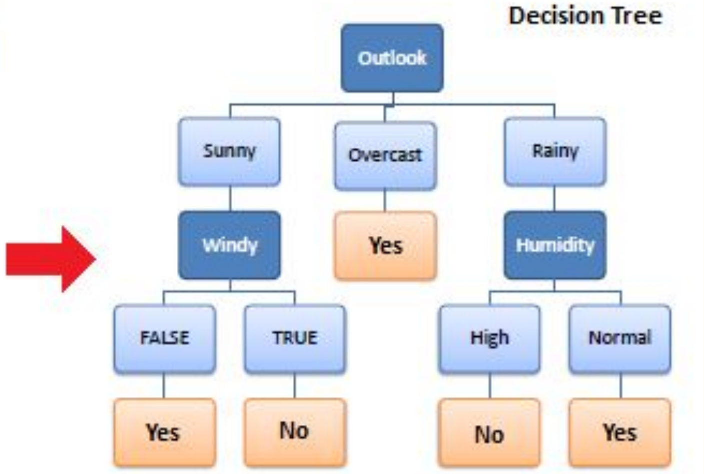
That disorder just means the randomness of a system. The higher the entropy, the higher the disorder and the higher the randomness.

Information Gain: Calculates the reduction in entropy and measures how well a feature classifies the targeted class. The **highest information** gain would be selected the best one.

Equation: **Entropy(S) = - $\sum p_i * \log_2(p_i)$; i = 1 to n**

n is the total number of attributes in the Predictors column

Predictors				Target
Outlook	**Temp.**	**Humidity**	**Windy**	**Play Golf**
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



p_i is the **probability of class 'i'** or the ratio of "*number of rows with class i in the target column*" to the "*total number of rows*" in the dataset.

Example: Entropy(**Outlook**) —> **- $\sum$ Sunny * $\log_2$(Sunny) → -$\sum$ $\frac{3}{5}$* $\log_2(\frac{3}{5})$ + ...**

Equation: **$Entropy(S) = - \sum p_i * \log_2(p_i)$; $i = 1$ to n**

n is the total number of attributes in the Predictors column

p_i is the **probability of class 'i'** or the ratio of "*number of rows with class i in the target column*" to the "*total number of rows*" in the dataset.

Example: Entropy(**Outlook**) —> **$- \sum$ Sunny $* \log_2$(Sunny) → $-\sum \frac{3}{5} * \log_2(\frac{3}{5})$ + ...**

Equation: Information Gain

(Note* Entropy(S) = - $\sum p_i * \log_2(p_i)$; i = 1 to n)

Information gain: **MAX(For every S: Entropy(Parent) - ∑Entropy(S))**

Parent: is the probability or ratio of how many **Big-Sharks** are in the dataset.

S: is the **probability** of the **column attributes where big-shark is yes.**

KanaLabs	BlueMove	Hippo-Labs	Souffl3	Aries	liquidSwap	AnimeSwap	Topaz	Argo	HoustonSwap	Cetus	Big_Shark
L-Ks	L-Be	M-Hs	M-S3	N-As	N-lp	L-Ap	N-Tz	N-Ao	N-Hp	N-Cs	Yes
N-Ks	N-Be	N-Hs	N-S3	N-As	N-lp	H-Ap	N-Tz	N-Ao	N-Hp	N-Cs	No
N-Ks	M-Be	N-Hs	L-S3	N-As	L-lp	H-Ap	N-Tz	N-Ao	N-Hp	N-Cs	No

Given our input case: **Big-Sharks** is the parent and all the other column attributes would be **S**

ID3 Algorithm Steps

1. Calculate the Information Gain of each feature.
2. Considering that all rows don't belong to the same class, split the dataset **S** into subsets using the feature for which the Information Gain is maximum.
3. Make a decision tree node using the feature with the maximum Information gain.
4. If all rows belong to the same class, make the current node as a leaf node with the class as its label.
5. Repeat for the remaining features until we run out of all features, or the decision tree has all leaf nodes.

DEMO

Yay.

https://colab.research.google.com/drive/1gDgk9yiJJ6zCFATU651pXPkFNhHJZU2J#scrollTo=Yht_WOC07vJZ

Now the input files and output.

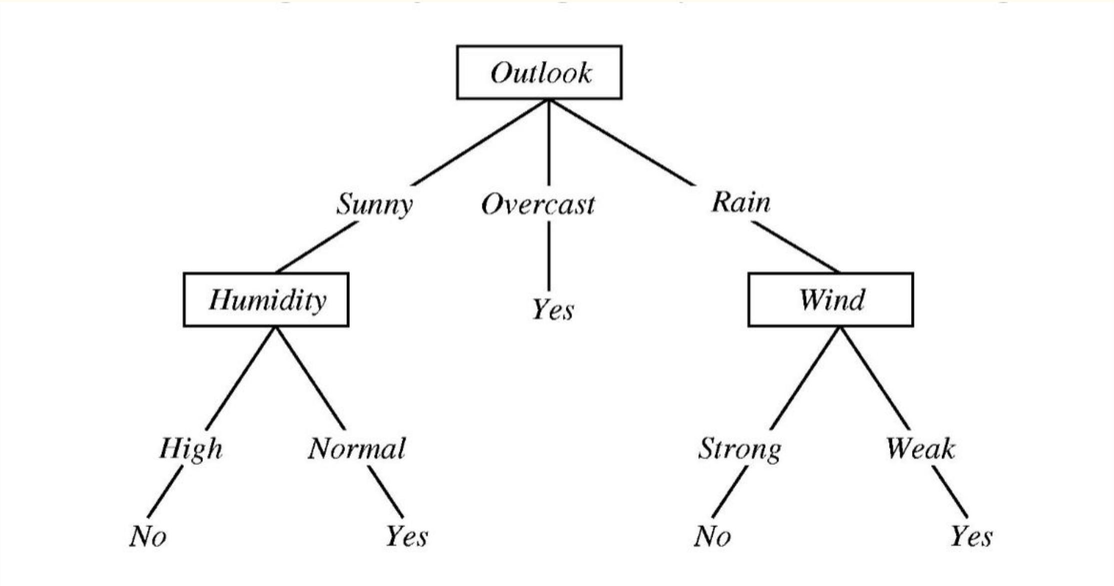
Comparison between inputs and text files

Day	Outlook	Temperature	Humidity	Wind	PlayTenn
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

```
Outlook  Temperature  Humidity  Wind  PlayTennis
Sunny  Hot  High  Weak  No
Sunny  Hot  High  Strong  No
Overcast  Hot  High  Weak  Yes
Rain  Mild  High  Weak  Yes
Rain  Cool  Normal  Weak  Yes
Rain  Cool  Normal  Strong  No
Overcast  Cool  Normal  Strong  Yes
Sunny  Mild  High  Weak  No
Sunny  Cool  Normal  Weak  Yes
Rain  Mild  Normal  Weak  Yes
Sunny  Mild  Normal  Strong  Yes
Overcast  Mild  High  Strong  Yes
Overcast  Hot  Normal  Weak  Yes
Rain  Mild  High  Strong  No
```

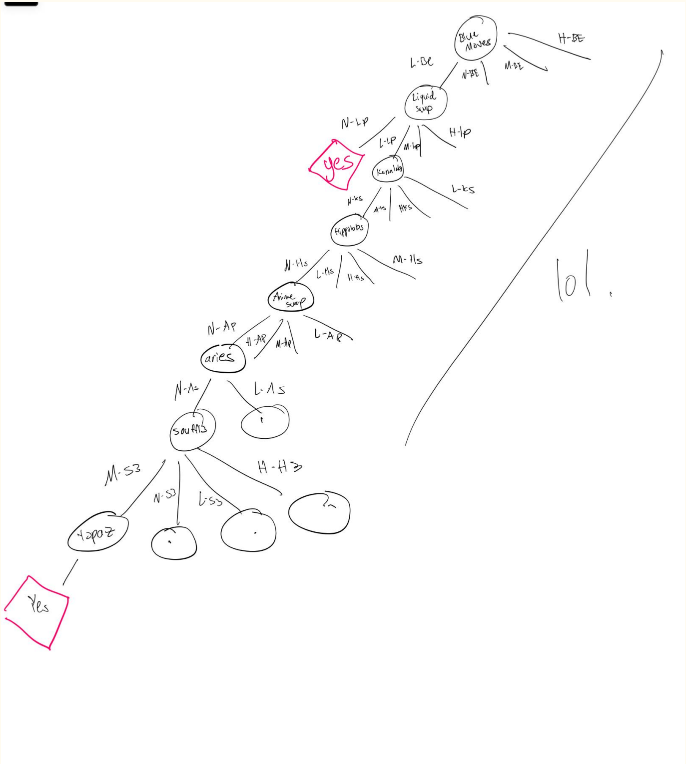
EASY EXAMPLE

I'm using an easier example so it's easier to understand what my algorithm **outputs** and and how to **interpret** it.



The Decision Tree

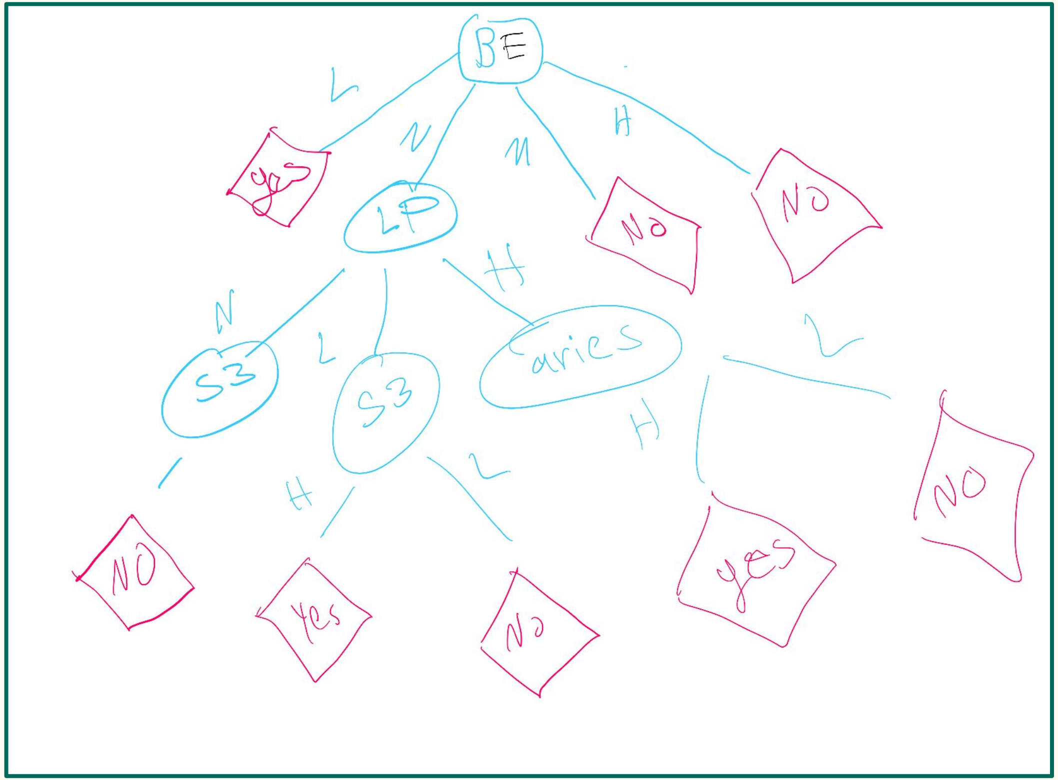
Open Images



Our example EASY

Note*

The reason this tree doesn't always have 4 attributes for each node is due to our data inputs having only 1-3 unique attributes and not always 4.



Other methods we Explored

KNN - we attempted to group users into Categories based on the types of apps used

```
def user_distance_matrix(users_used_dapps):
    user_addrs = list(users_used_dapps.keys())
    distance_matrix = np.zeros((len(user_addrs),len(user_addrs
    for i in range(len(user_addrs)):
        for j in range(len(user_addrs)):
            if i != j:
                distance_matrix[i][j] = user_distance(users_us
    return distance_matrix

distance_matrix = user_distance_matrix(users_used_dapps)
print(distance_matrix)
```

```
def dapp_diff(user1,user2):
    diff_dic = {}
    for dapp in dapp_ls:
        if dapp in user1.keys() and dapp in user2.keys():
            diff_dic[dapp] = abs(user1[dapp]-user2[dapp])
        elif dapp in user1.keys() and dapp not in user2.k
            diff_dic[dapp] = user1[dapp]
        elif dapp not in user1.keys() and dapp in user2.keys():
            diff_dic[dapp] = user2[dapp]
        else:
            pass
    print(diff_dic)
    return diff_dic
```

```
def user_distance(user1,user2):
    diff_dic = dapp_diff(user1,user2)
    total = 0
    for dapp in diff_dic.keys():
        total += diff_dic[dapp]
    return total
d = user_distance(user1,user2)
print(d)
```

Dapp use Prediction using Naive Bayes

```
KanaLabs given liquidSwap 0.7777777777777777
BlueMove given liquidSwap 0.6882129277566539
Hippo-Labs given liquidSwap 0.8082706766917293
Souffl3 given liquidSwap 0.8
Aries given liquidSwap 0.753623188405797
liquidSwap given Souffl3 0.4204131227217497
AnimeSwap given liquidSwap 0.7767857142857143
Topaz given Souffl3 0.989100817438692
Argo given liquidSwap 0.8306451612903226
HoustonSwap given liquidSwap 0.6923076923076923
Cetus given liquidSwap 0.8070175438596492
Aux given liquidSwap 0.8308457711442787
```

Bayes Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Average users uses 3 Dapps

```
def make_predictions():
  total=0
  total_valid =0
  for user in most_used_per_user.keys():
    top (variable) top_dapp: Any [user]
    if(top_dapp!="balance"):
      predicted_dapp = most_likely[top_dapp]
      if predicted_dapp[0] in test_users_dapps[user].keys():
        print("Correct prediction of :", predicted_dapp[0], "from use of
        # prediction is valid
        total_valid+=1
      total+=1
  print("Prob correct", total_valid/total)
```

Dapp prediction results

Given a user users a dapp we were able to predict another dapp they use with ~75% accuracy

```
Correct prediction of : liquidSwap from use of Souffl3
Correct prediction of : Souffl3 from use of Topaz
Correct prediction of : liquidSwap from use of Souffl3
Correct prediction of : liquidSwap from use of Hippo-Labs
Correct prediction of : liquidSwap from use of Souffl3
Correct prediction of : Souffl3 from use of Topaz
Correct prediction of : liquidSwap from use of BlueMove
Correct prediction of : liquidSwap from use of Aux
Correct prediction of : liquidSwap from use of Aux
Correct prediction of : Souffl3 from use of Topaz
Correct prediction of : liquidSwap from use of Souffl3
Correct prediction of : liquidSwap from use of Aux
Correct prediction of : Souffl3 from use of Topaz
Correct prediction of : liquidSwap from use of Souffl3
Correct prediction of : liquidSwap from use of Aux
Correct prediction of : liquidSwap from use of BlueMove
Correct prediction of : liquidSwap from use of Hippo-Labs
Correct prediction of : liquidSwap from use of BlueMove
Correct prediction of : liquidSwap from use of BlueMove
Correct prediction of : Souffl3 from use of liquidSwap
Correct prediction of : liquidSwap from use of Argo
Prob correct 0.75
```

Estimating User counts for each dapp

Liquid Swap users

Unique Users

94.6K

```
liquidSwap has 95600 users
KanaLabs has 1626 users
BlueMove has 10512 users
Hippo-Labs has 24974 users
Souffl3 has 40191 users
Aries has 3020 users
AnimeSwap has 10105 users
Topaz has 16378 users
Argo has 5982 users
HoustonSwap has 522 users
Cetus has 8015 users
Aux has 9699 users
5.402707863218198  % Error
```

Actual Hippo users **~30k**

Actual Animeswap users **~11k**

Possible sources of error and potential further exploration

How could this information be used?

Diversity in the training data

Issues: *Very convoluted and hard to read the code*

Very naive way of building this algorithm from scratch.

Time complexity - O(M*N^2)

N - Node/Attribute

M - Feature of Node/Feature of Attribute

Could definitely be improved to O(M*N*log(N))