Ensembles

Week	TUES. Week 8
Assignment Due	
Assignment Done	☐
Due Date	
Notes Done	☑

Homework Notes

You can expect the RMSE is proportional the scale of the features.

- Values on the order of 6k is normal

Class Notes

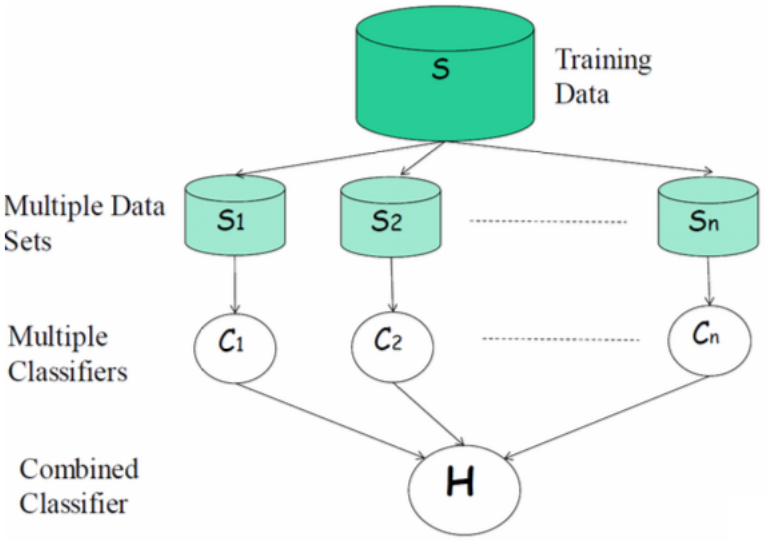
Ensembles

There is no algorithm that is guaranteed to get best results:

- We can look at our data and possibly come up with a plan as far as some set of algorithms to explore, train, validate, and see what algorithm does best.

The idea of **ensembles** asks “why do we only need one system?”

- Asks: “Why don’t we let our final decision be the contribution of multiple systems? Let this ensemble of systems collaboratively help us come up with a final decision!”
    - More often than not, it can improve performance!
    - Idea: Build different “experts” and let them collaborate to come up with a final decision



- Advantages:
  - Improves predictive performance
  - Different types of classifiers can be directly included
  - Relatively easy to do with training systems independently → with this, we need to think of a scheme to combine opinions from each system
  - Not too much parameter tuning (other than that of individual classifers themselves)
- Disadvantages:
  - Not as compact → if we're deploying an ensemble system, means we're deploying multiple subsystems
  - It may not be easy to interpret! → If the final decision made from collaboration of many subsystems, it is more difficult to interpret where final decision came from and why

Why do they work?

Things we can do to obtain different systems:

- We can use the *same type of system* with:
  - Different parameters

    - Different features
    - Different training sets
- We can use different classifier types with the *same data*.
- Unfortunately, there is no guarantee that an ensemble's decision will outperform that of a *single* one.
    - Empirically, there is no guarantee that this will be true.
    - To maximize gain (and the likelihood of some type of gain), we want to:
        - Ensure all of our systems are *somewhat* **accurate**. We certainly need to do better than random/prior.
        - Make sure that the classifers have **diverse errors** so they're as independent as possible.
            - If we can diversify our errors,, then this can aid ensemble performance (maximize our gain) since the things that the first system gets wrong, the second system will get right, effectively cancelling each other out.

Voting

Assume we'll have T total systems of different classifiers $c_1, c_2, \ldots, c_T$

- Now we must decide to train T different systems:
    - This could be an ensemble of different classifier built on different training sets and/or are different types of classifiers.
    - Now we want to use these classifiers to classify an unseen sample x
        - Each classifier c_t returns a probability that x belongs to a class $k = 1, ...K$ as $P_{tk}(x)$
            - Note: Some systems may not give a probability → may tell us whether something is in this class or not, giving either a 0 or 1 value
    - Now that we have Tsystems that each tell us the probability for each of the K classes, we need a strategy to combine these opinions.

There are many ways to combine these systems:

- We can undertake a **voting** scheme of combination:
  - Ex: Say we have 2 out of 3 classifers saying class 2 of 3 existing classes is most likely.

	Class 1	Class 2	Class 3
Classifier 1	0.2	0.5	0.3
Classifier 2	0.0	0.6	0.4
Classifier 3	0.4	0.4	0.2
Mean	0.2	**0.5**	0.3
Median	0.2	**0.5**	0.4
Min	0.0	**0.4**	0.2
Max	0.4	**0.6**	0.4
Product	0.0	**0.12**	0.032

    - We can combine these differing opinions by calculating the **mean** of each class' probability:
      - Mean given 3 classifiers: $P(y = 1) = 0.2$...etc.
    - We could also calculate the **median** → would be useful in the case that we want to safeguard against extreme values on either end of the probability spectrum
    - We could also use the **minimum** → would be helpful to safeguard against ever selecting a class when a system says there is no chance of that class being classified
    - We could also use the **maximum** → in this we're relying on most confident
    - We could also use the **product** of the probabilities.
    - We could also incorporate in our decisions is how accurate our system is in general with the **weighted mean**.

- Ex: if 2nd classifier had a validation accuracy of 34% in a 3-class training dataset, since it's barely doing better than random performance, maybe we shouldn't care as much about its opinion than another classifier that has way better accuracy.
- Opinions are *weighted* based on the classifiers' accuracies.
    - Note: We want to normalize the data with our weighted sums so that all α_t

- Mean: $P_k = \frac{1}{T}\sum_{t=1}^{T} P_{tk}$
- Weighted Mean: $P_k = \sum_{t=1}^{T} \alpha_t P_{tk}$ where $\sum_t \alpha_t = 1$
- Median : $P_k = \text{median}_t(P_{tk})$
- Minimum: $P_k = \min_t(P_{tk})$
- Maximum: $P_k = \max_t(P_{tk})$
- Product: $P_k = \prod_t P_{tk}$

Bagging

A related idea to combing up with different ensemble systems is **bagging**:

- We can create different ensemble systems by varying the training set to have different systems trained on different subsets of training data to increase generalizability and maximize diversity.
    - Important note: Of course, training is finite 😟
        - Ex: If we wanted to train 10 systems and divide the data into 10 subsets, we may end up with relatively small training sets for each individual system.
        - We know that small training sets are prone to overfitting and have a lack of generalizaiton.
    - Selecting samples of each datasets at random and with replacement is the idea of **bagging**.
        - You can think of the system as having a bag of observations in which observations randomly selected from the total training data *WITH* replacement → means any given system could get an observation *more*

than once (alternatively any observation may be included in several systems

- Helps with wanting slightly different training sets for our different systems with finite data.

Boosting

We further have the idea of **boosting**, *related* to **bagging**.

- Says: We know to maximize the gain from this ensemble technique, its good to diversify our errors (i.e.: what is wrong on one system should be correct on another)
- Hypothesis:
  - In training a system, we get a bag of observations and validate with the *entire* training dataset, seeing which samples the classifier gets correct and which are incorrect.
  - For the second system we'll train, we're going to create a second bag of observations.
    - In this second system, we want ***more*** of the samples of the observations we got incorrect in the first system.
      - If we include more samples of the observations we got incorrect, then whatever we learn in the new system is *focused* on these observations that we're "focusing" on here.

General Boosting Algorithm

General boosting algorithm works through the principle of wanting to boost probability of selecting things we got wrong and suppress probability of selecting things we got right:

- We are given:
  - Training set → $\{X, Y\}_{i=1}^{N}$
  - Initial sampling distribution → D_1 on $\{1, \ldots, N\}$
    - This is the probability of selecting each training sample to form a bag.
- For as many systems we want to build $t = 1, \ldots, T$, we create a bag S_t of size M by sampling from dataset $\{X, Y\}_{i=1}^{N}$ based on current sampling distribution D_t

- We then train our system c_t using the bag and we compute the **training** error.
- Then, we'll classify all of training data using the learned system.
- From this classification, we'll change the next sampling distribution D_{t+1} s.t. anything **right** will be ***less likely*** to be selected for next bag and anything **wrong** is ***more likely*** to be selected for the next class.
- After building the T systems, we have an ensemble and can combine the classifier to make a final decision, weighting their "opinion" by their training data.

AdaBoost

A version of **boosting** that has showed success and formalized some parameters was **Adaboost** (for ***adaptive boosting***)

For training:

- In this boosting algorithm, all it did was come up with some design decisions.
  1. **Generates initial, uniform sampling distribution** → the sampling distribution (probability of selecting any given sample) was the same
     - $D_1 = \frac{1}{N}$ → all samples treated equally at first
  - For as many systems as we want to build, for $t = 1$ to T:
    1. Train classifier c_t drawn according to distribution D_t and obtain training error ϵ_t
       - If $\epsilon_t > \frac{1}{2}$, stop and discard poor classifier
         - In this case, we wouldn't train all the way out to our T number of classifers because if we get to a point that we're training poorly performing systems then there's no point in continuing to that pathway.
         - ϵ_t is $1 - accuracy$
       - Otherwise:
         - Compute $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

    - Go through all samples X_i in the full training set and classify each training set point as follows:
        - If $c_t(X_i) == Y_i$, then $D_{t+1}^i = \beta_t D_t^i$ (*reduce probabilities* of things positively classified)
            - This sample's probability of being selected in the next bag is now decreased/suppressed by β_t value, which is between 0 and 1.
        - Else, $D_{t+1}^i = D_t^i$ → leave this sample probability alone
- Normalize the probabilities so they'll sum to 1.
    - Add up new probabilities and divide each of them by the sum term, which gives us a valid probability distribution.
    - $Z_{t+1} = \Sigma_{i=1}^N D_{t+1}^i$
    - $D_{t+1}^i = \frac{D_{t+1}^i}{Z_{t+1}}$
- Ex: Normalizing probabilities
    - $D_1 = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$
    - $\epsilon_1 = \frac{1}{4}$
    - $B_1 = \frac{\frac{1}{4}}{1-\frac{1}{4}} = \frac{\frac{1}{4}}{\frac{3}{4}} = \frac{1}{3}$
    - Say only the second sample in the training data was classified incorrectly:
        - $D_2 = [\frac{1}{9}, \frac{1}{3}, \frac{1}{9}]$
        - To normalize, we add these probabilities and divide each by that value. → $\Sigma D_2 = \frac{5}{9}$
        - Normalized distribution: $D_2 = [\frac{1}{5}, \frac{3}{5}, \frac{1}{5}]$
            - We can see here that the sample that we classified incorrectly here now has a higher probability of being chosen in the future iterations.
- Note: In performing this algorithm, we do not have to run it twice, but instead in obtaining the training error, we can obtain our $\hat{y}$ values and perform the calculations

under “otherwise” accordingly to update the next sample distribution.

For testing:

Once we have T systems created, we approach classification pretty much same as a regular ensemble.

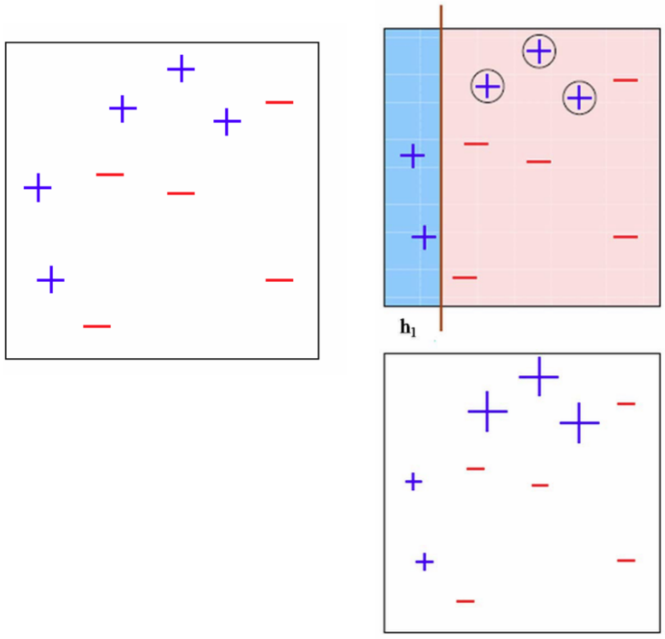
- Adaboost basically does a weighted average, where the weights used are slightly different than usual, such as:

$$P_k(\boldsymbol{x}) = \sum_{t=1}^{T} \log\left(\frac{1}{\beta_t}\right) P_{tk}(\boldsymbol{x})$$

  - Individual weights here is $\log(\frac{1}{\beta_t})$
- We choose the label as $\hat{y} = argmax_k P_k(\mathbf{x})$

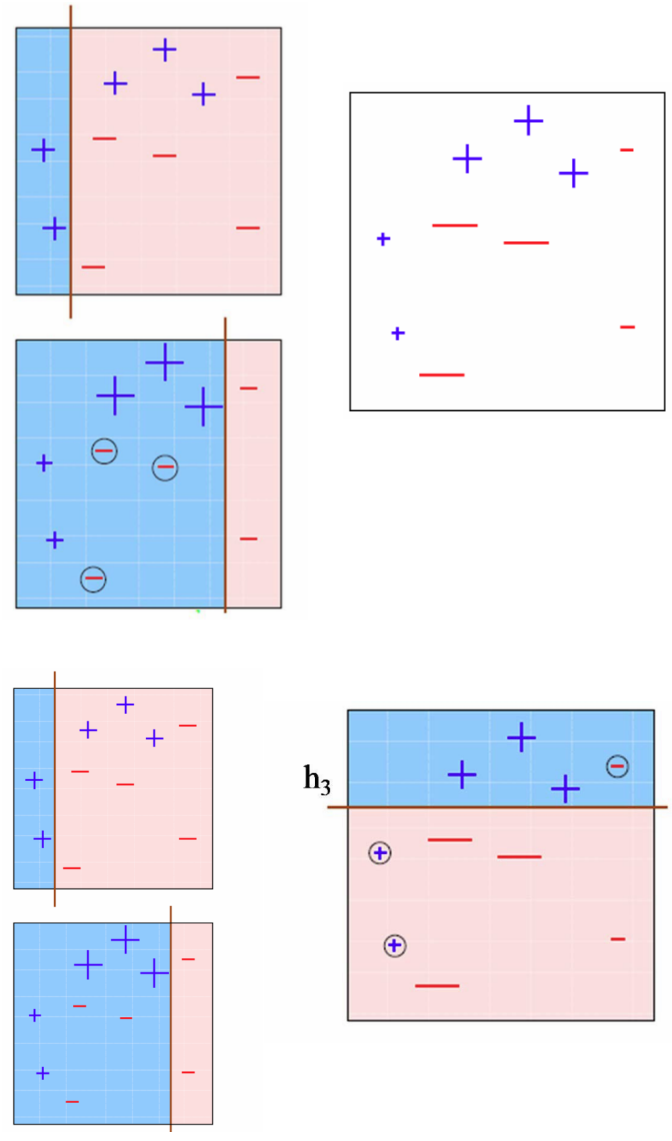
Toy Example

Size is representative of each sample’s probability of being selected.

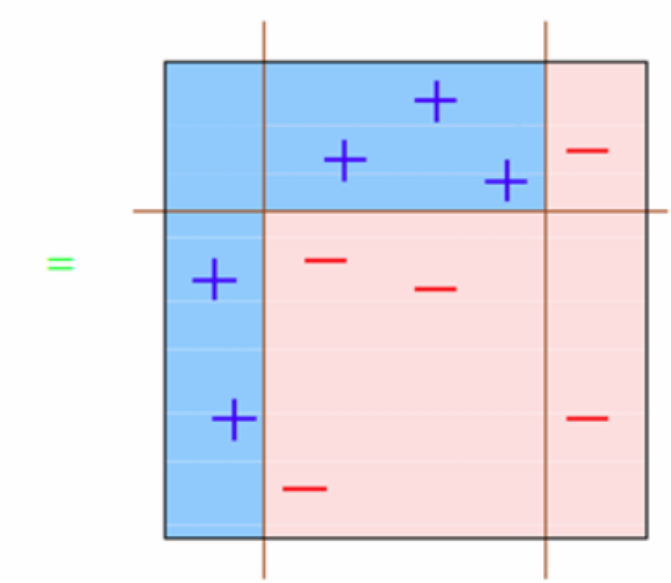


- Among training samples, we see above that those three circles are wrong.
    - Adaboost will decrease probability of everything we got being correct, it's all visualized.
    - Simultaneously, it'll increase the probability of everything we got wrong.

Using this distribution above, we can build a second bag of samples with a different sampling distribution based on what we got correct/incorrect.



Conceptually, we can think of the ensemble as the combination of the 3 rules above in a voting scheme, which can manage to give us this classification scheme (again, mainly for visualization).



Advantages

- **Simple and easy to implement** → all we're doing is changing probability of selecting training samples in our next bag
- **Flexible as it is Independent of any learning algorithms**
- **No parameters to tune** → Not many decisions to make, does not depend on individual classifiers doing well
- **Versatile** → can work on a lot of problems

Disadvantages

- Performance depends on individual data and weak learner
- Can fail if weak classifier is too complex (overfitting) or too weak (underfitting)

Random Forests

Ensemble learning in general is “wisdom of the crowd”

- The idea is that since there is NO *single system* guaranteed to do the best, we can build a bunch of systems where we consider the different ways in which we can do it, such as:
  - Different types of algorithms
  - Different training sets on the same algorithm
  - Different training sets, selecting samples *with replacement*
- Another type of classifer that is technically an ensemble classifer is referred to as a **random forest classifier**.
  - We could technically create a ***traditional ensemble*** using decision trees as our classifers.
    - Create T different classifers, possibly using bagging/boosting, so each trees have different datasets and learn a different tree structure.
      - Here, trees are our base classifer.
        - For a new input $\mathbf{x}$, get classification from each of the T trees and make a final decision (weighted sum, majority voting, etc.)

Random forests operate a bit differently in having **feature bagging**:

- **Feature bagging** adds additional randomness s.t. when we’re building our tree at any given location that we’re currently on (wherever on it), we look at the features currently available to us.
  - If we’re doing an ID3 algorithm, we compute the weighted average entropy of each feature that we could potentially use there.
- For **random forests**, to add the randomness, it says “when it comes time to select which feature to split on, don’t allow someone to use all features available to them, but *INSTEAD* a randomly selected subset to explore.
  - As a result, the best option there may no longer be available, so we’re forced to choose the 2nd best option.

Random Forest Algorithm

- Repeat the following T times:
  - Draw a training set S_t of size M from the original training data with replacement (bagging-type approach to creating training set).
  - Grow a random-forest tree c_t using dataset S_t by recursively repeating the following steps for each terminal node of the tree until there are no more features to pull from (build a full tree as we normally would).
    - Difference here from above!! Each time, when hitting a node and needing to decide on what features to split on, grab n features at random out of p remaining features available at that time
      - Empirical results suggest trying $m = \frac{1}{2}\sqrt{p}, \sqrt{p}$, and $2\sqrt{p}$.
    - Pick the best variable/split-point among the m, possibly using entropy.
    - Split the node into two daughter nodes.
  - Output the ensemble of trees $\{c_t\}^T$
- To make a prediction at a new point $\mathbf{x}$, combine the trees to make a final decision by either combining the votes in a traditional ensemble manner.

Example

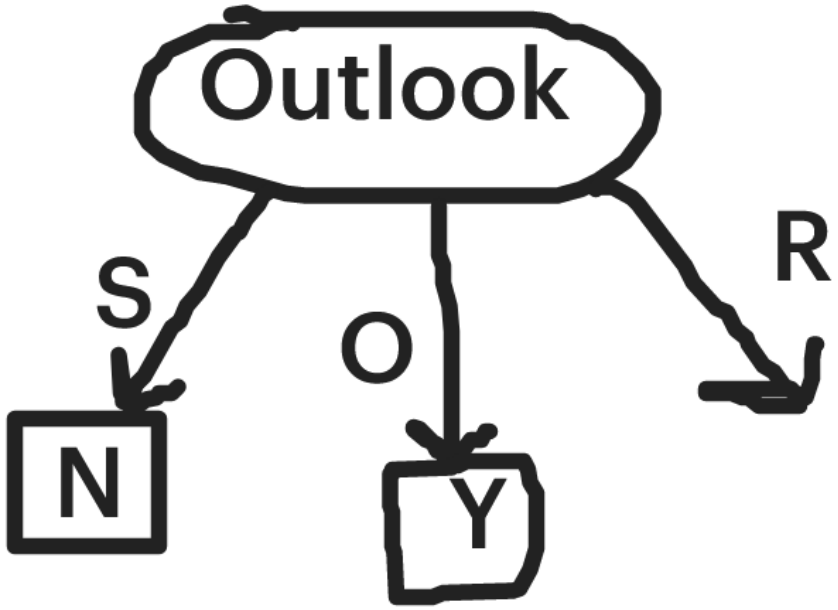
Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Say we have a dataset that is binary with respect to its binary.

- Let's decide to select $M = 7$ samples at random with replacement (for each tree we'll build) and each time, we need to decide on which features to look at for splitting.
    - We'll look at $m = ceil(\sqrt{p})$
    - Below is our "randomly" selected training set for our first tree:

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes

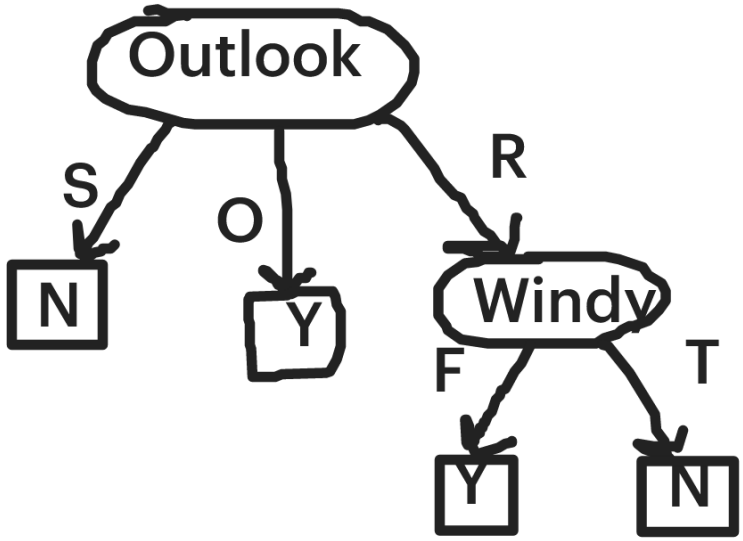
- At the root of the tree, we have $p = 4$ potential features to look at → **outlook**, **temperature**, **humidity**, and **windy**
  - Since $m = ceil(\sqrt{p}) = 2$, we'll choose two features at random.
    - Let us assume we chose **outlook** and **windy**.
      - With the ID3 algorithm, we'd calculate the weighted average entropy for the subsets and choose the best.
    - **Outlook** ends up being the best feature here to branch on.



      - The sunny and overcast branches have consensus so we assign those to their respective target values.
      - However, rainy has 3 samples left which do not have consensus.

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes

- At this point, we now have 3 features that we can branch on → **temperature**, **humidity**, and **windy**, $p = 3$
  - $m = ceil(\sqrt{p}) = 2$, so we choose 2 features at random here, which we'll say is **temperature** and **windy** at random
    - **Windy** ends up being the better feature to split on, so we choose that accordingly.



- Now we have our **random forest tree**!
  - We can repeat this process as many times as we'd like, having a different training set.
    - each time, we're selecting an internal node, we're selecting a subset at random to explore so the random forest trees generated should be slightly

different.

- Admittedly, the trees generated may be built on a small training set. However, due to the randomness in selecting features, the trees will not be prone to overfitting (rather may be prone to underfitting).
    - By having diverse sets of trees, the “wisdom of the crowd” part of this ensemble should be helpful to making final decisions.
- These decision trees have been shown do a lot in several problems.

To combine these trees, you would take your new observation and put them down each tree:

- For this training data, you’d get a yes or no from each tree.
    - Could decide by the mode (which target was predicted the most).
    - Could also calculate accuracy of single tree → if it says $\hat{y} = 0$, then the $P(\hat{y} = 0)$ could be the accuracy while the $P(\hat{y} = 1)$ could be 1 - accuracy