

Data Science

Lecture 3: Structured Data Processing
(Decision Tree and Random Forest)



UNIVERSITY
OF AMSTERDAM

Lecturer: Yen-Chia Hsu

Date: Feb 2025

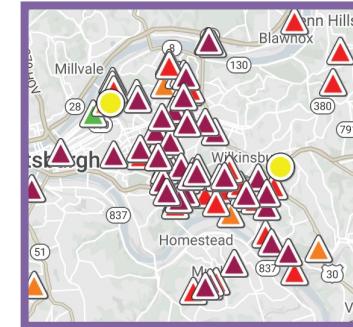
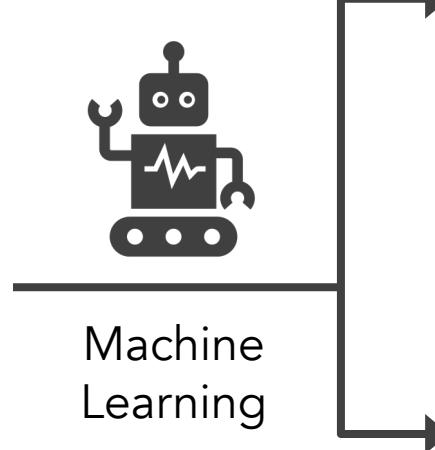
This lecture will introduce the Decision Tree
and Random Forest models.

Recall that in the structured data processing module, we used air quality sensor data in the past to predict the presence of bad odors in the future.

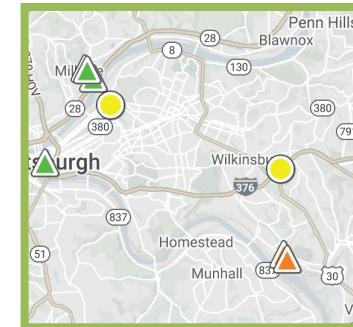
| | |
|-------------------------|---|
| O ₃ : 26 ppb | CO: 127 ppb |
| H ₂ S: 0 ppb | PM _{2.5} : 9 µg/m ³ |
| Wind: 17 deg | ... |
| Observation 1 | |

| | |
|-------------------------|--|
| O ₃ : 1 ppb | CO: 1038 ppb |
| H ₂ S: 9 ppb | PM _{2.5} : 23 µg/m ³ |
| Wind: 213 deg | ... |
| Observation 2 | |

⋮



:(Has Event



:> No Event

Structured data generally means the type of data that has **standardized formats** and **well-defined structures** (i.e., those that can be stored in a relational database).

- Samples of Citizen-Contributed Smell Reports

| EpochTime | feelings_symptoms | smell_description | smell_value | zipcode |
|------------|---|---------------------------|-------------|---------|
| ... | ... | ... | ... | ... |
| 1478353854 | Headache, sinus, seeping into house even though it is as shut and sealed as possible. Air purifiers are unable to handle it thoroughly. | Industrial, acrid, strong | 4 | 15206 |
| 1478354971 | ... | Industrial | 4 | 15218 |
| ... | ... | ... | ... | ... |

These are technically not structured.

- Samples of Air Quality Sensor Measurements

| EpochTime | 3.feed_28.H2S_PPM | 3.feed_28.SO2_PPM | 3.feed_28.SIGTHETA_DEG | 3.feed_28.SONICWD_DEG | 3.feed_28.SONICWS MPH |
|------------|-------------------|-------------------|------------------------|-----------------------|-----------------------|
| ... | ... | ... | ... | ... | ... |
| 1478046600 | 0,019 | 0,020 | 14,0 | 215,0 | 3,2 |
| 1478050200 | 0,130 | 0,033 | 13,4 | 199,0 | 3,4 |
| ... | ... | ... | ... | ... | ... |

Mathematically speaking, we want to estimate a function f that can map **feature X** to **label y** such that the prediction $f(X)$ is close to y as much as possible.

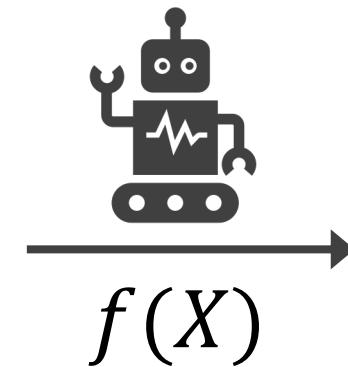
| O ₃ | CO | ... | ... | Wind |
|----------------|---------|-----|-----|--------|
| 26 ppb | 127 ppb | ... | ... | 17 deg |

| |
|----------------|
| O ₃ |
| 26 ppb |
| 10 ppb |
| : |
| 21 ppb |

observation: $x^{(1)}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & \dots & x_p^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & \dots & x_p^{(2)} \\ \vdots & \vdots & \vdots & \dots & \dots & \vdots \\ \vdots & \vdots & \vdots & \dots & \dots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \dots & \dots & x_p^{(n)} \end{bmatrix}$$

feature: x_1



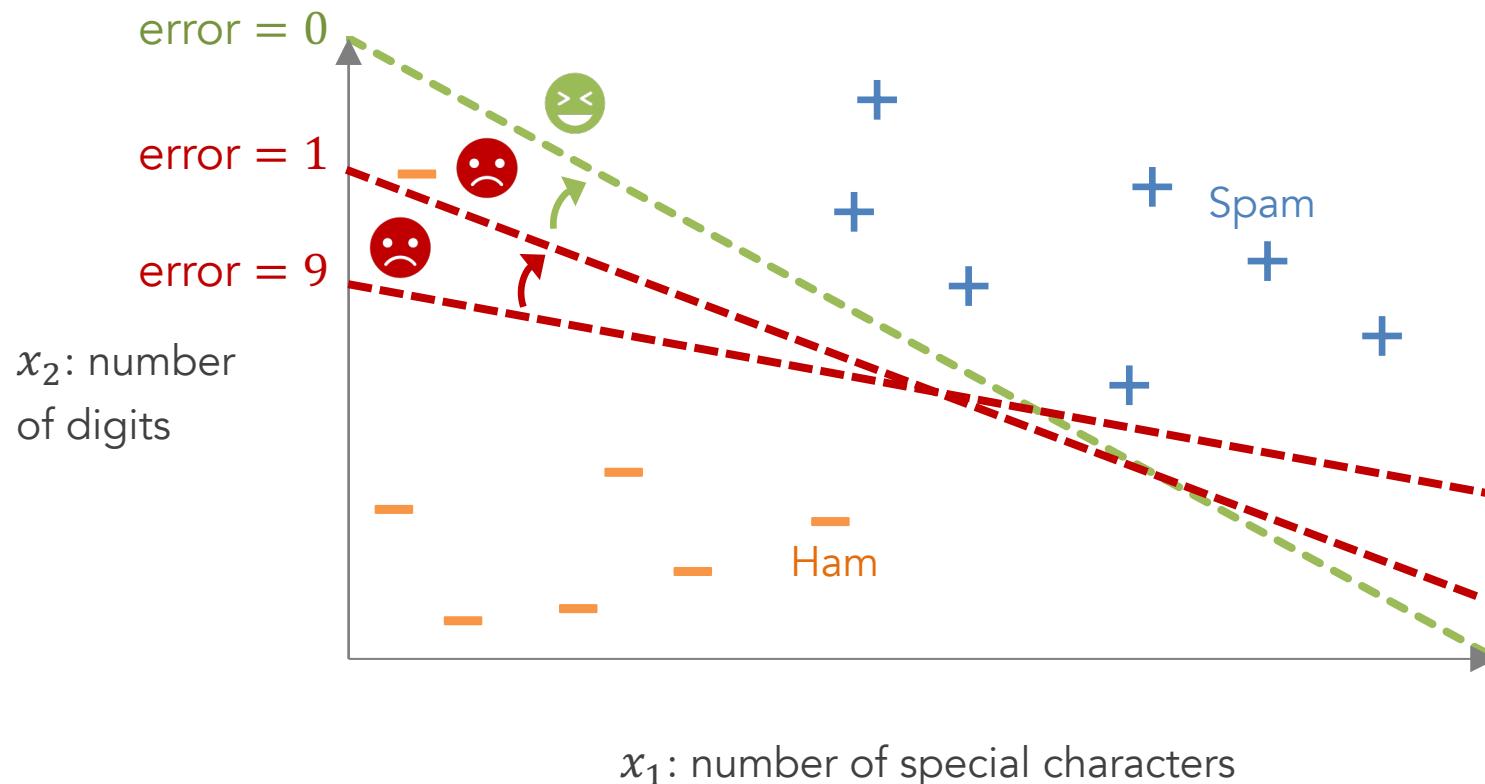
label: y

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} = y$$

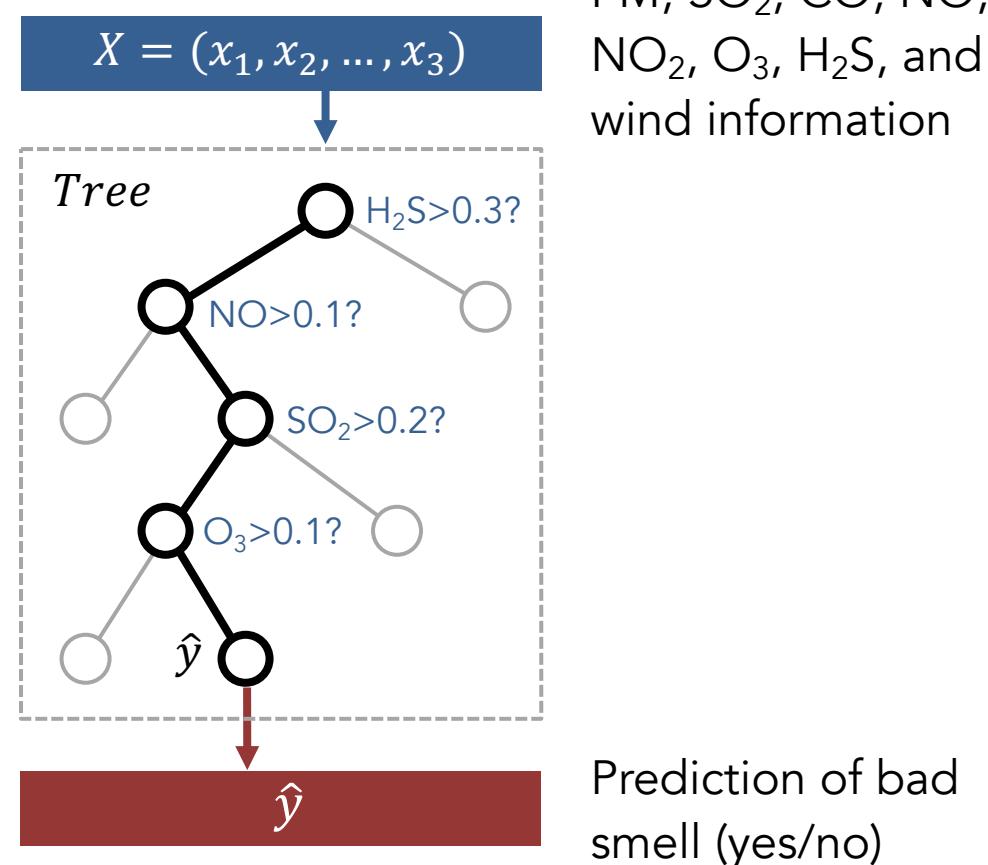
| |
|------------|
| Has event? |
| no |
| yes |
| : |
| no |

We have learned the concept of training a [linear classifier](#) using a [general optimization algorithm](#) to minimize an [error metric](#) to classify spam and ham (non-spam) messages.

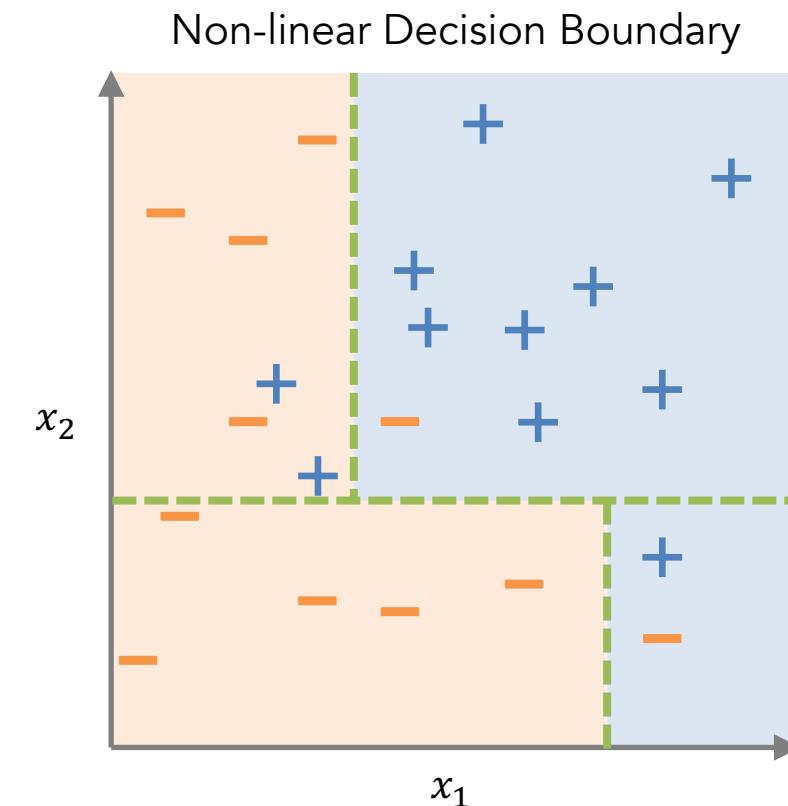
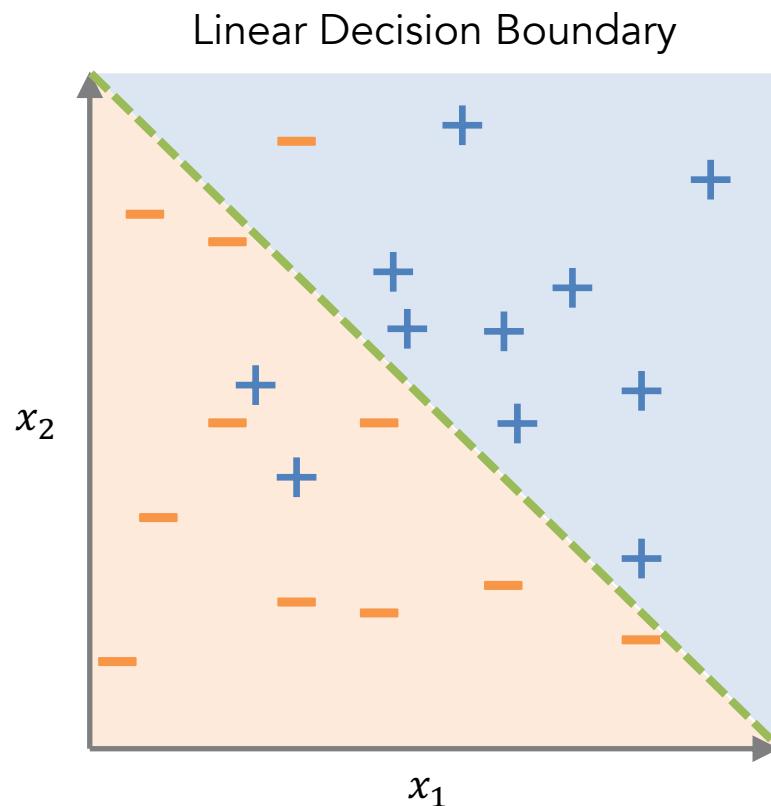
$$\text{minimize error} = \sum -y \cdot f(X) \text{ for each misclassified point } X$$



Not every model is trained using a general optimization algorithm. In this lecture, we will introduce **Decision Tree**, which has a different training mechanism.



Unlike the linear classifier (which has a linear decision boundary), Decision Tree has a **non-linear decision boundary** that iteratively partitions the feature space.



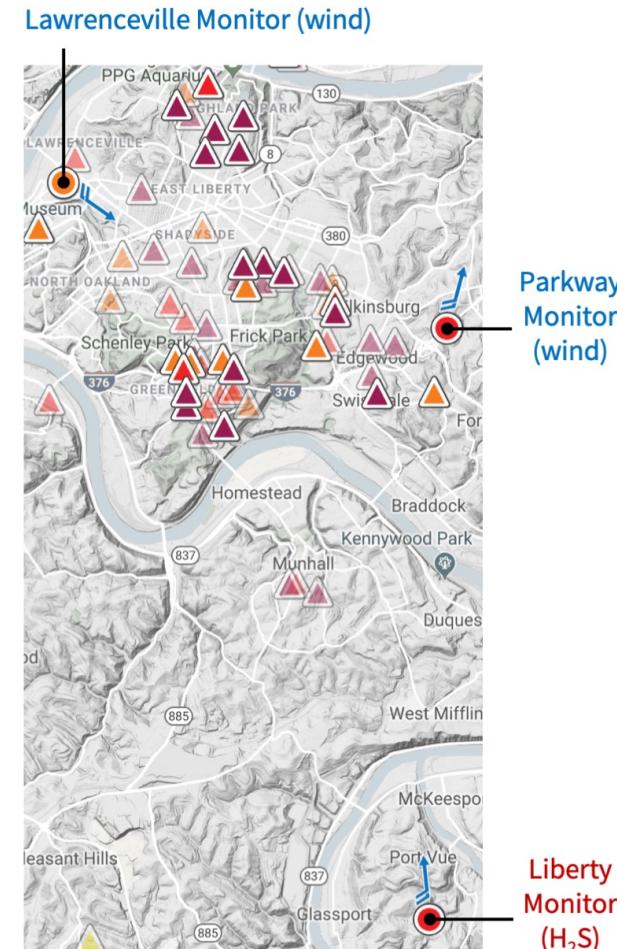
For simplicity, assume that all features are binary. We can use domain knowledge to determine if H₂S can be detected by human nose (e.g., larger than 0.03 ppm).

| EpochTime | 3.feed_28.H2S_PPM | 3.feed_28.SO2_PPM | 3.feed_28.SIGTHETA_DEG | 3.feed_28.SONICWD_DEG | 3.feed_28.SONICWS MPH |
|------------|-------------------|-------------------|------------------------|-----------------------|-----------------------|
| ... | ... | ... | ... | ... | ... |
| 1478046600 | 0,019 | 0,020 | 14,0 | 215,0 | 3,2 |
| 1478050200 | 0,130 | 0,033 | 13,4 | 199,0 | 3,4 |
| ... | ... | ... | ... | ... | ... |

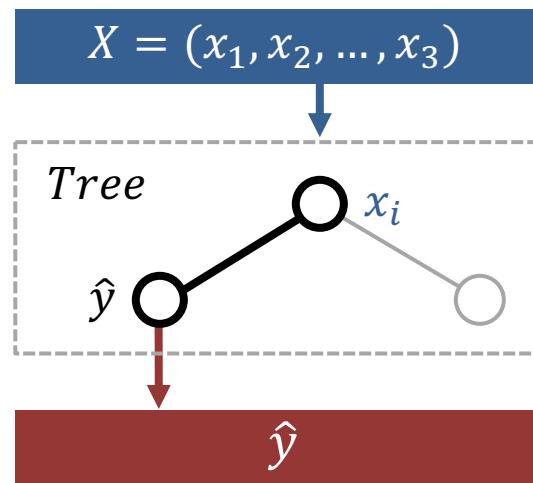
| Is H ₂ S noticeable (sensor 28)? | Is SO ₂ noticeable (sensor 28)? | Wind is turbulent (sensor 28)? | Is south wind (sensor 28)? | Is east wind (sensor 28)? | Is wind calm (sensor 28)? | ... |
|---|--|--------------------------------|----------------------------|---------------------------|---------------------------|-----|
| ... | ... | ... | ... | ... | ... | ... |
| No | No | No | Yes | No | Yes | ... |
| Yes | No | No | Yes | No | Yes | ... |
| ... | ... | ... | ... | ... | ... | ... |

In order to guess whether the smell will be bad in the future ("yes" or "no"), suppose we are only allowed to **ask binary questions** in a sequence.

- Q: Is the hourly averaged H₂S reading (at the Liberty monitor) noticeable two hours ago by human nose?
- A: YES
- Q: Is the hourly averaged wind direction from south (at the Parkway monitor) one hour ago?
- A: YES
- Q: Is the hourly averaged wind direction from east (at the Lawrenceville monitor) one hour ago?
- A: NO
- We predict that bad smell will happen within 8 hours.



If we could **only ask one question**, which question would we ask? We want to use the most useful feature that can **give us the most information** to help us guess.



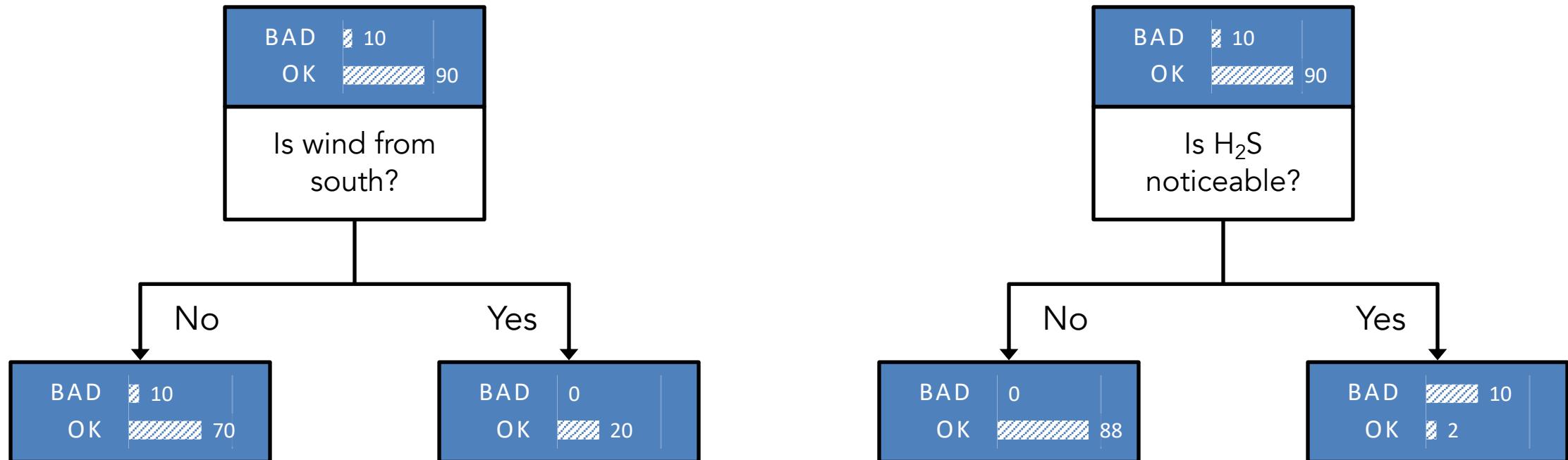
PM, SO₂, CO, NO,
NO₂, O₃, H₂S, and
wind information

Prediction of bad
smell (yes/no)

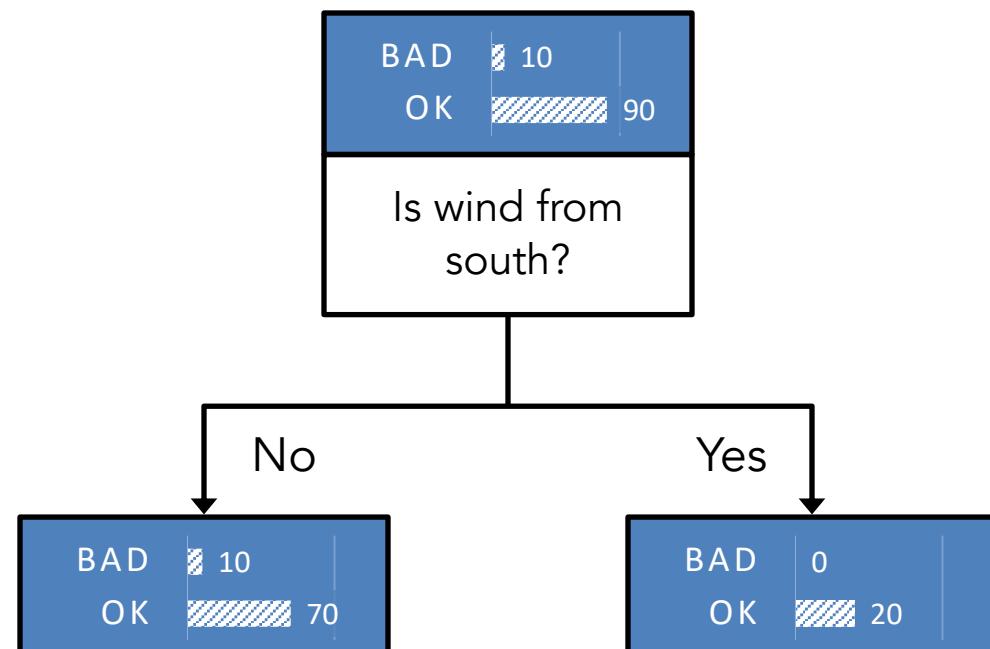
- Is PM noticeable?
- Is SO₂ noticeable?
- Is wind strong?
- Is wind turbulent?
- Is south wind?
- Is morning time?
- ...

Suppose we want to compare two features when predicting bad smell, as shown below.

Which one is better? How can we **quantify** which feature gives the most information?

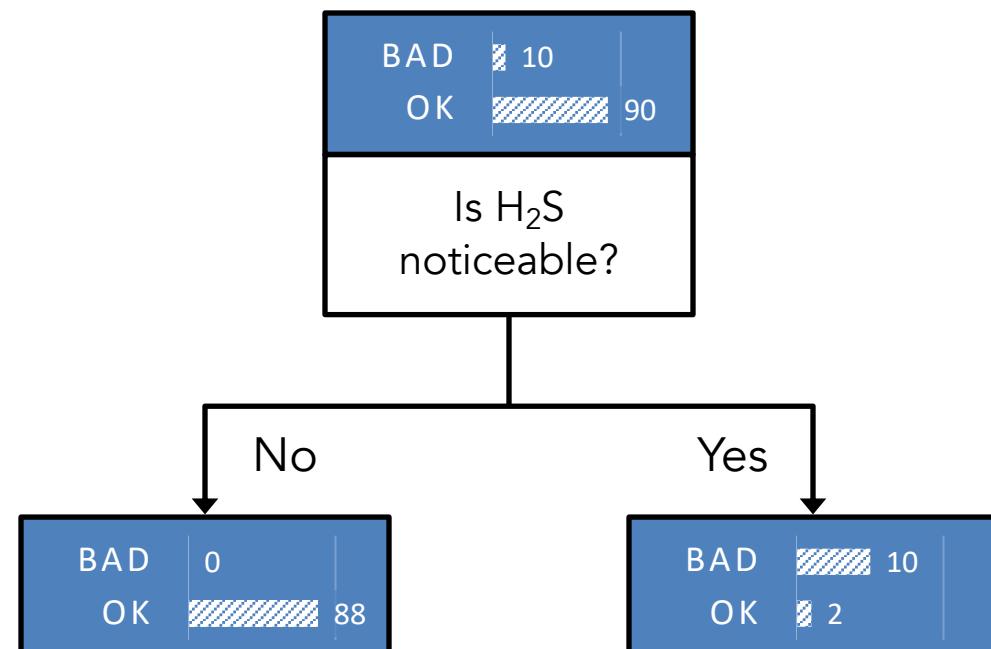


Exercise 4.1: For each split, compute the misclassification error rates (between 0 and 1) for the best guess (BAD or OK) that we can make *after* guessing the answer.



This node made
█ mistakes when
guessing █

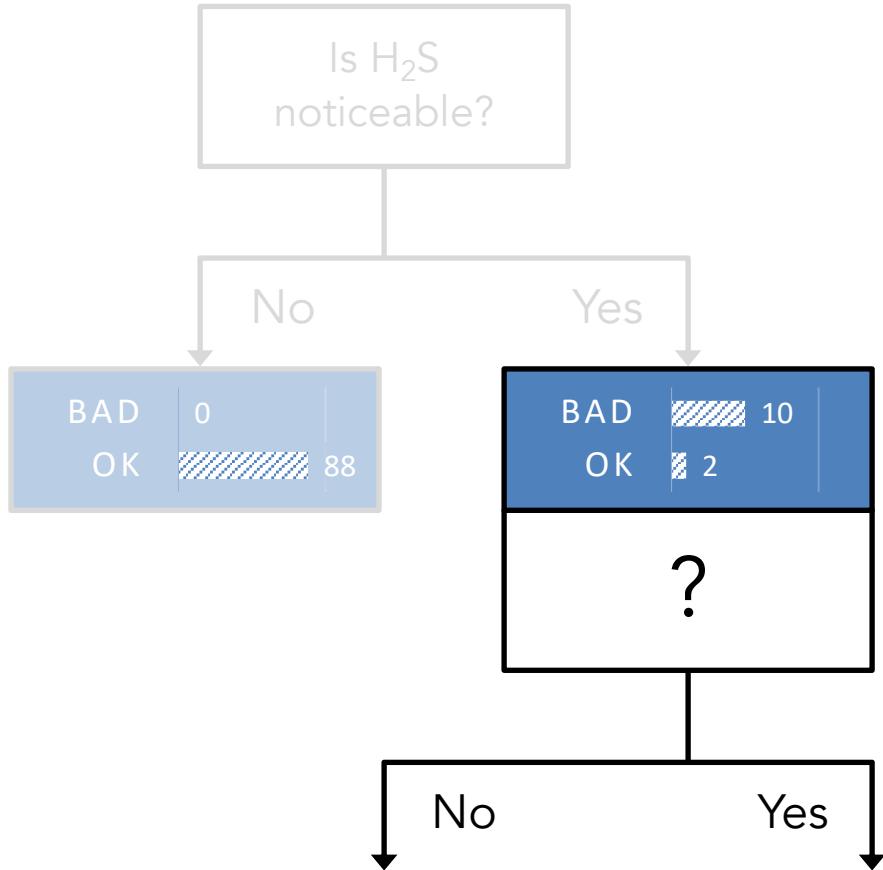
Total error rate for this tree split = ?



This node made
█ mistakes when
guessing █

Total error rate for this tree split = ?

We can use the same strategy to iteratively select the best feature for each tree node.



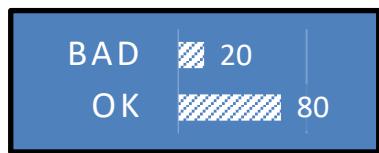
Algorithm 1 DECISIONTREETRAIN(*data*, *remaining features*)

```
1: guess  $\leftarrow$  most frequent answer in data // default answer for this data
2: if the labels in data are unambiguous then
3:   return LEAF(guess) // base case: no need to split further
4: else if remaining features is empty then
5:   return LEAF(guess) // base case: cannot split further
6: else // we need to query more features
7:   for all f  $\in$  remaining features do
8:     NO  $\leftarrow$  the subset of data on which f=no
9:     YES  $\leftarrow$  the subset of data on which f=yes
10:    score[f]  $\leftarrow$  # of majority vote answers in NO
11:      + # of majority vote answers in YES
12:      // the accuracy we would get if we only queried on f
13:   end for
14:   f  $\leftarrow$  the feature with maximal score(f)
15:   NO  $\leftarrow$  the subset of data on which f=no
16:   YES  $\leftarrow$  the subset of data on which f=yes
17:   left  $\leftarrow$  DECISIONTREETRAIN(NO, remaining features \ {f})
18:   right  $\leftarrow$  DECISIONTREETRAIN(YES, remaining features \ {f})
19:   return NODE(f, left, right)
end if
```

There are also other methods to quantify information, such as entropy.

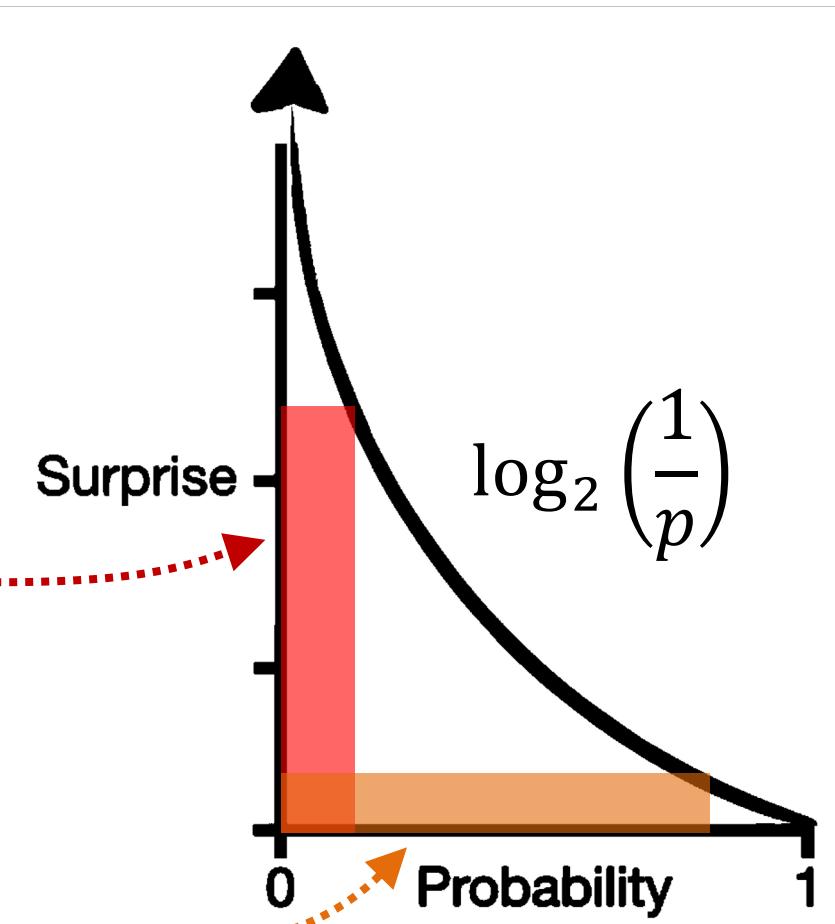
Suppose we have a coin: one side has label "BAD", and another side has label "OK".

The entropy H intuitively means the **averaged surprise** when we flip this coin.

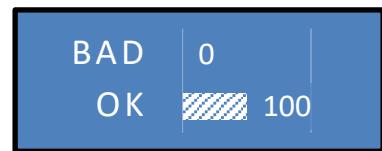


$$H = \sum \text{Probability} \cdot \text{Surprise}$$

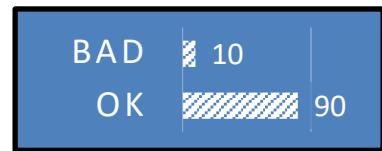
$$= p_{\text{BAD}} \cdot \log_2 \left(\frac{1}{p_{\text{BAD}}} \right) + p_{\text{OK}} \cdot \log_2 \left(\frac{1}{p_{\text{OK}}} \right)$$



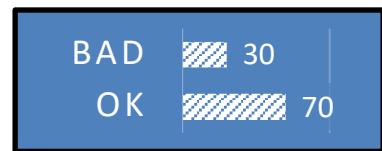
We want a small entropy. Entropy is zero when the coin always gives one side. Entropy reaches the maximum when the coin is fair, meaning two sides have equal probability.



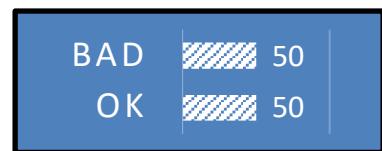
$$H = 0 + 1 \cdot \log_2 \left(\frac{1}{1} \right) = 0$$



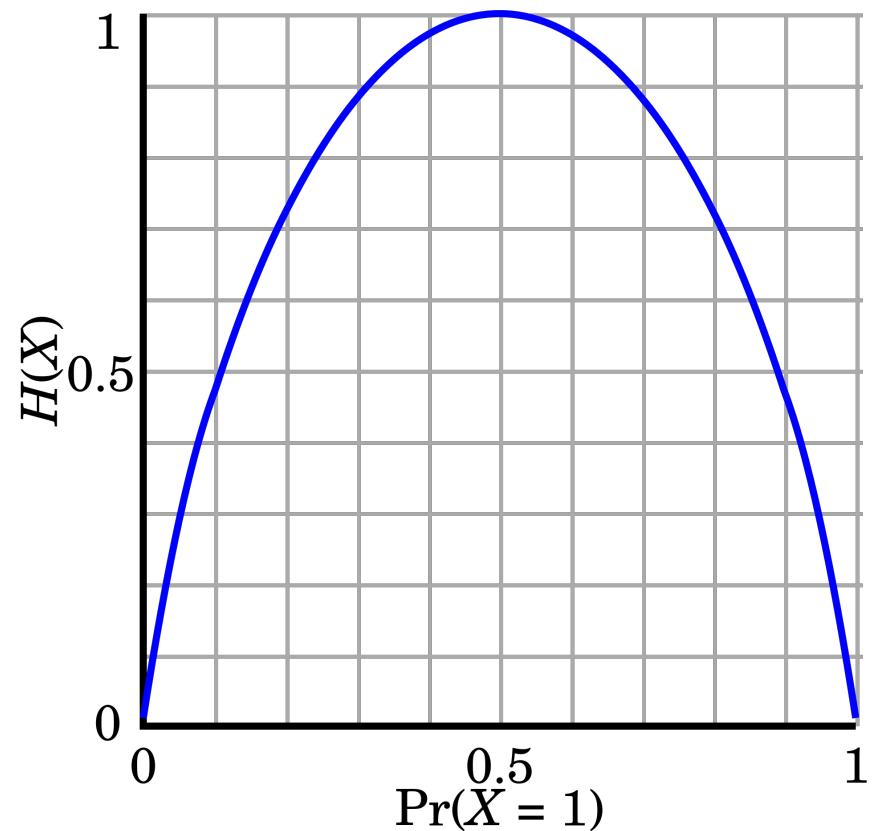
$$H = 0.1 \cdot \log_2 \left(\frac{1}{0.1} \right) + 0.9 \cdot \log_2 \left(\frac{1}{0.9} \right) = 0.47$$



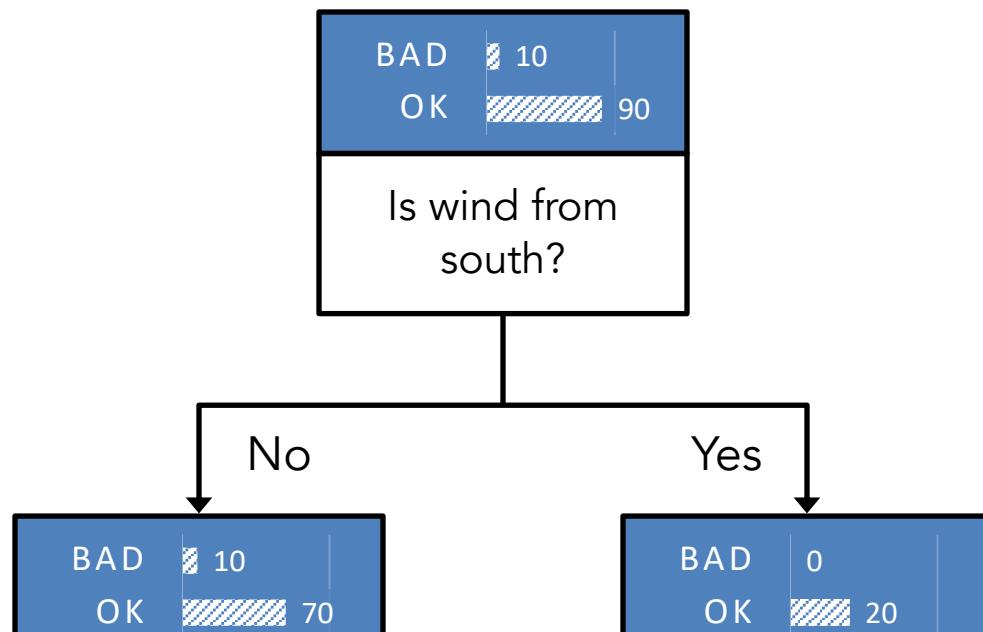
$$H = 0.3 \cdot \log_2 \left(\frac{1}{0.3} \right) + 0.7 \cdot \log_2 \left(\frac{1}{0.7} \right) = 0.88$$



$$H = 0.5 \cdot \log_2 \left(\frac{1}{0.5} \right) + 0.5 \cdot \log_2 \left(\frac{1}{0.5} \right) = 1$$

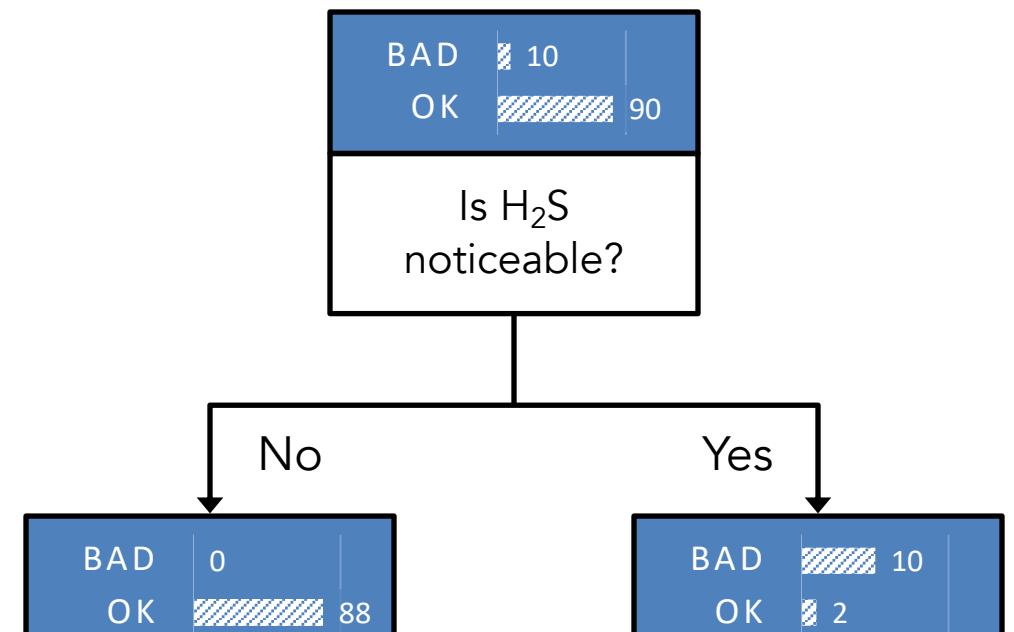


When splitting the parent node, we can use the **averaged entropy** of the leaf nodes to measure and quantify the information that each feature gives.



Entropy = 0.54

$$\text{Averaged entropy} = \frac{80}{100} * 0.54 + 0 = 0.43$$



Entropy = 0

$$\text{Averaged entropy} = 0 + \frac{12}{100} * 0.65 = 0.08$$

We can also use [information gain](#) to measure the reduction in uncertainty after the split.

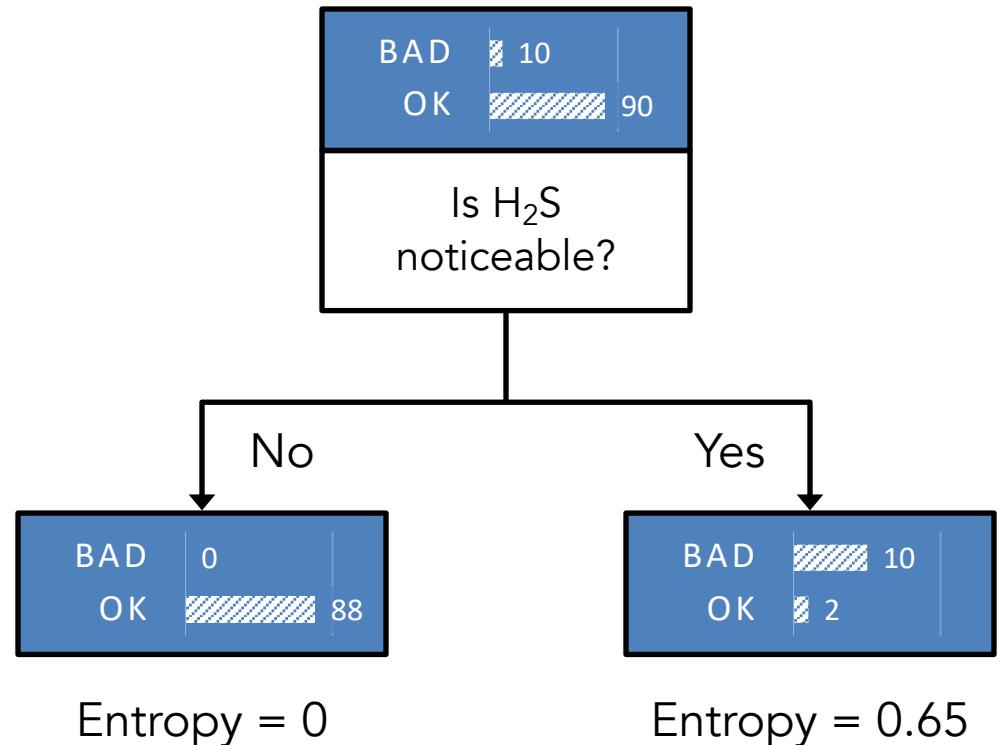
$$H_{parent} = \text{entropy of the parent node} = 0.47$$

Information Gain

$$= H_{parent} - H_{leaf}$$

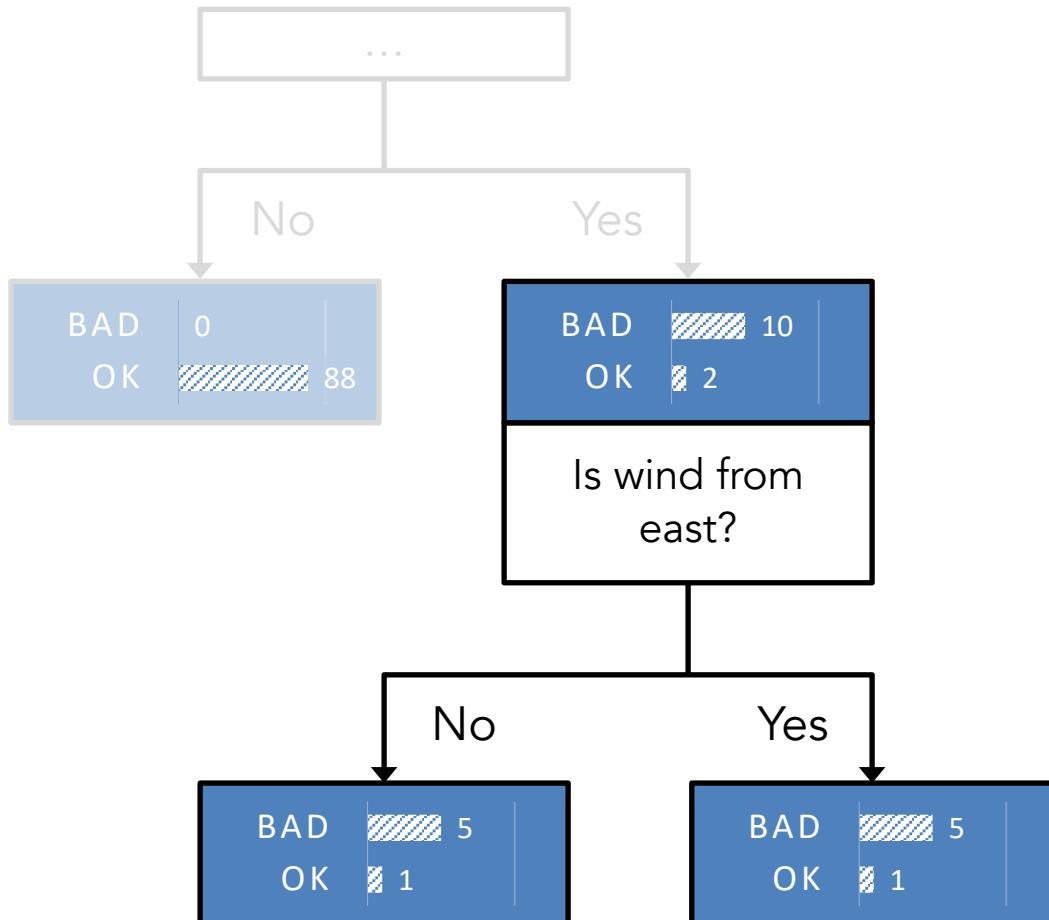
$$= 0.47 - 0.08$$

$$= 0.39$$



$$H_{leaf} = \text{averaged entropy of leaves} = 0.08$$

We can stop splitting when [the information gain is too small](#) for the best feature, which means splitting the node does not give a reasonable reduction of error (or uncertainty).



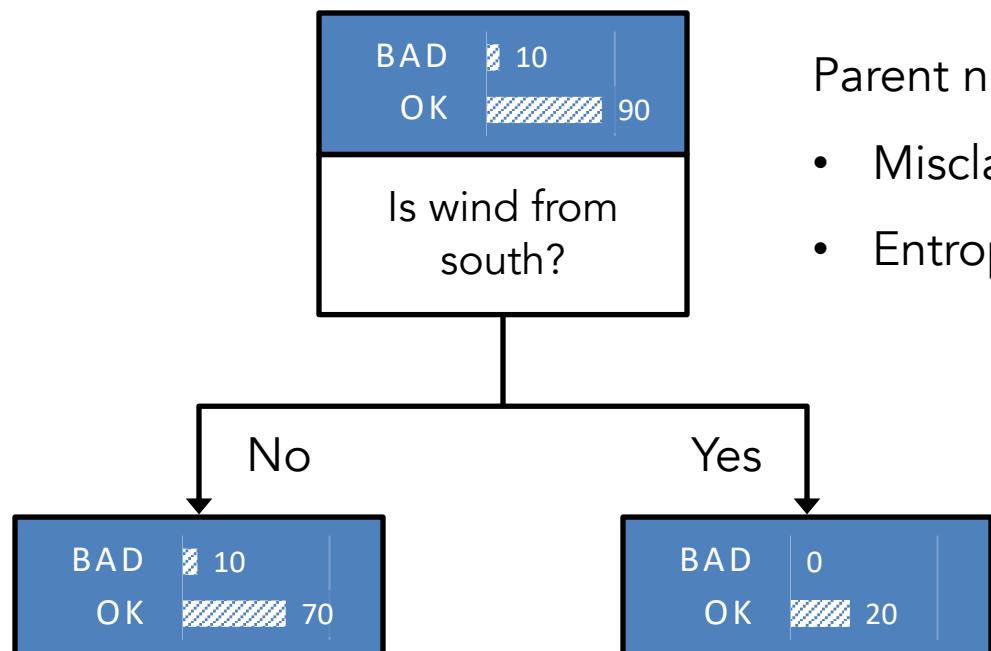
Parent node:

- Misclassification error = 0.17
- Entropy = 0.65

Leaf nodes:

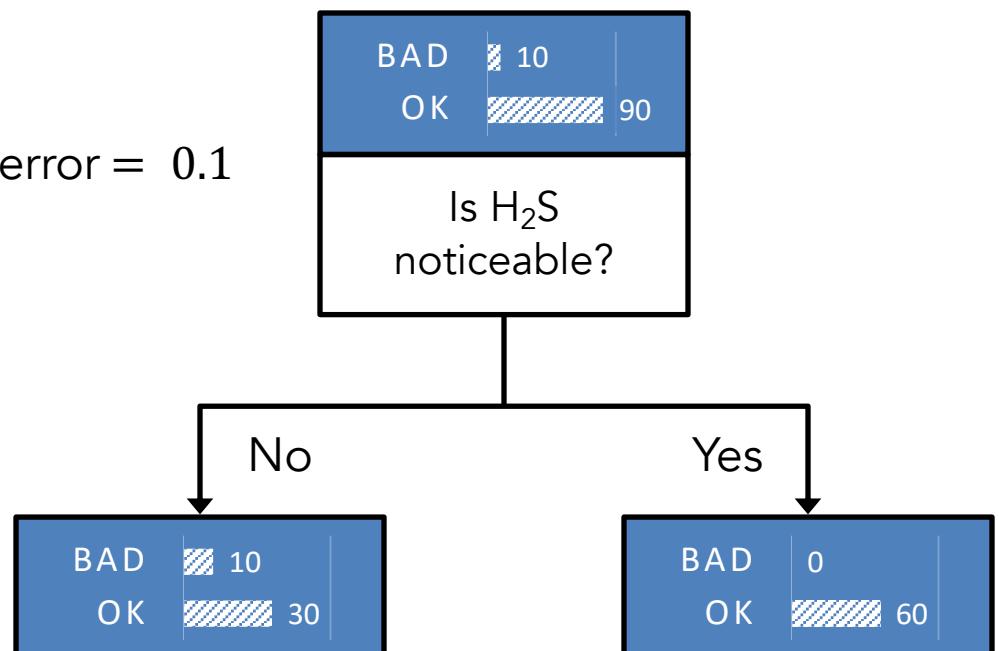
- Total misclassification error = 0.17
- Averaged entropy = 0.65

Exercise 4.2: Why is using the misclassification error as the node-splitting strategy not a good idea when training a decision tree?



Parent node:

- Misclassification error = 0.1
- Entropy = 0.47



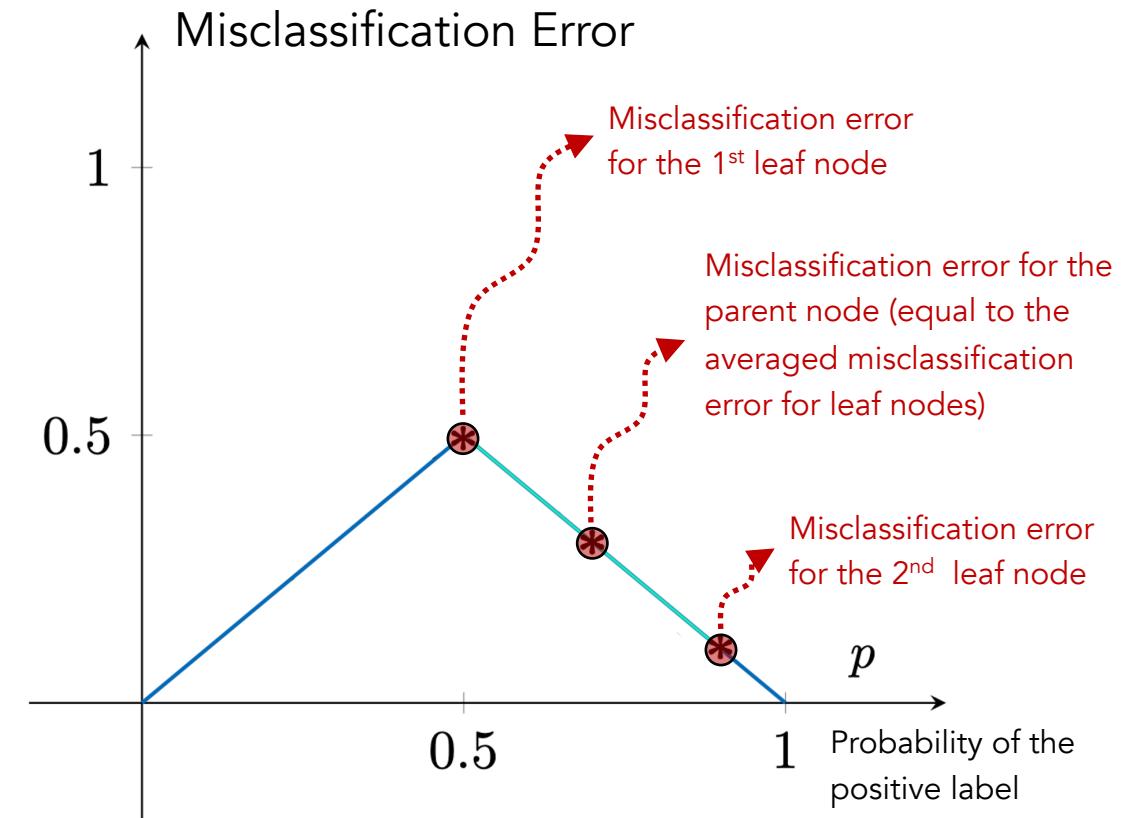
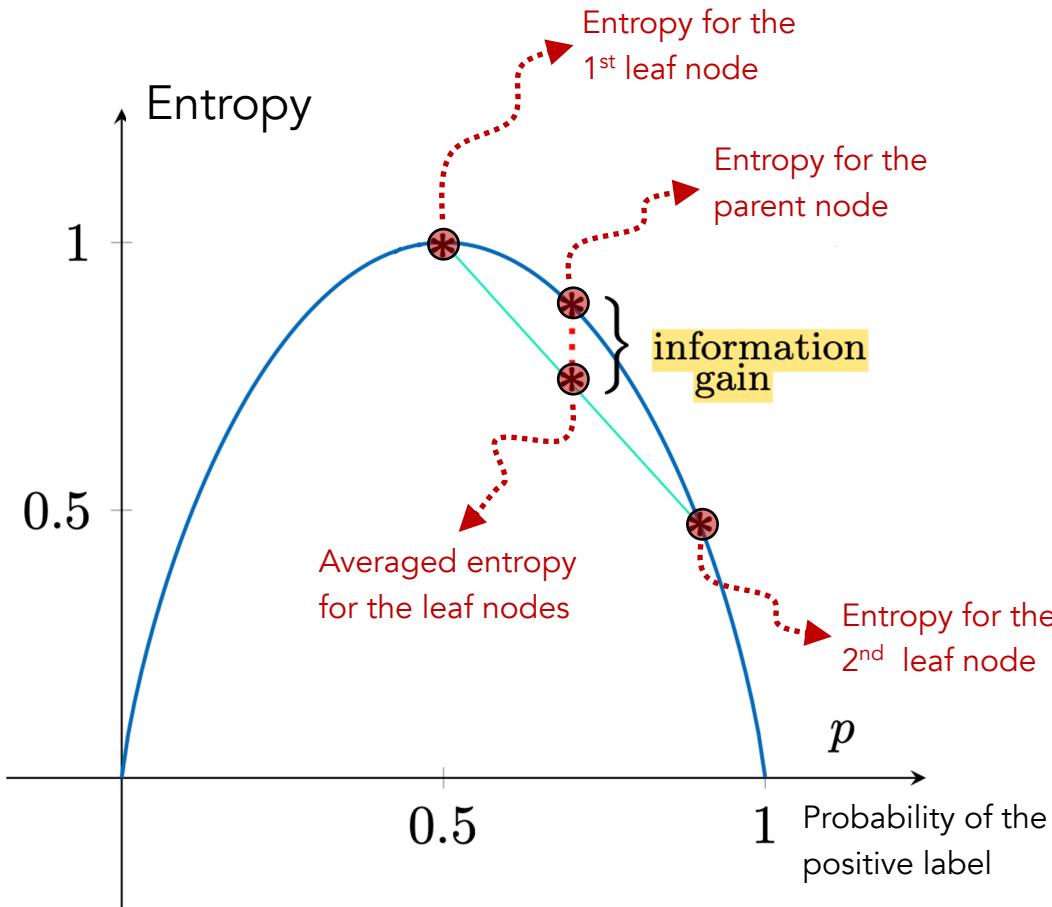
Leaf nodes:

- Total misclassification error = 0.1
- Averaged entropy = 0.43

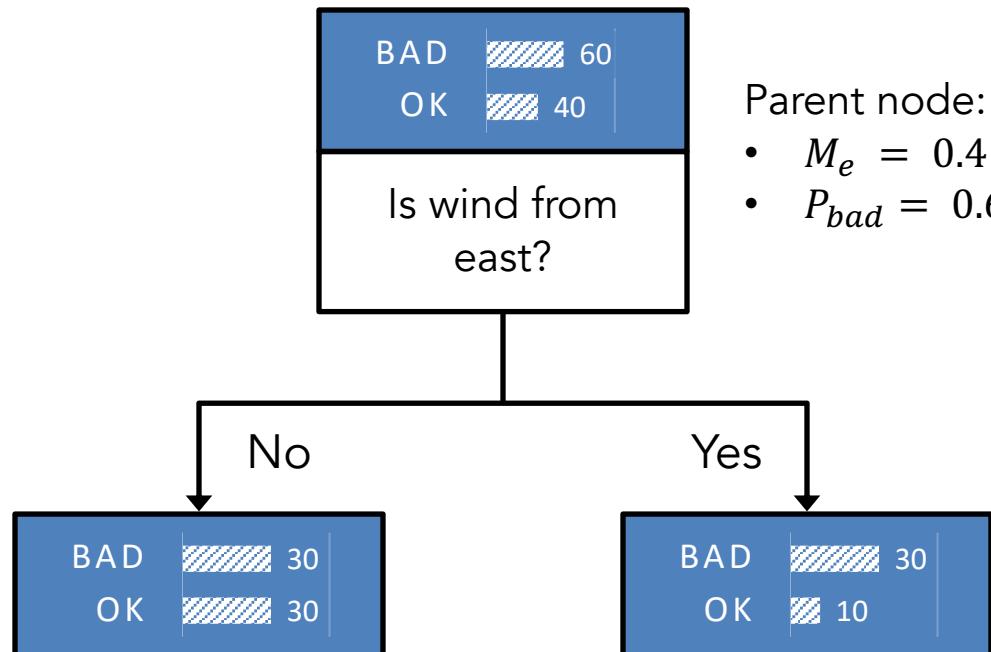
Leaf nodes:

- Total misclassification error = 0.1
- Averaged entropy = 0.32

The misclassification error is not very sensitive to changes in probabilities and can lead to zero information gain, as shown in the bottom right graph.



Below is an example of zero information gain when using the misclassification error to split nodes. The tree at the bottom left corresponds to the graph at the right.



1st leaf node:

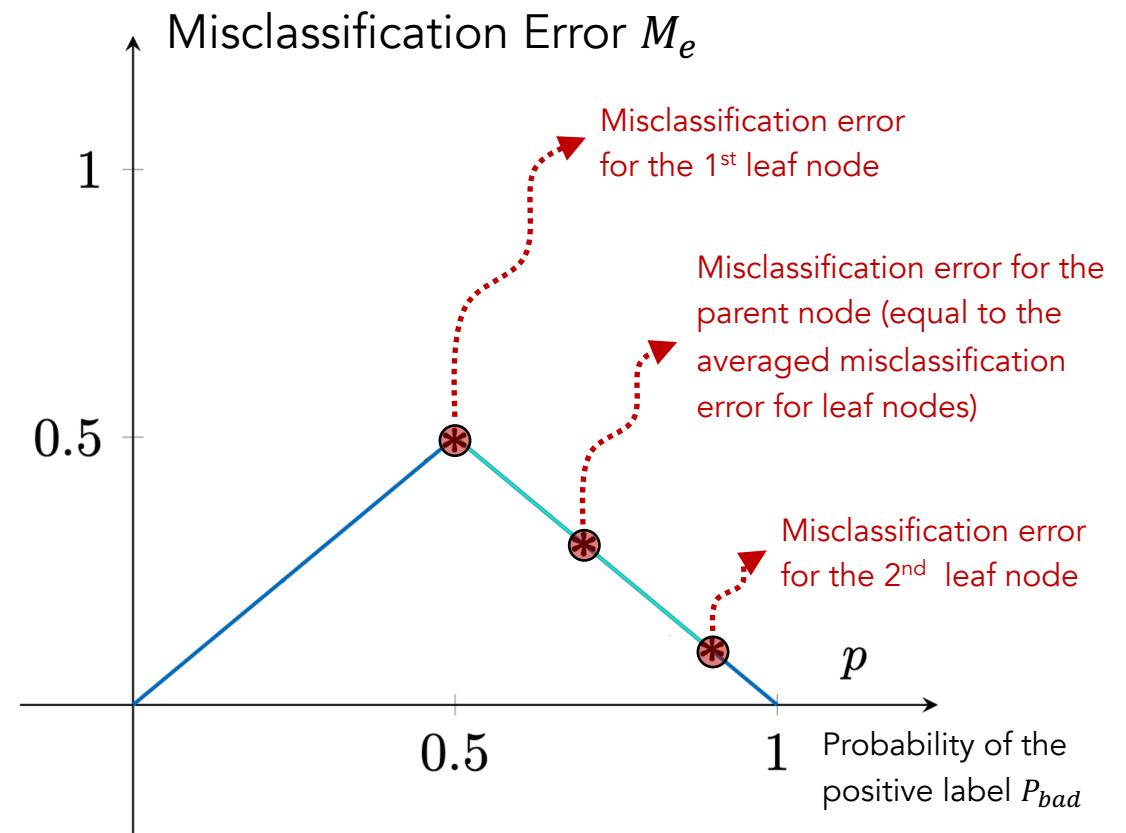
- $M_e = 0.5$
- $P_{bad} = 0.5$

2nd leaf node:

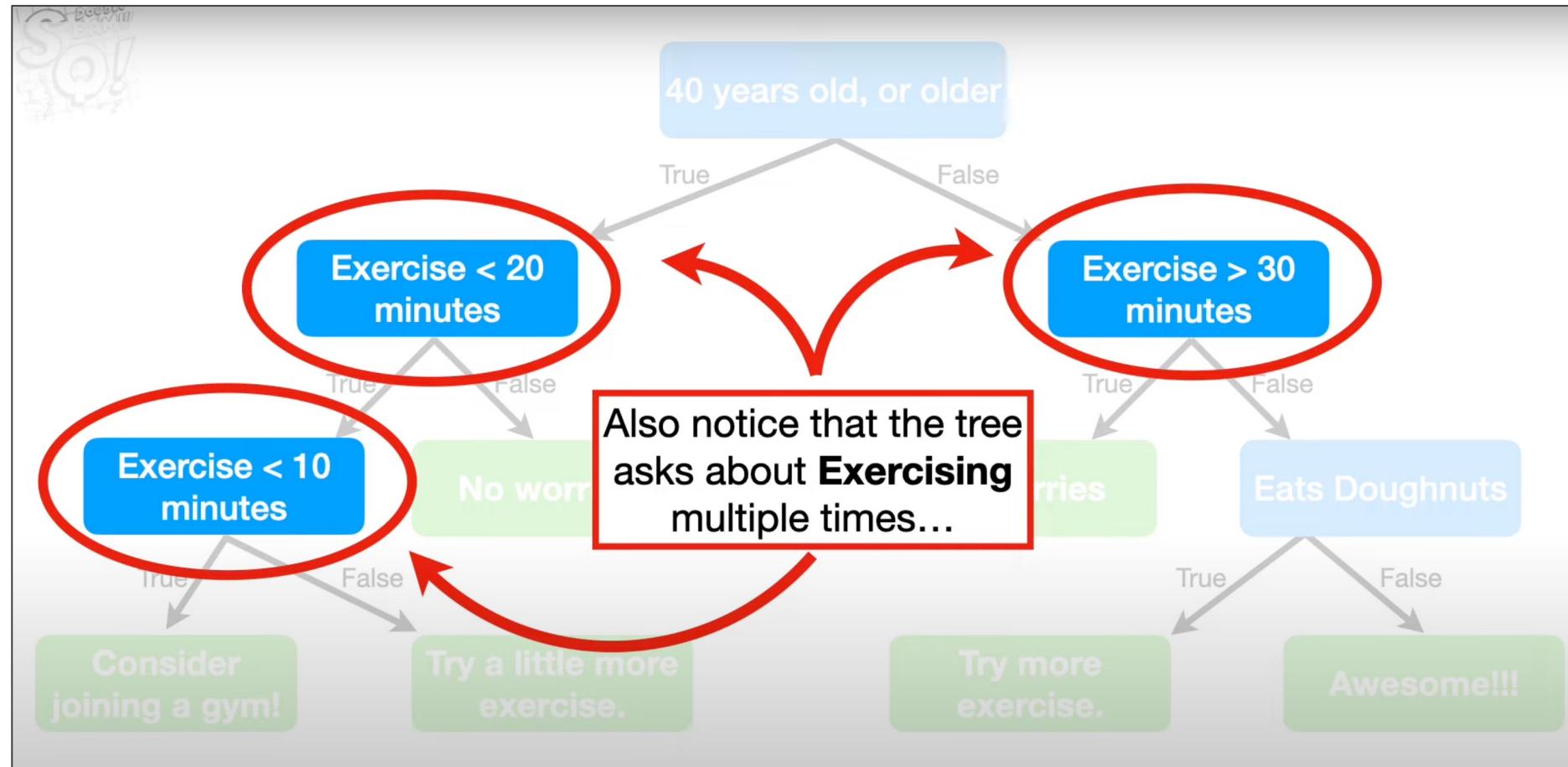
- $M_e = 0.25$
- $P_{bad} = 0.75$

Parent node:

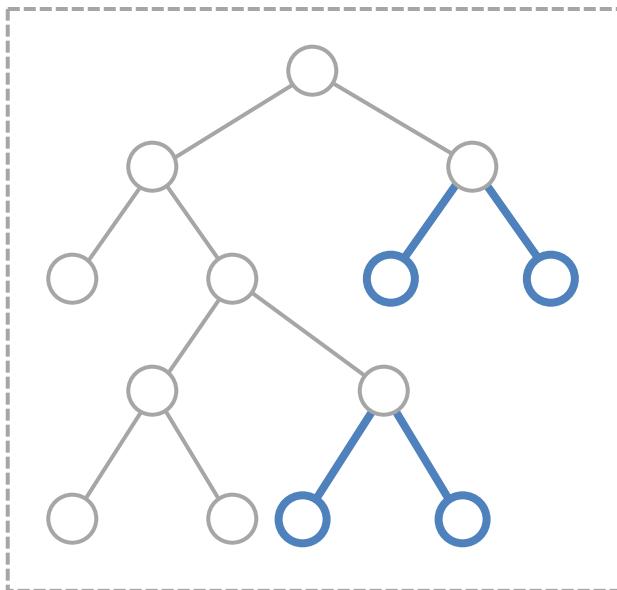
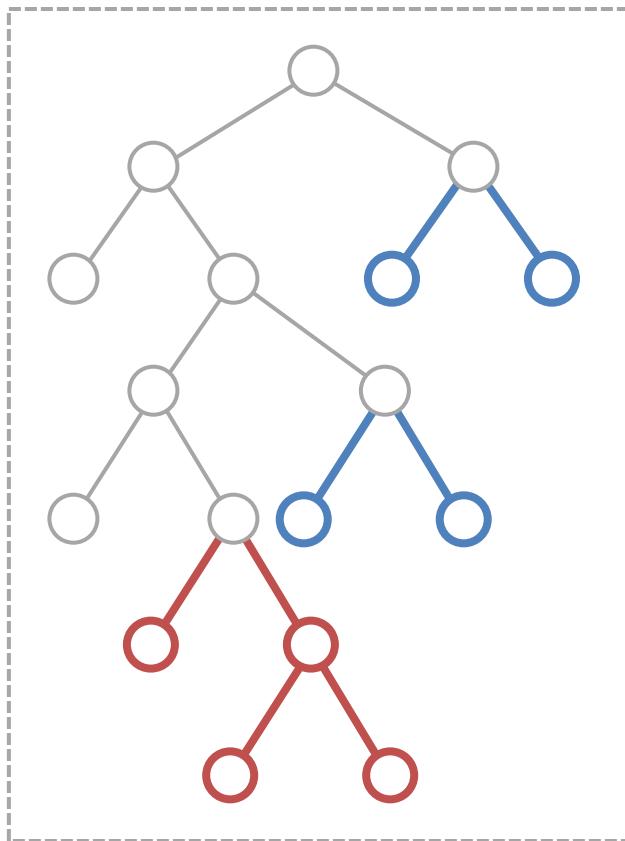
- $M_e = 0.4$
- $P_{bad} = 0.6$



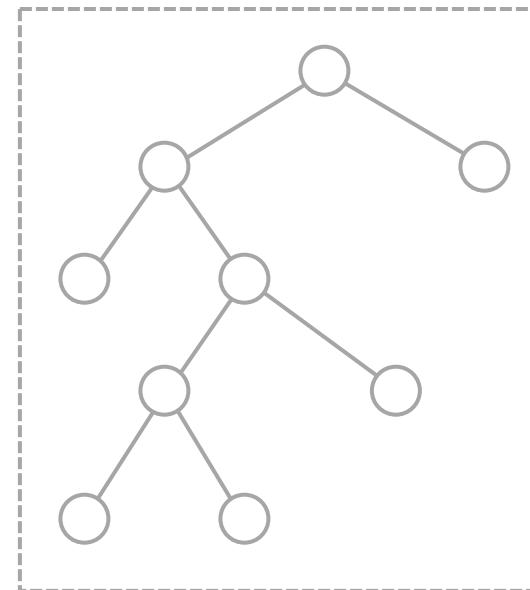
Decision Tree also works on continuous features but requires extra care. For example, one method is to **bin the continuous values** and treat each bin as a categorical feature.



Decision Tree can overfit easily. To combat overfitting, we can stop splitting a node when it reaches the maximum tree depth or does not have a minimum sample size.

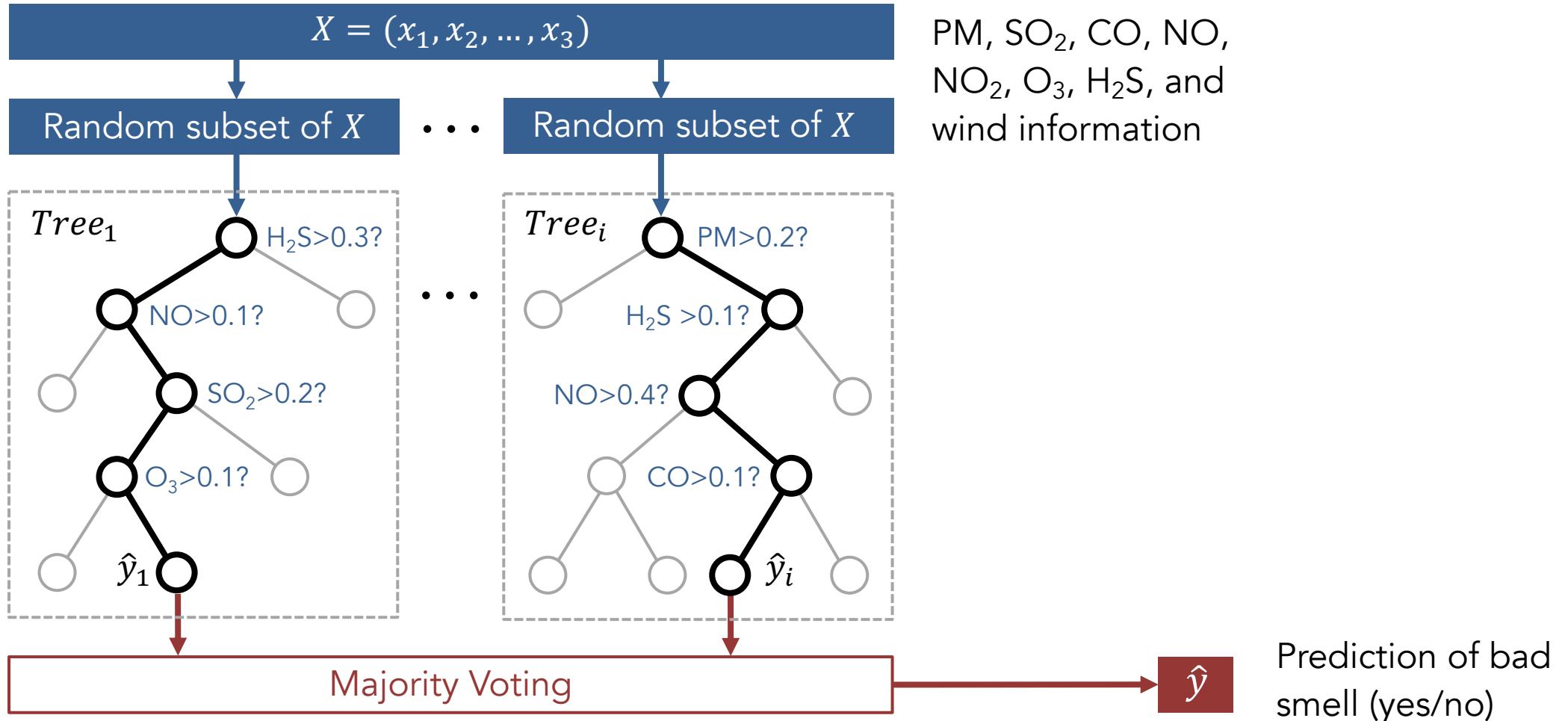


Limit max depth but allow
a small sample size.



Limit max depth and
restrict sample size.

We can also mitigate the overfitting problem of the Decision Tree by using the [bagging technique](#), which is an ensemble of multiple trees, such as the Random Forest model.



PM, SO₂, CO, NO,
NO₂, O₃, H₂S, and
wind information

...

$Tree_i$

$PM > 0.2?$

$H_2S > 0.1?$

$NO > 0.4?$

$CO > 0.1?$

\hat{y}_i

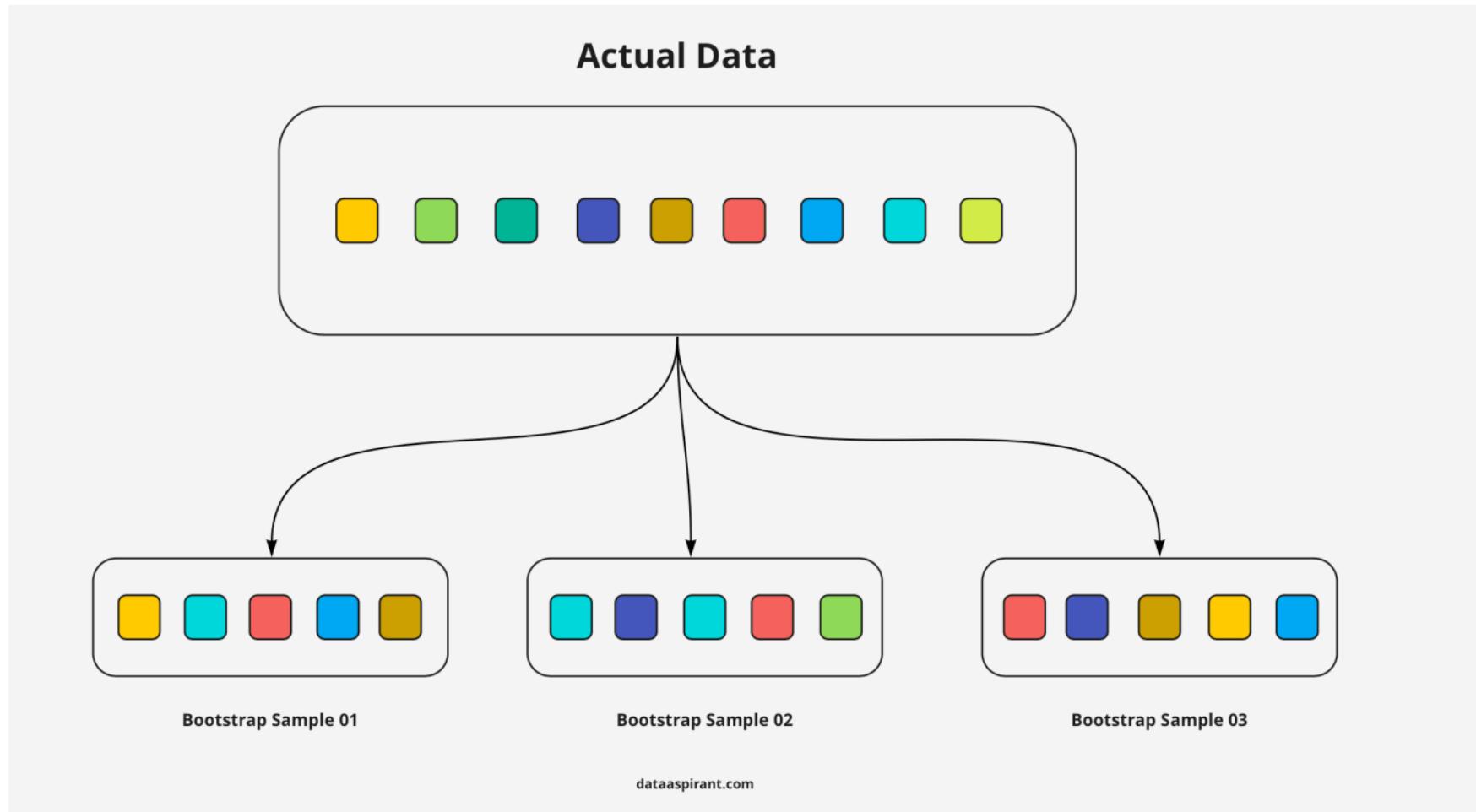
...

Majority Voting

Prediction of bad
smell (yes/no)

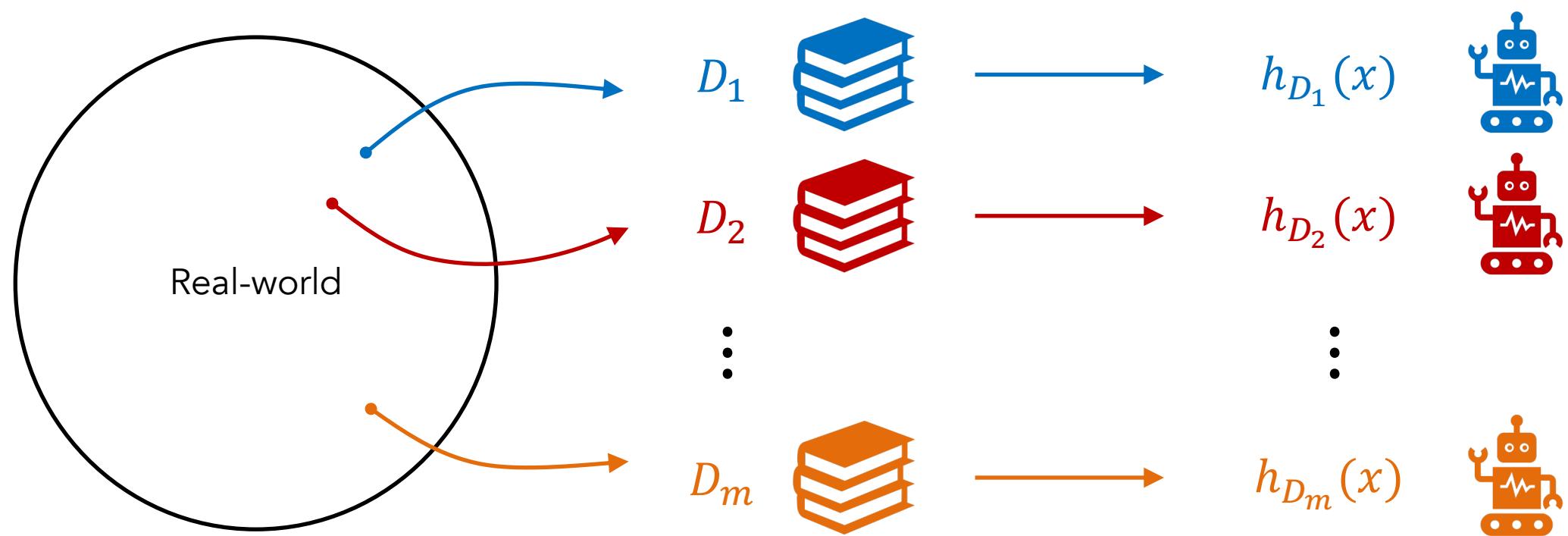
\hat{y}

The bagging technique for the Random Forest model uses **randomly selected features** and **bootstrapped samples** (i.e., sampling with replacement).



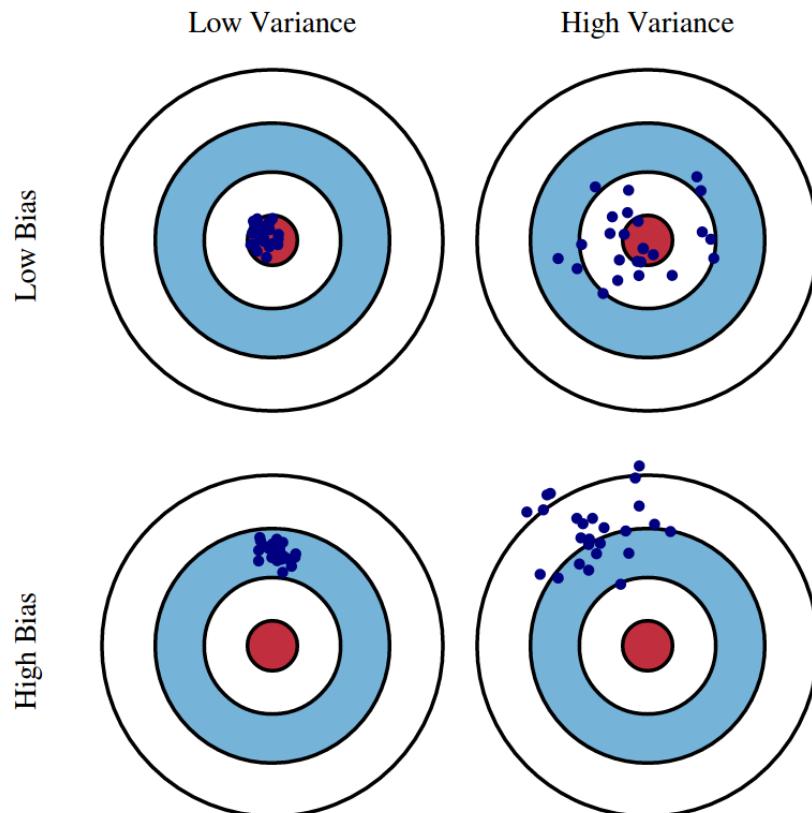
Why and how does the bagging technique work in dealing with overfitting?

Statistically speaking, the classifier that we trained is one of the all possible classifiers (i.e., drawn from a statistical distribution). We can sample many datasets D with pairs of features x and labels y . For all D , we can train a set of models $\{h_{D_1}(x), \dots, h_{D_m}(x)\}$.



The generalization errors of a model can be decomposed into bias, variance, and noise.

$$\underbrace{\mathbb{E}[(h_D(x) - y)^2]}_{\text{Error}} = \underbrace{\mathbb{E}[(h_D(x) - \bar{h}(x))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(x) - \bar{y}(x))^2]}_{\text{Bias}} + \underbrace{\mathbb{E}[(\bar{y}(x) - y(x))^2]}_{\text{Noise}}$$



\mathbb{E} : the expected value (i.e., average value)

D : the training dataset $(x^{(i)}, y^{(i)})$ pairs, sampled from P

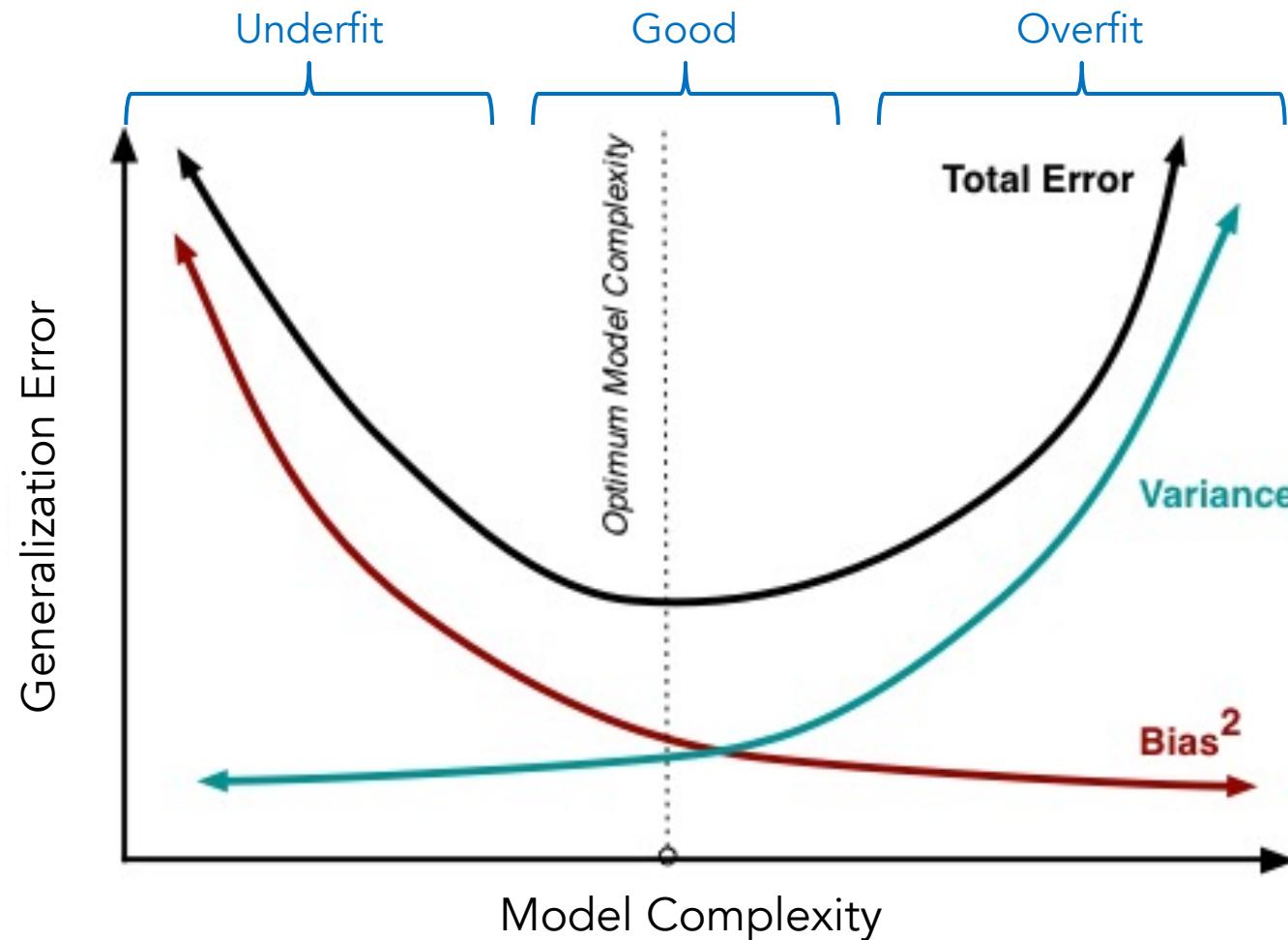
$h_D(x)$: the classification/regression model, trained on D

$\bar{h}(x)$: the average model (from all possible models)

y : the true labels that are used for testing a model

$\bar{y}(x)$: the average label value given feature x

Overfitting comes from training a very complex model that has a **high variance**. Both overfitting and underfitting means that the model does not generalize well on new data.



In practice, we estimate the generalization error using a validation set (i.e., validation error). Overfitting usually happens when the model has a low training error but a high validation error.

| | Underfitting | Just right | Overfitting |
|------------------------------------|--|---|---|
| Symptoms | <ul style="list-style-type: none"> • High training error • Training error close to test error • High bias | <ul style="list-style-type: none"> • Training error slightly lower than test error | <ul style="list-style-type: none"> • Very low training error • Training error much lower than test error • High variance |
| Regression illustration | | | |
| Classification illustration | | | |
| Deep learning illustration | | | |
| Possible remedies | <ul style="list-style-type: none"> • Complexify model • Add more features • train longer | | <ul style="list-style-type: none"> • Perform regularization • Get more data |

We can use the [weak law of large numbers](#) to reduce the variance of a complex model.

$$\underbrace{\mathbb{E}[(h_D(x) - y)^2]}_{\text{Error}} = \underbrace{\mathbb{E}[(h_D(x) - \bar{h}(x))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(x) - \bar{y}(x))^2]}_{\text{Bias}} + \underbrace{\mathbb{E}[(\bar{y}(x) - y(x))^2]}_{\text{Noise}}$$

Our goal is to reduce the variance term: $\mathbb{E}[(h_D(x) - \bar{h}(x))^2]$.

For this, we want $h_D \rightarrow \bar{h}$.

The weak law of large numbers says (roughly) for i.i.d. random variables x_i with mean \bar{x} , we have,

$$\frac{1}{m} \sum_{i=1}^m x_i \rightarrow \bar{x} \text{ as } m \rightarrow \infty$$

Apply this to classifiers: Assume we have m training sets D_1, D_2, \dots, D_n drawn from P^n . Train a classifier on each one and average result:

$$\hat{h} = \frac{1}{m} \sum_{i=1}^m h_{D_i} \rightarrow \bar{h} \quad \text{as } m \rightarrow \infty$$

We refer to such an average of multiple classifiers as an **ensemble** of classifiers.

Bagging is one of the [ensemble learning](#) methods, where multiple weak classifiers are combined into a stronger classifier using various techniques.

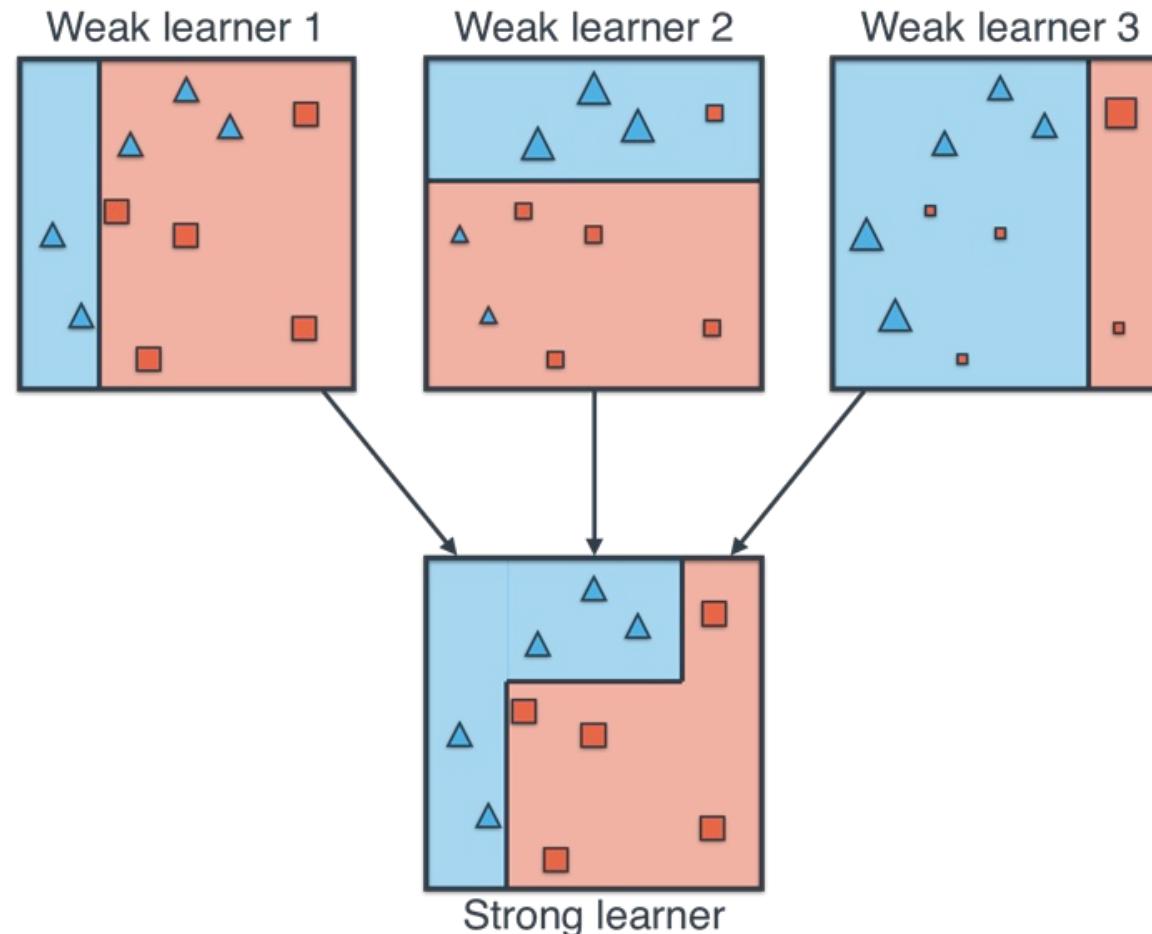
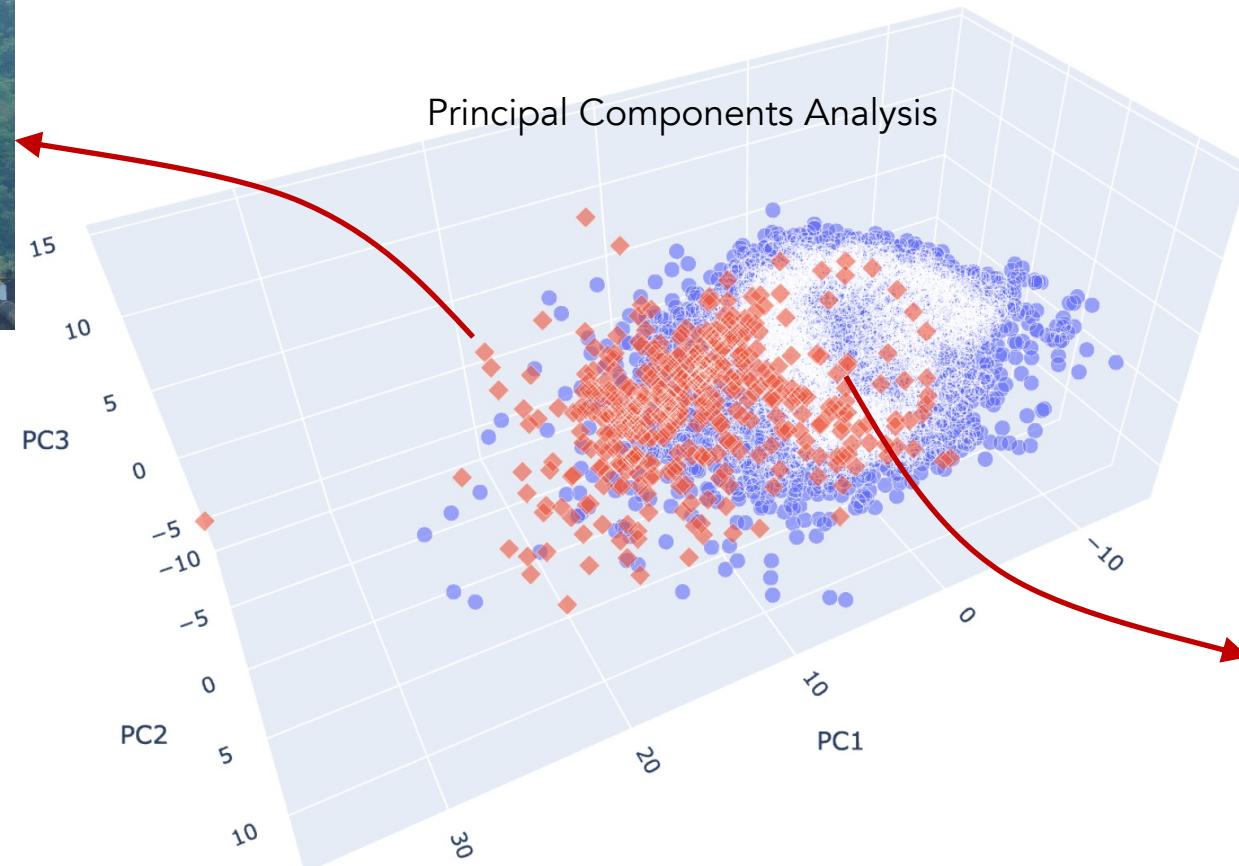


Figure source -- <https://mathchi.medium.com/weak-learners-strong-learners-for-machine-learning-e73e32f86ebd>

Not everything in the data is learnable, and we typically consider these as **noise**. For example, smell reports could come from sources that may not have obvious patterns.



Smell from pollution may have patterns.



Source of the right photo -- <https://unsplash.com/photos/YZQd7ICWAsQ>

Reducing the dimensions using **PCA (Principal Components Analysis)** can be a good idea for reducing computational cost or visualizing data. However, explaining what these reduced dimensions (i.e., principal components) mean becomes hard.

| | |
|-------------------------|---|
| O ₃ : 26 ppb | CO: 127 ppb |
| H ₂ S: 0 ppb | PM _{2.5} : 9 µg/m ³ |
| Wind: 17 deg | ... |

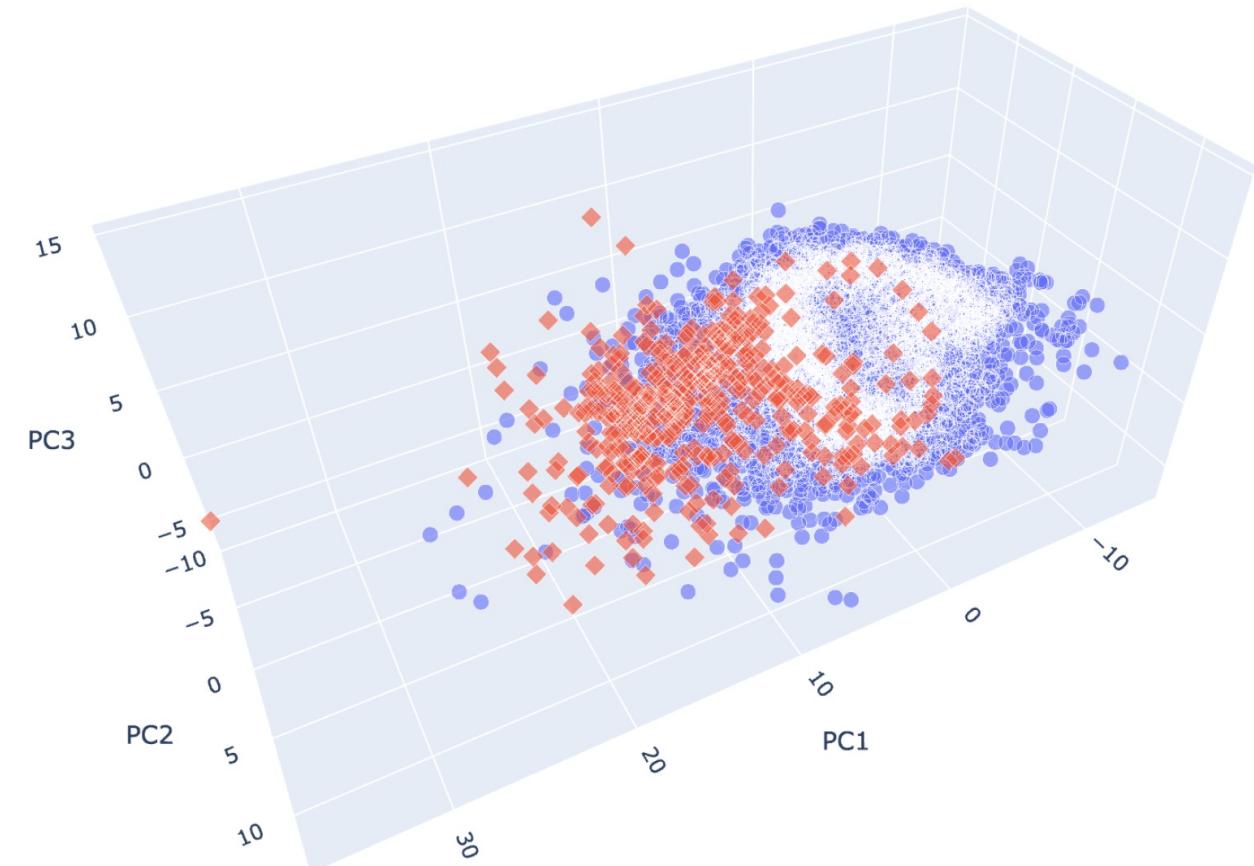
Observation 1

| | |
|-------------------------|--|
| O ₃ : 1 ppb | CO: 1038 ppb |
| H ₂ S: 9 ppb | PM _{2.5} : 23 µg/m ³ |
| Wind: 213 deg | ... |

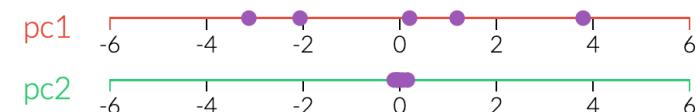
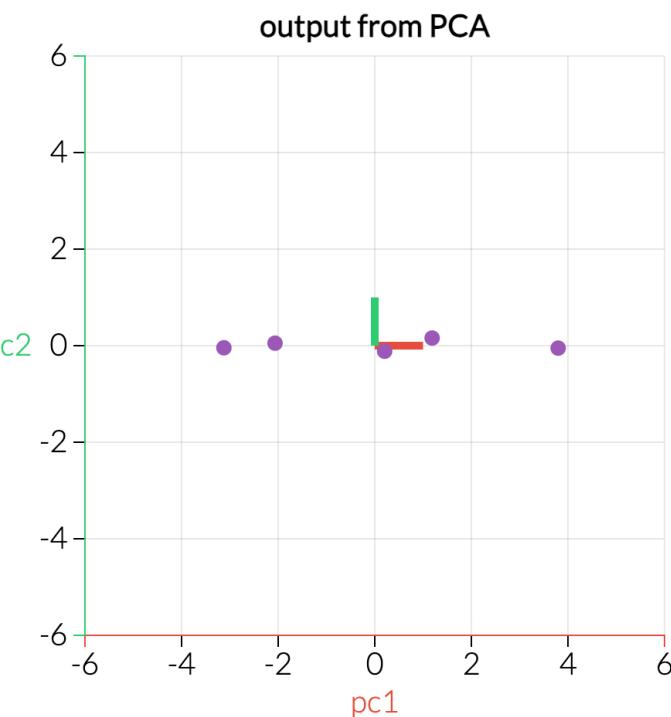
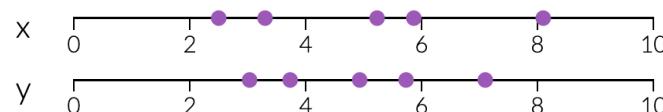
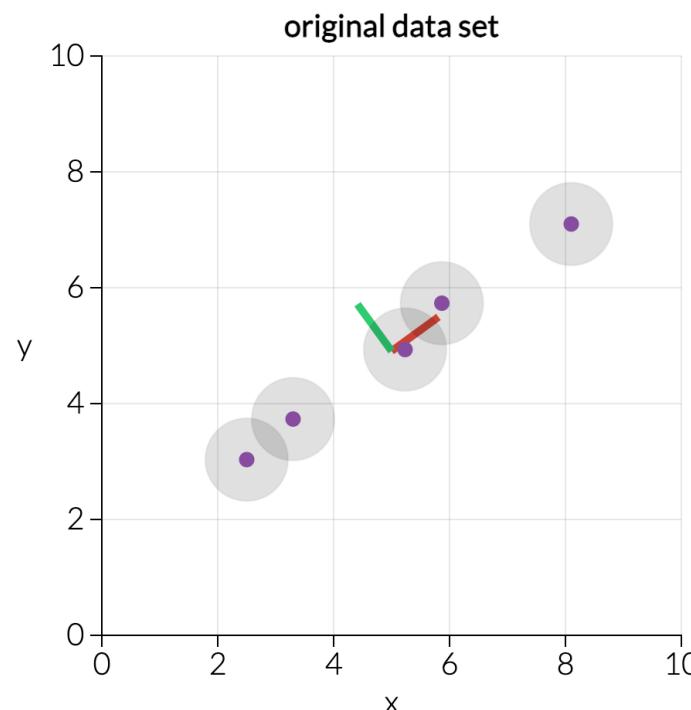
Observation 2

⋮

→ PCA

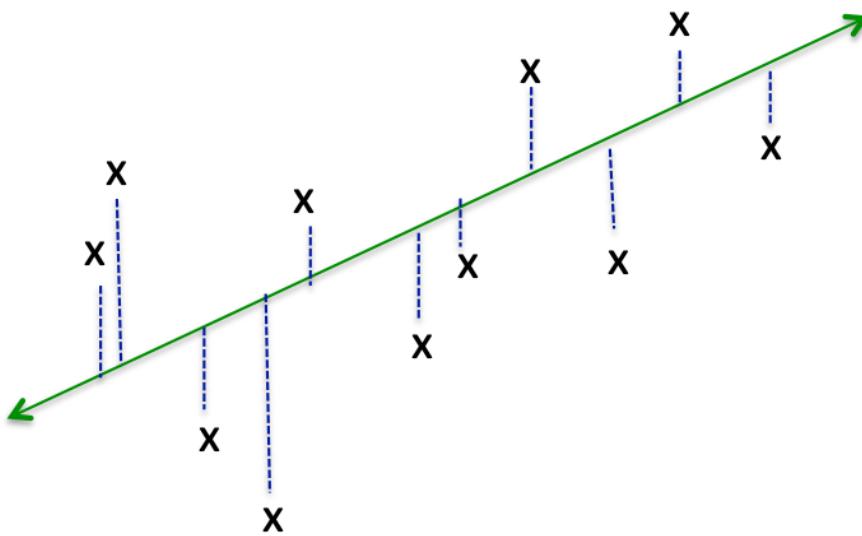


PCA finds a new orthogonal coordinate system by rotating the axes to identify the directions (i.e., principal components) that capture the largest variation in the data.

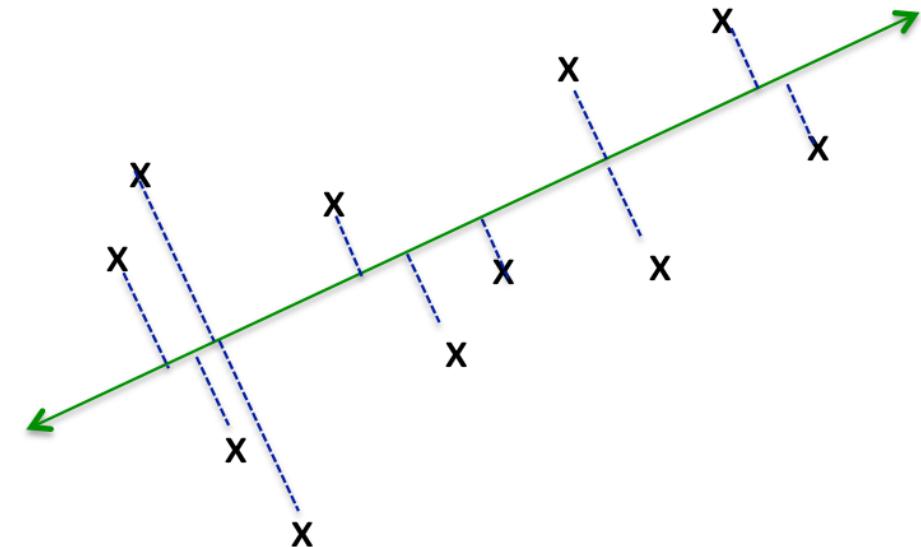


Keep pc1
Discard pc2

PCA minimizes the sum of squared perpendicular distances between the data points and the line, while linear regression minimizes the sum of squared vertical distances.



(a) Linear regression

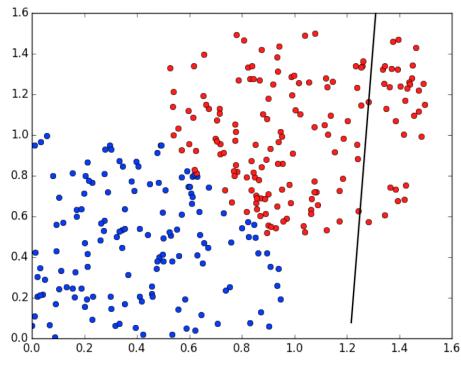


(b) PCA

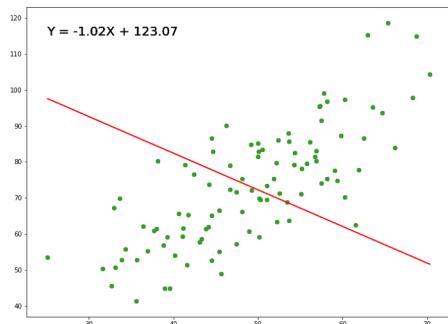
PCA is a form of unsupervised learning, which means learning with no labeled data.

Classification/regression are supervised learning techniques that requires labeled data.

Classification

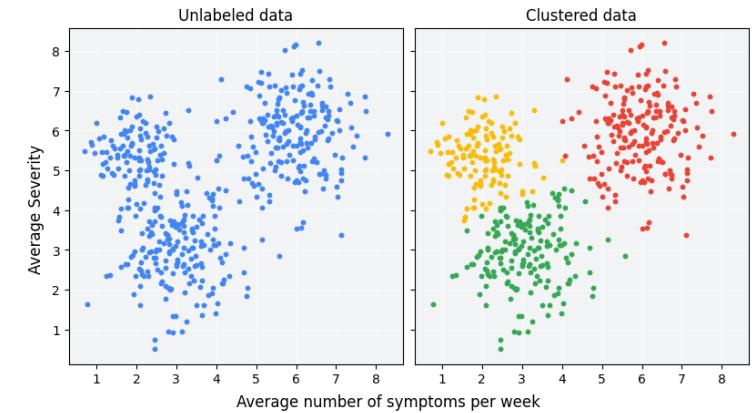


Regression

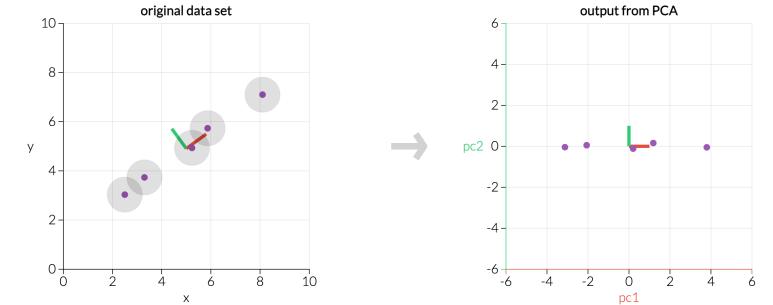


Supervised Learning

Clustering



Dimension Reduction



Unsupervised Learning

Take-Away Messages

- Decision Tree has a non-linear decision boundary.
- Decision Tree can overfit easily. You can limit the tree depth or sample size to mitigate the problem.
- Entropy and misclassification error can help us find the best feature to split a tree node.
- Entropy is the averaged surprise from flipping a coin with two classes (in the binary setting).
- We want to see the entropy as small as possible after splitting a tree node.
- Random Forest can be seen as a committee of Decision Tree models.
- The bagging technique can reduce the variance term in the error by combining multiple models.
- We can use PCA to reduce dimensions for visualizing data or reducing computational cost.



Questions?