Bayes' Theorem

The probability of spam given the word "free" is $P(\text{Spam}|\text{Free})$.

$$P(\text{Spam}|\text{Free}) = \frac{P(\text{Free}|\text{Spam}) \cdot P(\text{Spam})}{P(\text{Free})}$$

Definition:

Bayes' theorem provides a mathematical rule for inverting conditional probabilities, allowing us to find the probability of a cause given its effect.

It's a math rule that helps us **figure out the likelihood** of **something happening based on what we already know**. In other words, it helps us **update our beliefs when new information comes in**.

Application: It helps in updating the probability of an event (cause) based on new evidence (effect).

Example:

If age increases the risk of developing health problems, Bayes' theorem can assess the risk to an individual by conditioning it on their age, instead of assuming they are typical of the general population.

Base-Rate Fallacy:

Bayes' theorem helps in avoiding the base-rate fallacy by considering both the prevalence of a disease and the error rate of a test when interpreting a positive test result.

Bayesian Inference:

A key application is in Bayesian inference, where it is used to:

Reverse the probability of observations given a model (likelihood function).

Calculate the probability of the model configuration given the observations (posterior probability).

Prior Probability (P(A)):

This is the **probability of an event or hypothesis before any new evidence is observed**. It reflects the **belief about the event** or hypothesis based on **existing knowledge or assumptions.** For example, if you are predicting the weather and have no new data, you might estimate the prior probability of rain to be 30%.

Posterior Probability (P(A|B)):

This is the **probability of an event or hypothesis after considering new evidence or data** It is updated based on the prior probability and the likelihood of the new evidence. Mathematically, the posterior probability is calculated using Bayes' Theorem:

Bayes' Theorem is given by:

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

Where:

- $P(A|B)$ is the **posterior probability** (the probability of A given B),
- $P(B|A)$ is the **likelihood** (the probability of B given A),
- $P(A)$ is the **prior probability** (our initial belief about A),
- $P(B)$ is the **total probability** of observing B.

Example: Spam Email

We want to calculate the probability that an email is spam, given that it contains the word "free."

Step 1: Use Bayes' Theorem

Bayes' Theorem is:

$$P(\text{Spam}|\text{Free}) = \frac{P(\text{Free}|\text{Spam}) \cdot P(\text{Spam})}{P(\text{Free})}$$

Where (P(\text{Free})) is the total probability of seeing the word "free," calculated as:

$$P(\text{Free}) = P(\text{Free}|\text{Spam}) \cdot P(\text{Spam}) + P(\text{Free}|\text{Not Spam}) \cdot P(\text{Not Spam})$$

Step 2: Calculate $P(\text{Free})$

Now calculate the total probability of seeing the word "free":

$$P(\text{Free}) = (0.7 \times 0.3) + (0.2 \times 0.7)$$

$$P(\text{Free}) = 0.21 + 0.14 = 0.35$$

Step 3: Apply Bayes' Theorem

Substitute the values into Bayes' Theorem:

$$P(\text{Spam}|\text{Free}) = \frac{0.7 \times 0.3}{0.35}$$

$$P(\text{Spam}|\text{Free}) = \frac{0.21}{0.35} \approx 0.6$$

Result:

The probability that an email is spam, given that it contains the word "free," is **60%**.

```
import matplotlib.pyplot as plt

# Initial Inputs: Given probabilities
P_Free_given_Spam = 0.7  # P(Free|Spam): Probability that the word "Free" appears given that the email is Spam
P_Spam = 0.3  # P(Spam): Prior probability that an email is Spam
P_Free_given_NotSpam = 0.2  # P(Free|Not Spam): Probability that the word "Free" appears given that the email is Not Spam
P_NotSpam = 0.7  # P(Not Spam): Prior probability that an email is Not Spam

# Step 1: Calculate P(Free) using the law of total probability
P_Free = (P_Free_given_Spam * P_Spam) + (P_Free_given_NotSpam * P_NotSpam)

# Step 2: Apply Bayes' Theorem to calculate P(Spam|Free)
P_Spam_given_Free = (P_Free_given_Spam * P_Spam) / P_Free
P_NotSpam_given_Free = 1 - P_Spam_given_Free

# Visualization of the process
labels = ['P(Spam|Free)', 'P(Not Spam|Free)']
probabilities = [P_Spam_given_Free, P_NotSpam_given_Free]

# Plotting the results
fig, ax = plt.subplots(figsize=(12, 9))
ax.bar(labels, probabilities, color=['skyblue', 'lightcoral'])
ax.set_title('Probability of Email Being Spam Given the Word "Free"', fontsize=18)
ax.set_ylabel('Probability', fontsize=18)
ax.set_ylim(0, 1)

# Display the probability values on top of the bars
for i, v in enumerate(probabilities):
    ax.text(i, v + 0.05, f'{v:.2f}', ha='center', fontsize=18)

# Adding a legend with the formulas used for the calculation
ax.text(0.5, 0.95, f'Bayes\' Theorem: P(Spam|Free) = (P(Free|Spam) * P(Spam)) / P(Free)\n'
        f'P(Free) = P(Free|Spam) * P(Spam) + P(Free|Not Spam) * P(Not Spam)',
        ha='center', va='top', fontsize=14, transform=ax.transAxes)

# Adding the input values for clarity
ax.text(0.5, 0.85, f'P(Spam) = {P_Spam}\nP(Free|Spam) = {P_Free_given_Spam}\n'
        f'P(Free|Not Spam) = {P_Free_given_NotSpam}\nP(Not Spam) = {P_NotSpam}',
        ha='center', va='top', fontsize=14, transform=ax.transAxes)

plt.tight_layout()
plt.show()
```

Information Gain Definition

In general terms, the expected information gain is the reduction in information entropy H from a prior state to a state that takes some information as given:

$$IG(T, a) = \mathrm{H}(T) - \mathrm{H}(T \mid a),$$

where:

H(T) is the entropy of the entire dataset before the split.

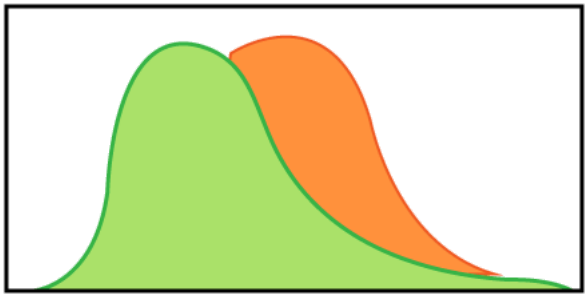
H(T | a) is the weighted average entropy of the subsets after splitting based on attribute a.

also where **H (T | a) is the conditional entropy of T given the value of attribute a**.

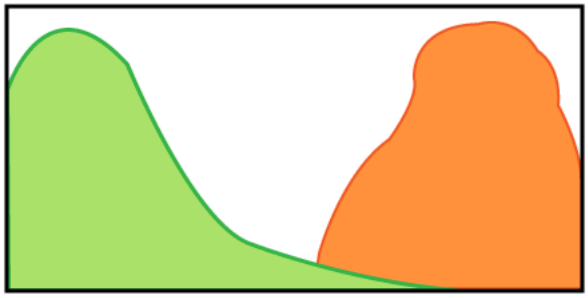
This is intuitively plausible when interpreting **entropy H as a measure of uncertainty of a random variable T: by learning (or assuming) a about T,**

our uncertainty about T is reduced (i.e. IG(T,a) is positive), unless of course T is independent of a, in which case

H(T|a) = H(T), meaning IG(T,a) = 0.



Low information gain High entropy



High information gain Low entropy

Example of No Information Gain:

If T and a are independent, knowing a does not change our understanding of T:

In this case

IG(T, A) = H(T) - H(T|A)

Given that H(T|A) = H(T):

IG(T, A) = H(T) - H(T) = 0

Entropy as Uncertainty:

High Entropy: A system with high entropy is highly uncertain or disordered.

Low Entropy: A system with low entropy is more certain or ordered.

Information Gain as Uncertainty Reduction:

When we gain information about a system (like learning the value of attribute 'a' for a given instance), we reduce the uncertainty about the system's outcome (the value of the target variable). This reduction in uncertainty is quantified by the information gain.

Imagine you're trying to sort a pile of mixed-up toys.

You want to put all the cars together, all the dolls together, and all the blocks together.

Information gain is like a special tool that helps you decide which way to sort the toys first.

Let's say you have two ways to sort:

By color: You could sort the toys into red, blue, and green piles.

By type: You could sort the toys into car, doll, and block piles.

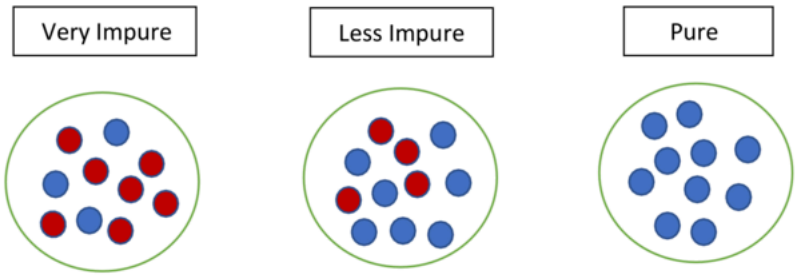
Which way is the best?

That's where information gain comes in! It helps you figure out which way will make the piles more organized and easier to sort.

If you sort by type first, you'll probably end up with piles that are mostly one type of toy. This means you've gained a lot of information about how to sort the rest of the toys.

If you sort by color first, you might end up with piles that still have a mix of toys. This means you haven't gained as much information, and you'll still have to do more sorting.

From Encylopedia



Another Take on Information Gain, with Example

For a better understanding of information gain, let us break it down. As we know, **information gain is the reduction in information entropy**, what is entropy? Basically, **entropy is the measure of impurity or uncertainty in a group of observations.**

In engineering applications, **information is analogous to signal**, and **entropy is analogous to noise**.

Information = Signal: The meaningful content or message in a system. Entropy = Noise: The randomness or uncertainty that obscures or distorts the signal.

In essence, in engineering, we aim to **maximize the signal (information) and minimize the noise (entropy)** to ensure that the communication or data transmission is as accurate and clear as possible.

It determines how a decision tree chooses to split data.[1]

The leftmost figure below is very impure and has high entropy corresponding to higher disorder and lower information value.

As we go to the right, the entropy decreases, and the information value increases.

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
import seaborn as sns

# Example Dataset
data = {
    'Contains_Free': [1, 0, 1, 1, 0, 1, 0, 0, 1, 0],  # 1: Yes, 0: No
    'Spam': [1, 0, 1, 0, 0, 1, 0, 1, 0, 0]  # 1: Spam, 0: Not Spam
}

# Create DataFrame
df = pd.DataFrame(data)

# Split features and target
X = df[['Contains_Free']]
y = df['Spam']

# Initialize Decision Tree Classifier
clf = DecisionTreeClassifier(criterion='entropy', max_depth=1)  # Use 'entropy' to calculate information gain
clf.fit(X, y)

# Visualize the Decision Tree
plt.figure(figsize=(10, 6))
plot_tree(clf, feature_names=['Contains_Free'], class_names=['Not Spam', 'Spam'], filled=True, rounded=True)
plt.title("Decision Tree Visualization Showing Information Gain")
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Create a function to plot distributions with entropy and information gain, with color filling
def plot_distributions(ax, before_split, after_split_left, after_split_right, entropy_before, entropy_after, info_gain, title
    x = np.linspace(-5, 5, 1000)

    # Plot the original distribution (before split)
    ax.plot(x, norm.pdf(x, loc=before_split[0], scale=before_split[1]), label="Before Split (High Entropy)", color="blue", lw
```

```
    ax.plot(x, norm.pdf(x, loc=before_split[0], scale=before_split[1]), label="Before Split (High Entropy)", color="blue", lw=2
    ax.fill_between(x, 0, norm.pdf(x, loc=before_split[0], scale=before_split[1]), color="blue", alpha=0.3)

    # Plot the distributions after the split
    ax.plot(x, norm.pdf(x, loc=after_split_left[0], scale=after_split_left[1]), label="After Split - Left", color="red", lw=2
    ax.fill_between(x, 0, norm.pdf(x, loc=after_split_left[0], scale=after_split_left[1]), color="red", alpha=0.3)

    ax.plot(x, norm.pdf(x, loc=after_split_right[0], scale=after_split_right[1]), label="After Split - Right", color="green",
    ax.fill_between(x, 0, norm.pdf(x, loc=after_split_right[0], scale=after_split_right[1]), color="green", alpha=0.3)

    ax.set_title(f"{title}\nEntropy Before Split: {entropy_before:.3f} | Entropy After Split: {entropy_after:.3f} | Informati
    ax.legend()
    ax.set_xlabel("Feature Value")
    ax.set_ylabel("Density")
    ax.grid(True)

# Create subplots for all three scenarios with their sub-cases
fig, axs = plt.subplots(3, 3, figsize=(18, 18))

# Scenario 1: High Entropy, Low Information Gain
# Sub-case 1: Small reduction in entropy
before_split_high_entropy = [0, 1]  # mean = 0, std = 1
after_split_left_high = [-1, 1]     # Left split: mean = -1, std = 1
after_split_right_high = [1, 1]     # Right split: mean = 1, std = 1
entropy_before_high = 1.0           # High entropy before split (uncertainty)
entropy_after_high = 0.9            # Small reduction in entropy
info_gain_high = entropy_before_high - entropy_after_high
plot_distributions(axs[0, 0], before_split_high_entropy, after_split_left_high, after_split_right_high, entropy_before_high,

# Sub-case 2: More noticeable reduction in entropy
after_split_left_high_2 = [-2, 0.7]
after_split_right_high_2 = [2, 0.7]
entropy_after_high_2 = 0.7
info_gain_high_2 = entropy_before_high - entropy_after_high_2
plot_distributions(axs[0, 1], before_split_high_entropy, after_split_left_high_2, after_split_right_high_2, entropy_before_hi

# Sub-case 3: Very small reduction in entropy
after_split_left_high_3 = [-0.5, 1.2]
after_split_right_high_3 = [0.5, 1.2]
entropy_after_high_3 = 0.95
info_gain_high_3 = entropy_before_high - entropy_after_high_3
plot_distributions(axs[0, 2], before_split_high_entropy, after_split_left_high_3, after_split_right_high_3, entropy_before_hi

# Scenario 2: High Information Gain, Low Entropy
# Sub-case 1: Clear separation of classes (large information gain)
before_split_low_entropy = [0, 1]
after_split_left_low = [-3, 0.5]
after_split_right_low = [3, 0.5]
entropy_before_low = 1.0
entropy_after_low = 0.2
info_gain_low = entropy_before_low - entropy_after_low
plot_distributions(axs[1, 0], before_split_low_entropy, after_split_left_low, after_split_right_low, entropy_before_low, entr

# Sub-case 2: Good separation but not perfect (moderate information gain)
after_split_left_low_2 = [-2, 0.8]
after_split_right_low_2 = [2, 0.8]
entropy_after_low_2 = 0.3
info_gain_low_2 = entropy_before_low - entropy_after_low_2
plot_distributions(axs[1, 1], before_split_low_entropy, after_split_left_low_2, after_split_right_low_2, entropy_before_low,

# Sub-case 3: Slight reduction in entropy (lower information gain)
after_split_left_low_3 = [-1.5, 1]
after_split_right_low_3 = [1.5, 1]
entropy_after_low_3 = 0.5
info_gain_low_3 = entropy_before_low - entropy_after_low_3
plot_distributions(axs[1, 2], before_split_low_entropy, after_split_left_low_3, after_split_right_low_3, entropy_before_low,

# Scenario 3: Zero Information Gain
# Sub-case 1: No split (original distribution stays the same)
before_split_zero = [0, 1]
after_split_left_zero = [0, 1]
after_split_right_zero = [0, 1]
entropy_before_zero = 0.0
entropy_after_zero = 0.0
info_gain_zero = entropy_before_zero - entropy_after_zero
plot_distributions(axs[2, 0], before_split_zero, after_split_left_zero, after_split_right_zero, entropy_before_zero, entropy_

# Sub-case 2: Splitting doesn't reduce entropy
after_split_left_zero_2 = [0, 1]
after_split_right_zero_2 = [0, 1]
entropy_after_zero_2 = 0.0
info_gain_zero_2 = entropy_before_zero - entropy_after_zero_2
plot_distributions(axs[2, 1], before_split_zero, after_split_left_zero_2, after_split_right_zero_2, entropy_before_zero, entr
```

```
# Sub-case 3: Another case where entropy stays the same post-split
after_split_left_zero_3 = [0, 1]
after_split_right_zero_3 = [0, 1]
entropy_after_zero_3 = 0.0
info_gain_zero_3 = entropy_before_zero - entropy_after_zero_3
plot_distributions(axs[2, 2], before_split_zero, after_split_left_zero_3, after_split_right_zero_3, entropy_before_zero, entr

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```