Spring 2024: CSCI 6521 Homework # 1
Name: Padam Jung Thapa
ID: 2623560

1. Information Gain:

- Information gain in the context of machine learning and decision trees is a measure of the effectiveness of an attribute in classifying a dataset. It quantifies the *reduction in entropy*, which is a measure of uncertainty or impurity in a group of examples, after the dataset is split on an attribute.

- To calculate information gain, we need to understand entropy first here. So, entropy is a measure of the uncertainty or disorder within a dataset. A high entropy value means that the dataset is very mixed, with no clear distinction between the classes of data points. Conversely, a low entropy value indicates that the dataset is more homogeneous, or pure, with respect to the class labels.

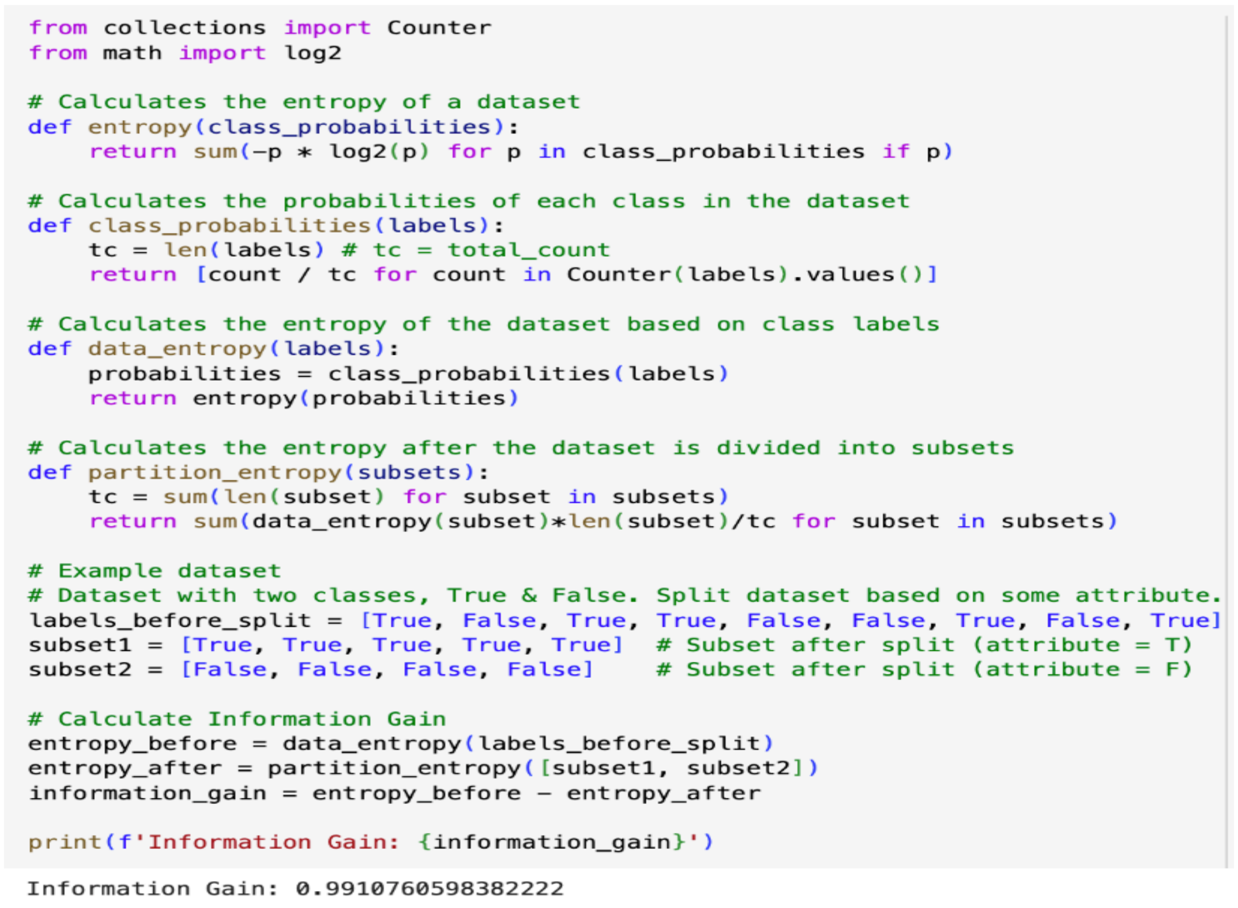
- Information Gain (IG) measures the change in entropy before and after a dataset is split on an attribute. It quantifies how much splitting on this attribute has reduced the uncertainty or disorder in the dataset. The formula for Information Gain is:

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

Where,

- IG(T,a) is the information gain of the dataset T after splitting on attribute a,
- H(T) is the entropy of the dataset before the split, and
- H(T|a) is the weighted average entropy of the dataset after the split on attribute a.[1][2]

- An Information Gain of approximately 0.991 indicates a substantial decrease in entropy due to the split, or in other words - a significant increase in the purity of the subsets. This means that the attribute used for the split has effectively categorized the dataset into subsets that are more uniform in terms of their class labels than the original dataset was.
- The pros of information gain is that it can work with both continuous and discrete variables and is less sensitive to noise. The cons of information gain is that it is more complex to calculate & is less interpretable.



```python
from collections import Counter
from math import log2

# Calculates the entropy of a dataset
def entropy(class_probabilities):
    return sum(-p * log2(p) for p in class_probabilities if p)

# Calculates the probabilities of each class in the dataset
def class_probabilities(labels):
    tc = len(labels) # tc = total_count
    return [count / tc for count in Counter(labels).values()]

# Calculates the entropy of the dataset based on class labels
def data_entropy(labels):
    probabilities = class_probabilities(labels)
    return entropy(probabilities)

# Calculates the entropy after the dataset is divided into subsets
def partition_entropy(subsets):
    tc = sum(len(subset) for subset in subsets)
    return sum(data_entropy(subset)*len(subset)/tc for subset in subsets)

# Example dataset
# Dataset with two classes, True & False. Split dataset based on some attribute.
labels_before_split = [True, False, True, True, False, False, True, False, True]
subset1 = [True, True, True, True, True]   # Subset after split (attribute = T)
subset2 = [False, False, False, False]     # Subset after split (attribute = F)

# Calculate Information Gain
entropy_before = data_entropy(labels_before_split)
entropy_after = partition_entropy([subset1, subset2])
information_gain = entropy_before - entropy_after

print(f'Information Gain: {information_gain}')
```

```
Information Gain: 0.9910760598382222
```

- In practical terms, this attribute is very informative because it significantly increases our ability to predict the class label of the data points based on the subsets they fall into after the split. In decision tree algorithms, attributes that result in high Information Gain are preferred for making splits because they lead to simpler trees and better classification performance. A high Information Gain obtained over here, suggests that the attribute used for the split is very effective at classifying the dataset, making it a good choice for the root or an internal node of a decision tree.

References:

[1] "Information gain ratio - Wikipedia."
https://en.wikipedia.org/wiki/Information_gain_ratio
[2] “How Is Information Gain Calculated? IBKR Quant.”
http://theautomatic.net/2020/02/18/how-is-information-gain-calculated/

2. Kullback–Leibler (KL) divergence:

- Kullback–Leibler (KL) divergence, also known as relative entropy, is a measure of how one probability distribution diverges from a second, expected probability distribution. It is a non-symmetric measure of the difference between two probability distributions P and Q, typically P representing the true distribution of data, observations, or a precisely calculated theoretical distribution, and Q representing a theory, model, description, or approximation of P.

- The KL divergence is defined for discrete probability distributions as:

$$D_{KL}(P||Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right)$$

- And for continuous probability distributions as:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

Where,

1. P and Q are the two probability distributions being compared,
2. P(i) and Q(i) are the probabilities of the i'th element in the discrete distributions,
3. p(x) and q(x) are the probability density functions of P and Q in the continuous case.

Properties:
1. KL Divergence is not symmetric. i.e. DKL(P||Q) ≠ DKL(Q||P). As a result, it is also not a distance metric.
2. The KL divergence can take on values in [0, ∞]. Particularly, if P and Q are the exact same distribution (P = Q), then DKL(P||Q) =0, and by symmetry DKL(P||Q) =0.
3. For KL divergence to be finite, the support of P needs to be contained in the support of Q. If a point x exists with Q(x) = 0 but P(x) > 0, then DKL(P||Q) = ∞

KL divergence is widely used in various fields such as machine learning, statistics, and information theory. In machine learning, it is often used for tasks such as:

- Model Selection: Comparing different models to see which one best approximates the true distribution.
- Feature Selection: Determining which features provide the most information about the outcome.
- Clustering: Measuring the divergence between different clusters of data.

It is also a key component of many algorithms, including:

- Expectation-Maximization (EM): For finding maximum likelihood estimates in the presence of latent variables.
- Variational Bayesian Methods: For approximating complex probability distributions.
- Information Gain: In decision tree learning, where it can be related to the concept of information gain.

The KL divergence is a foundational tool in the field of information theory, providing a way to quantify the difference between two probability distributions and thus the efficiency of encoding schemes or the fidelity of approximations to the true data distribution. [1][2]

```python
from scipy.special import rel_entr
import numpy as np

# Define two probability distributions P and Q
# Note: The probabilities in each distribution should sum to 1.
P = [.05, .1, .2, .05, .15, .25, .08, .12]
Q = [.3, .1, .2, .1, .1, .02, .08, .1]

# Calculate the KL divergence from P to Q
kl_divergence = sum(rel_entr(P, Q))

print(f'KL divergence from P to Q: {kl_divergence:.3f} nats')

# Calculate the KL divergence from Q to P for comparison
kl_divergence_reverse = sum(rel_entr(Q, P))

print(f'KL divergence from Q to P: {kl_divergence_reverse:.3f} nats')
```

```
KL divergence from P to Q: 0.590 nats
KL divergence from Q to P: 0.498 nats
```

Here, the result tells us two main things:

- ***Difference Between Distributions***: Both values quantify the difference between the distributions P and Q, but in different directions. A KL divergence of 0.590 nats from P to Q means that if you were to use distribution Q to approximate or encode distribution P, on average, you would need 0.590 extra nats per event to describe P compared to if you used P itself. Similarly, a divergence of 0.498 nats from Q to P indicates the inefficiency in using P to approximate Q.

- ***Asymmetry of KL Divergence***: The fact that the two values are not equal (0.590 vs. 0.498) illustrates the asymmetry of KL divergence. It highlights that the "distance" or divergence from P to Q is not the same as from Q to P. This asymmetry is a fundamental property of KL divergence, indicating that it is not a true metric distance since it does not satisfy the symmetry property (i.e., D (P || Q) ≠ D (Q || P)).

References:

[1] "Kullback–Leibler divergence." Wikipedia. Available from: https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence.

[2] “Kullback-Leibler Divergence - an overview." ScienceDirect. Available from: https://www.sciencedirect.com/topics/engineering/kullback-leibler-divergence.

3. Entropy:

- Entropy, in the context of information theory, is a measure of the uncertainty or unpredictability of a random variable's possible outcomes. It quantifies the average amount of information produced by a stochastic source of data. The concept of entropy was introduced by Claude Shannon in his seminal 1948 paper "A Mathematical Theory of Communication," and is sometimes referred to as Shannon entropy.

When considering a random variable X with possible outcomes x_1, x_2 , … , x_n, each with probability p(x_i), the entropy H(X) is defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i)$$

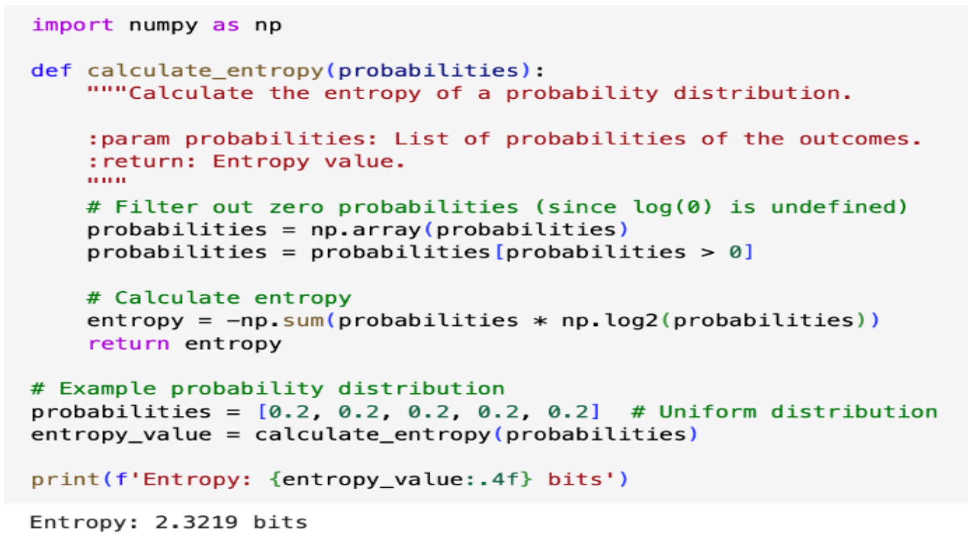
The base of the logarithm determines the unit of entropy. If the base is 2, the unit is bits (also known as "shannons"), if the base is e e (the natural logarithm), the unit is nats, and if the base is 10, the unit is dits or hartleys.

Entropy can be interpreted as the expected value of the self-information of a variable, which is the information content associated with the outcome of the variable. The self-information of an outcome (x_i) is defined as:

Entropy has several important properties:

1. Positivity: Entropy is always non-negative, H(X)≥0.
2. Maximum when Uniform: Entropy is maximized when all outcomes are equally likely (i.e., the distribution is uniform).
3. Additivity: For independent random variables X and *Y*, the entropy of the joint distribution is the sum of their individual entropies, H(X,Y)=H(X)+H(Y).

In machine learning and data science, entropy is often used as a criterion for feature selection and for building decision trees. It helps to choose the features that provide the most information gain, which is the reduction in entropy. [1][2]



```
import numpy as np

def calculate_entropy(probabilities):
    """Calculate the entropy of a probability distribution.

    :param probabilities: List of probabilities of the outcomes.
    :return: Entropy value.
    """
    # Filter out zero probabilities (since log(0) is undefined)
    probabilities = np.array(probabilities)
    probabilities = probabilities[probabilities > 0]

    # Calculate entropy
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy

# Example probability distribution
probabilities = [0.2, 0.2, 0.2, 0.2, 0.2]  # Uniform distribution
entropy_value = calculate_entropy(probabilities)

print(f'Entropy: {entropy_value:.4f} bits')
```

```
Entropy: 2.3219 bits
```

The output Entropy: 2.3219 bits indicates that the average amount of information, or uncertainty, in the given probability distribution is 2.3219 bits per event. This means that, on average, you would need 2.3219 bits to encode an event from the distribution, assuming an optimal encoding scheme based on the given probabilities.

References:
[1] "Entropy (information theory)." Wikipedia, The Free Encyclopedia. Available from: https://en.wikipedia.org/wiki/Entropy_%28information_theory%29
[2] Brownlee, J. (2020). "A Gentle Introduction to Information Entropy." Machine Learning Mastery. Available from: https://machinelearningmastery.com/what-is-information-entropy/

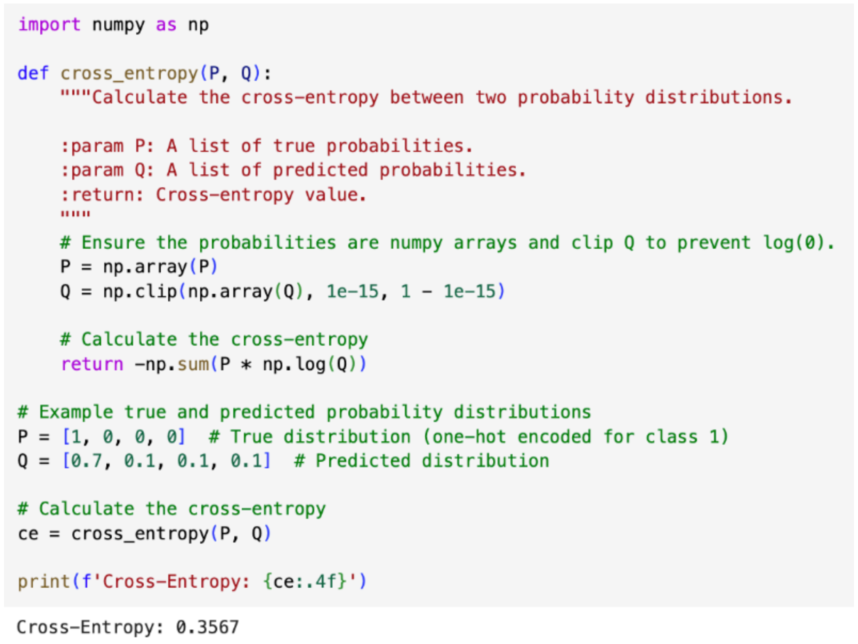
4. Cross-Entropy:

- Cross-entropy is a measure from the field of information theory that quantifies the difference between two probability distributions. It is often used in machine learning to define the loss function for classification problems, especially in models like logistic regression and neural networks.
- For two discrete probability distributions, P and Q, where P represents the true distribution of the data and Q represents the distribution estimated by the model, the cross-entropy of Q relative to P is defined as:

$$H(P,Q) = -\sum_i P(i)\log Q(i)$$

Where, P(i) is the true probability of outcome i, and Q(i) is the estimated probability of outcome i.

Cross-entropy is particularly useful because it penalizes predictions that are confident but wrong. The lower the cross-entropy, the better the model's predictions are in terms of accurately reflecting the true distribution of the data. In the context of training a model, the goal is to minimize the cross-entropy loss, which aligns with maximizing the likelihood of the true labels given the predictions made by the model. [1][2]



```python
import numpy as np

def cross_entropy(P, Q):
    """Calculate the cross-entropy between two probability distributions.

    :param P: A list of true probabilities.
    :param Q: A list of predicted probabilities.
    :return: Cross-entropy value.
    """
    # Ensure the probabilities are numpy arrays and clip Q to prevent log(0).
    P = np.array(P)
    Q = np.clip(np.array(Q), 1e-15, 1 - 1e-15)

    # Calculate the cross-entropy
    return -np.sum(P * np.log(Q))

# Example true and predicted probability distributions
P = [1, 0, 0, 0]  # True distribution (one-hot encoded for class 1)
Q = [0.7, 0.1, 0.1, 0.1]  # Predicted distribution

# Calculate the cross-entropy
ce = cross_entropy(P, Q)

print(f'Cross-Entropy: {ce:.4f}')
```

```
Cross-Entropy: 0.3567
```

The output Cross-Entropy: 0.3567 indicates that the model's predicted probability distribution is relatively close to the true distribution, with a cross-entropy loss of 0.3567, suggesting a good level of accuracy in the predictions.

References:

[1] "Cross-entropy." Wikipedia, The Free Encyclopedia. Available at: https://en.wikipedia.org/wiki/Cross-entropy
[2] Zhong, Yutao. "Cross-Entropy Loss Functions: Theoretical Analysis and Applications." arXiv, submitted on 14 Apr 2023, last revised 20 Jun 2023. Available at: https://arxiv.org/abs/2304.07288

5. Binary-Cross Entropy:

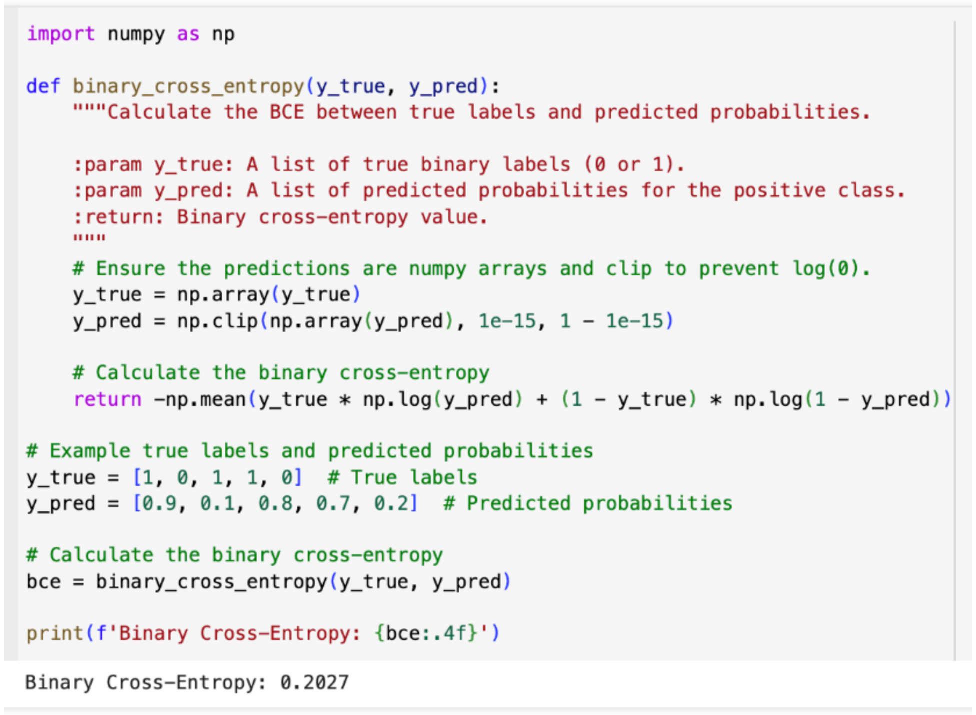
- Binary Cross-Entropy, also known as log loss, is a loss function used for binary classification problems. It measures the performance of a classification model whose output is a probability value between 0 and 1. Binary cross-entropy quantifies the difference between the true labels and the predicted probabilities of the positive class.

- For a set of observations where the true labels are denoted by y_i (with values 0 or 1) and the predicted probabilities by p_i, the binary cross-entropy is calculated as:

$$BCE = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Where,
- N is the number of observations,
- y_i is the true label of the i-th observation,
- p_i is the predicted probability of the i'th observation being of the positive class.

The binary cross-entropy loss is minimized when the predicted probabilities match the actual labels. A perfect model would have a binary cross-entropy loss of 0. [1][2]



```
import numpy as np

def binary_cross_entropy(y_true, y_pred):
    """Calculate the BCE between true labels and predicted probabilities.

    :param y_true: A list of true binary labels (0 or 1).
    :param y_pred: A list of predicted probabilities for the positive class.
    :return: Binary cross-entropy value.
    """
    # Ensure the predictions are numpy arrays and clip to prevent log(0).
    y_true = np.array(y_true)
    y_pred = np.clip(np.array(y_pred), 1e-15, 1 - 1e-15)

    # Calculate the binary cross-entropy
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

# Example true labels and predicted probabilities
y_true = [1, 0, 1, 1, 0]  # True labels
y_pred = [0.9, 0.1, 0.8, 0.7, 0.2]  # Predicted probabilities

# Calculate the binary cross-entropy
bce = binary_cross_entropy(y_true, y_pred)

print(f'Binary Cross-Entropy: {bce:.4f}')
```

```
Binary Cross-Entropy: 0.2027
```

- The output Binary Cross-Entropy: 0.2027 indicates that the average loss per observation for your binary classification model is 0.2027. This value represents how well the model's predicted probabilities align with the actual class labels; the closer this value is to 0, the better the model's predictions are. A binary cross-entropy of 0.2027 suggests that the model is performing reasonably well, but there may still be room for improvement.

References:

[1] "Cross entropy." Wikipedia, The Free Encyclopedia. Available at: https://en.wikipedia.org/wiki/Cross_entropy.
[2] Brownlee, J. (2019). "How to Implement Binary Cross-Entropy Loss in Python." Machine Learning Mastery. Available at: https://machinelearningmastery.com/how-to-implement-binary-cross-entropy-loss-in-python/