Entropy, Cross-Entropy and KL-Divergence Brief Review

Lizi Chen

Motivation: In Machine Learning, cross-entropy is commonly used as a cost function when training classifiers.

1 Background

Laws of nature states that things goes from in-order state to dis-order state. The number of micro-states of an in-order state is much greater than the one of a dis-order state. The definition of Entropy in Physics:

$$S = \kappa \ln \Omega$$

κ is Boltzmann constant, Ω is the number of micro-states.

The Mathematical Theory of Communication, Claude Shannon states that: To transmit one bit of information (from sender to recipient), means to reduce the recipient's uncertainty by a factor of 2 (assume that probability of each event is the same).

Example: A event of two equally likely possible options; A is 50% chance and the other B is also 50% chance, given that the sender told the recipient that the next event will be B, the sender just reduce the recipient's uncertainty for the event by 2.

1.1 Compute the actual bit of information:

N is uncertainty reduction factor, the number of possible options.
When $N = 2$, a binary prediction situation, transmitting $\log_2 N = \log_2 2 = 1$ bit is good enough.
When $N = 8$, effective bit of data transmission requires $\log_2 8 = 3$ bits.
(Again, all options have the same possibility.)

1.2 When the possibilities are not equally likely:

Say, A is 75% chance and B is 25% chance. When we are told B is going to occur, the uncertainty reduction factor becomes

$$\frac{1}{25\%} = 4$$

Therefore, we have the actual bit $= \log_2 4 = 2$ bits for transmitting B.
Similarly, for transmitting A, we need $\log_2(1/0.75) = 0.41$ bits of information.

Observation: The event with higher probability; when transmitting, requires less bits. On the opposite, the one with lower probability requires more bits. (This is efficient.)

2 Entropy:

In the example above; transmitting A with P_A probability and B with P_B probability, each requires b_A bits and b_B bits of information. Thus, the entropy is defined as:

$$E = P_A \times b_A + P_B \times b_B$$

which is the expected amount of bits of information for a transmission from sender to recipient. In the example, we have $75\% \times 0.41 + 25\% \times 2 = 0.81$ bits.

In general, entropy is defined as:

$$H(p) = -\sum_i p_i \times \log_2(p_i) \tag{1}$$

Note: Entropy is always non-negative.

Summarize 'Lecture Notes on Information Theory' [9] todo

3 Cross-Entropy:

In short, Cross-Entropy is the average weighted length of a message. (i.e., number of bits).

For example: Four different messages; A, B, C, D are sending between a sender and a recipient, we need to know the encoding for these messages. Optimally, we need to know the best encoding so that we use less 0's and 1's for the message representation. Assume all four messages have the same probability; $\frac{1}{4}$, we use the Table 1. Our most optimal length of message encoding is $2 \times \frac{1}{4} \times 4 = 2$ bits.

Message	A	B	C	D
Code	00	01	10	11

Table 1: Same probability messages

Now assume we have different probabilities for the four messages, as shown in Table 2, our expected encoding length is $1_A \times \frac{1}{2} + 2_B \times \frac{1}{4} + 3_C \times \frac{1}{8} + 3_D \times \frac{1}{8} = 1.75$ bits, better than the previous 2 bits for the same-length encoding.

Message	A	B	C	D
Probability	1/2	1/4	1/8	1/8
Code	0	10	110	111

Table 2: Different probability messages

3.1 Encoding Space

In the example that the code for A is 0, and thereafter the rest codes cannot start with 0. This way we can decode unique series of code. For example, if we have A as 0, B as 01, and C as 1 then when the recipient receive 0101, it can be interpreted as ACB, or BB, or BAC. Such coding is wrong. We need follow the **Prefix Codes** method for different code length to avoid ambiguity. To visualize the encoding space, we have Figure **??**. When we choose 1 for the second bit, $\frac{1}{4}$ encoding space will no longer be use-able.

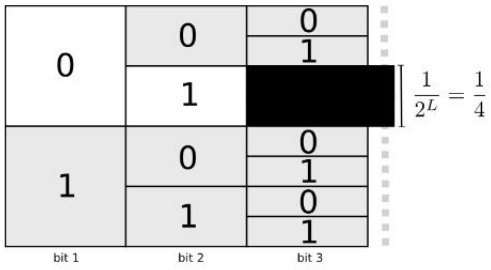


Figure 1: Encoding Space Example

3.2 Optimal Encoding

In general, a code of length $\mathcal{L}$ results in lost of $\frac{1}{2^{\mathcal{L}}}$ encoding space.

Therefore, for limited length of a code, we want to optimize the use of encoding space. That is, (1) use code that is as short as possible, (2) use as many encoding space as possible.

Optimal Encoding can be reached when we assign encoding space loss to a message according to its probability $P(X)$, where X is the message, $P(X)$ means its **prior probability of showing-up among all messages**.

Let $\mathcal{L}(X)$ be the length of X, we have $\frac{1}{2^{\mathcal{L}(X)}} = P(X)$, which is $\mathcal{L}(X) = \log_2 \frac{1}{P(X)}$. Therefore, for a probability distribution P, we have the expected weighted average code length:

$$H(P) = \sum_{x \in X} \underbrace{p(x)}_{prob} \underbrace{\log_2 \frac{1}{p(x)}}_{\mathcal{L}(x)}, \text{ where } H(P) \text{ is the Entropy.} \tag{2}$$

$H(P)$; the entropy, means the average length code using optimal encoding under the P distribution.

3.3 Not Using Optimal Encoding

Now we have messages in probability distribution Q; however, we choose to use another distribution P. Thus, the expected average length of code is:

$$H(Q||P) = H_P(Q) = \sum_{x \in X} q(x) \log_2 \frac{1}{p(x)} \tag{3}$$

$H(Q||P)$ or $H_P(Q)$ is the **Cross Entropy** for Q on P. It measures the average length of code for using P distribution on Q message distribution.

Example: We have Table 3. The third row; Probability (Q), is the true probability for the message to show up. The fifth row; P(X), is the distribution we use according to the code. It is obvious to notice that we use longer code for message that appears more frequently; D is 111 but has 40% to show up. Such encoding is not a good choice, since we will be more likely to send more bits.

$$H(Q||P) = 0.1 \times \log_2 1/(1/2) + 0.2 \times \log_2 1/(1/4) \cdots = 2.6$$

This means on average a sender has to send 2.6 bits. However, if we set Q = P and follow Table 2, we can have $H = 1.75$ bits, which is much better than 2.6 bits for such set-up.

Message	A	B	C	D
Code	0	10	110	111
Probability (Q)	10%	20%	30%	40%
Code Length	1	2	3	3
P(X)	1/2	1/4	1/8	1/8

Table 3: Different probability messages

When P is 'perfect'; meaning when $P = Q$, Cross Entropy equals to Entropy. Otherwise, Cross Entropy is greater than Entropy. Such difference between Cross Entropy and Entropy is called **Relative Entropy**, or more commonly called **KullbackLeibler (KL) Divergence**, as reviewed in the next section.

4 Kullback-Leibler (KL) Divergence (Relative Entropy)

Two probability distributions; Q and P, the KL-divergence measures the similarity of Q and P. If $KL\_Divergence(Q||P) = 0$, the two distributions are equal.

For discrete probability distributions:

$$D_{KL}(Q||P) = \sum_i Q(i) \log \frac{Q(i)}{P(i)} \tag{4}$$

For continuous probability distributions:

$$D_{KL}(Q||P) = \int_{-\infty}^{\infty} q(x) \log \frac{q(x)}{p(x)} dx \tag{5}$$

Also, as mentioned previously, the difference between Cross Entropy and Entropy is KL Divergence. Therefore, we can have:

$$D_{KL}(Q||P) = H(Q||P) - H(Q) \tag{6}$$

Figure 2 illustrates such relationship.

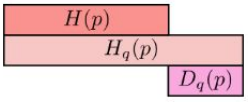


Figure 2: KL Divergence, Entropy and Cross Entropy

4.1 Proof that $\mathbf{D_{KL}(Q||P) \geq 0}$ always true, using Convex Function

Review the definition of **Convex Function**:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2) \tag{7}$$

In general:

$$f(\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n) \leq \theta_1 f(x_1) + \theta_2 f(x_2) + \cdots + \theta_n f(x_n) \tag{8}$$

, where

$$0 \leq \theta_i \leq 1 \tag{9}$$

, and

$$\theta_1 + \theta_2 + \cdots + \theta_n = 1 \tag{10}$$

Prove $\mathbf{D_{KL}(Q||P) \geq 0}$:

$$\begin{aligned} D_{KL}(Q||P) &= \sum_i Q(i) \log \frac{Q(i)}{P(i)} \\ &= \sum_i Q(i) \Big[-\log \frac{P(i)}{Q(i)} \Big] \\ &= -\sum_i Q(i) \log \frac{P(i)}{Q(i)} \end{aligned} \tag{11}$$

As $0 \leq Q(i) \leq 1$, $\sum_i Q(i) = 1$; similar to θ_i in Eq. 9 and Eq. 10, we can treat $Q(i)$ as θ_i, and treat $-\log \frac{P(i)}{Q(i)}$ as $f(x)$, thus we have:

$$D_{KL}(Q||P) = -\Big[Q(i_1) \log \frac{P(i_1)}{Q(i_1)} + \cdots + Q(i_n) \log \frac{P(i_n)}{Q(i_n)} \Big]$$

, where according to definition of Convex Function in Eq. 8:

$$Q(i_1) \Big[-\log \frac{P(i_1)}{Q(i_1)} \Big] + \cdots + Q(i_n) \Big[-\log \frac{P(i_n)}{Q(i_n)} \Big] \geq -\log \Big[Q(i_1) \frac{P(i_1)}{Q(i_1)} + \cdots + Q(i_n) \frac{P(i_n)}{Q(i_n)} \Big]$$

, thus we have:

$$D_{KL}(Q||P) \geq -\log \Big[Q(i_1) \frac{P(i_1)}{Q(i_1)} + \cdots + Q(i_n) \frac{P(i_n)}{Q(i_n)} \Big]$$

As $Q(i)$ in $Q(i)\frac{P(i)}{Q(i)}$ cancel each other, we have:

$$\begin{aligned} D_{KL}(Q||P) &\geq -\log \Big[P(i_1) + \cdots + P(i_n) \Big] \\ &= -\log(1) \\ &= 0 \quad \square \end{aligned} \tag{12}$$

5 Cross-Entropy as Cost Function in Machine Learning

Cross-Entropy is widely used in machine learning as a cost function for classifier. In a supervised learning set-up, after a classifier, the true class distribution is Q; commonly represented as y_i, and the predicted class distribution is P. The Cross-Entropy Loss is defined as:

$$\text{Loss} = H(Q, P) = -\sum_{i}^{N} q_{y_i} \log p_{x_i}$$

, and the objective is to optimize $\arg\min_{P(X)} Loss$.

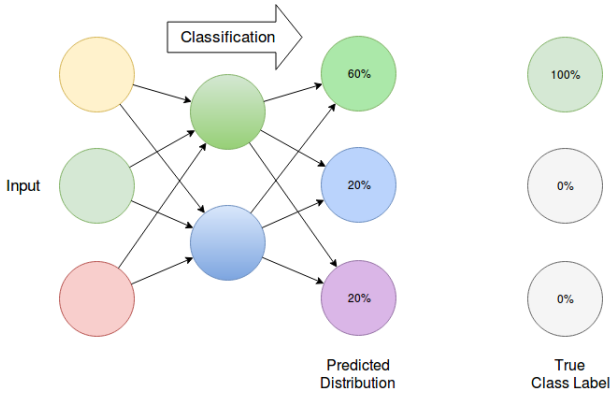
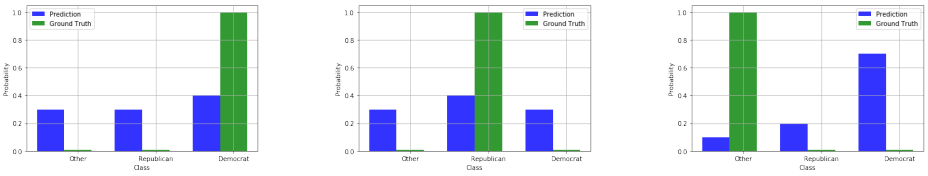


Figure 3: * The colors here do not represent any value.

5.1 Cost Function: Problem of using *Classification Error* as cost function:[2]

Classification Error: $= \frac{count\ of\ error\ items}{count\ of\ all\ items}$, such cost function does not care about the probability distribution of each prediction:
For example: The blue bar represents predicted probability, the green bar represents ground truth.

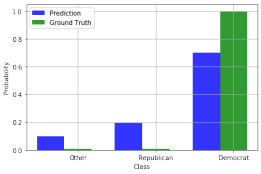


(a) Correct (b) Correct (c) Wrong

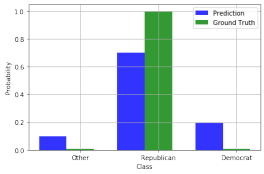
Figure 4: a and b are correctly predicted items, $Classification\_Error = \frac{1}{3}$

Note:
In Figure 4, Prediction Probability of (a) is [0.3, 0.3, 0.4], (b) is [0.3, 0.4, 0.3], (c) is [0.1, 0.2, 0.7].
In Figure 5, (a) is [0.1, 0.2, 0.7], (b) is [0.1, 0.7, 0.2], (c) is [0.3, 0.4, 0.3].

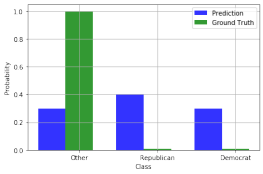
Even though model from Figure 4 is not as promising as the one from Figure 5, their *Classification Errors* are the same, which indicates drawbacks of not being able to describe



(a) Correct



(b) Correct



(c) Wrong

Figure 5: a and b are correctly predicted items; but with relatively higher prediction probability. However, *Classification_Error* $= \frac{1}{3}$; same as Example shown in Figure 4.

the model accuracy correctly. Instead, if we us Cross Entropy, we take into account of the prediction probability.

For example:
In Figure 4, the cross-entropy of prediction (a) is $-\{(\ln 0.3)\times 0+(\ln 0.3)\times 0+(\ln 0.4)\times 1\} = -\ln 0.4$, the average cross-entropy error is $\frac{-\ln 0.4+\ln 0.4+\ln 0.1}{3} = 1.38$, yet the model from Figure 5 has an average cross-entropy error 0.64. As $0.64 < 1.38$; therefore, the second model has lower entropy, is a better prediction(classification) model.

5.2 Cost Function: Problem of using *Mean Squared Error* as cost function: Non-Convexity

The biggest problem of using Classification Error as a cost function; as stated in the previous section, is not considering the probability of a prediction. Now if we use Mean Squared Error as a cost function, the prediction loss from Figure 4 (a) is:

$$(0.3-0)^2+(0.3-0)^2+(0.4-1)^2 = 0.54$$

The average loss from the model in Figure 4 is $\frac{0.54+0.54+1.34}{3} = 0.81$, the one in Figure 5 is 0.34. Even though such difference in loss helps distinguish two models, there are situations where Mean Squared Error is not well suited for cost function.

As shown in a Coursera video by Andrew Ng [5], in the set-up of a **Logistic Regression**[1], the prediction function h_θ is non-linear, i.e.,

$$h_\theta = \frac{1}{1+e^{-\theta^T x}} \tag{13}$$

, when using such prediction function in the 'Mean Squared Error' cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost\_function}(h_\theta(x^i), y^i) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(h_\theta(x^i)-y^i)^2$$

Substitute non-linear Logistic equation 13, we have:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(\frac{1}{1+e^{-\theta^T x}}-y^i)^2 \tag{14}$$

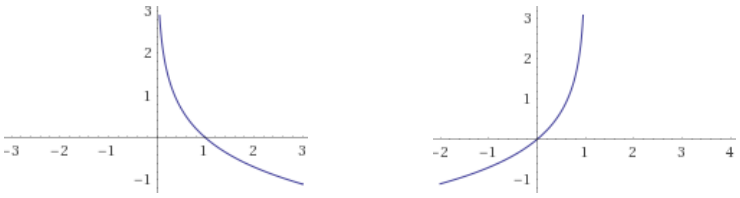
[1]Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic (sigmoid) function to return a probability value which can then be mapped to two or more discrete classes.

, is non-convex. *(Refer to the Convexity tutorial by Po-Shen Loh)*

Instead, we use an Entropy-alike Cost Function as shown below 15, it looks like the formula of Entropy as shown in equation 1.

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & if\ y = 1 \\ -\log(1 - h_\theta(x)) & if\ y = 0 \end{cases} \tag{15}$$

, where $h_\theta(x)$ is the prediction probability in range $[0, 1]$. The plots of each are shown in Figure 6 below.



(a) When y=1 (b) When y=0

Figure 6: Cost($h_\theta(x)$), The horizontal axis $h_\theta(x) \in [0, 1]$

The equation 15 is the primitive type of Logistic Regression cost function. To make it easier for gradient descent, we combine the two lines of equation into one, as shown in equation 16 below.

$$Cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x)) \tag{16}$$

And the overall cost can be measured by summing over all data points:

$$\begin{aligned} J(\theta) &= \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^i), y^i) \\ &= \frac{1}{m} \sum_{i=1}^{m} \Big[-y^i \log(h_\theta(x^i)) - (1 - y^i) \log(1 - h_\theta(x^i)) \Big] \\ &= -\frac{1}{m} \sum_{i=1}^{m} \Big[y^i \log(h_\theta(x^i)) + (1 - y^i) \log(1 - h_\theta(x^i)) \Big] \end{aligned} \tag{17}$$

Substitute Sigmoid Eq.13, we can simplify the cost function as:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \Big[y_i \theta x^i - \log(1 + e^{\theta x^i}) \Big] \tag{18}$$

Vectorized form of Eq.17:

$$J(\theta) = -\frac{1}{m} \Big[Y^T \log\Big(Sig(\theta^T X)\Big) + (1 - Y)^T \log\Big(1 - Sig(\theta^T X)\Big) \Big] \tag{19}$$

, where

$$Sig(\theta^T X) = \frac{1}{1 + e^{-\theta^T X}}$$

Understanding the difficulty of training deep feedforward neural networks to-read

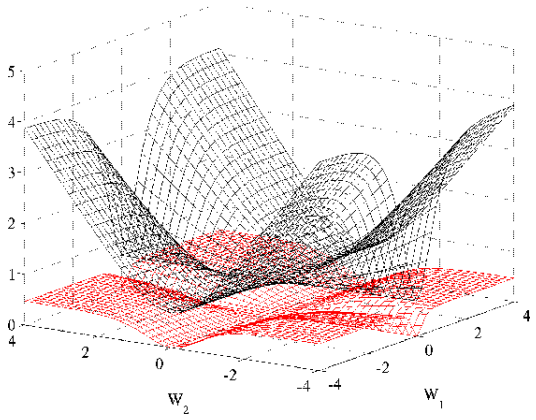


Figure 7: Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers. Source: [4]

Section 4.1 Effect of the Cost Function Figure 7 shows that squared error cost function are relatively 'flat' than using cross-entropy loss function, which implies that cross-entropy are more likely to have larger gradient so that converge faster.

Further Reading: Cross-Entropy vs. Squared Error Training: A theoretical and experimental comparison, Pavel Golik.

Prove Convexity

5.3 Gradient Descent on Logistic Regression Cost Function

To find the optimal parameters for a Logistic Regression model θ, we need to find the minimum of $J(\theta)$ from equation 17. One optimization algorithm is call Gradient Descent.

```
# Gradient Descent basic idea:
repeat until convergence{
   # Simultaneously update all j.
   θ_j := θ_j − α · ∂J(θ)/∂θ_j  # α is the learning rate.
}
```

The $\frac{\partial J(\theta)}{\partial \theta_j}$ can be represented as:

$$\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \quad \text{for } [j := 0, \cdots, n], \ (n = \#\text{features}) \tag{20}$$

Detailed derivation process, see Appendix 10.

For the simplified Logistic Cost function, Eq.18:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\Big[\underbrace{y_i\theta x^i}_{A} - \underbrace{\log(1+e^{\theta x^i})}_{B}\Big] \tag{21}$$

where,

$$\text{A:} \quad \rightarrow \quad \frac{d}{d\theta_j} y_i \theta_j x_j^i = y_i \cdot x_j^i$$

$$\text{B:} \quad \rightarrow \quad \frac{d}{d\theta_j}\log(1+e^{\theta x_j^i}) = \frac{x_j^i exp[\theta_j x_j^i]}{1+exp[\theta_j x_j^i]}$$

Note again that the index j refers to the j^{th} feature.

Sample Python Code 1

There are many advanced numerical optimization algorithms other than Gradient Descent: Conjugate Gradient, BFGS, and L-BFGS, as mentioned in Andrew Ng's video. [5]

Learning Strategies Note

5.4 MLE on Logistic Regression Cost Function

A probabilistic interpretation:

$$P(y|x) = \begin{cases} h(\theta^T X) & for\ y = 1 \\ 1 - h(\theta^T X) & for\ y = 0 \end{cases} \tag{22}$$

More compact form:

$$P(y|x) = h(y \cdot \theta^T x) \tag{23}$$

Interpretation: The probability of getting y from x is based on the prediction function h on $\theta^T x$.

Recall the assumption of independence of $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$. We have:

$$P(y_1, y_2, \cdots, y_n) = \prod_{n=1}^{N} P(y_n|x_n) \tag{24}$$

To get the best parameters, we maximize $P(y_1, y_2, \cdots, y_n)$:

$$\begin{aligned} \text{maximize } P(y_1, y_2, \cdots, y_n) &= \text{maximize } \prod_{n=1}^{N} P(y_n|x_n) \\ &\Leftrightarrow \text{maximize } \ln\Big(\prod_{n=1}^{N} P(y_n|x_n)\Big) \\ &= \text{maximize } \sum_{n=1}^{N} \ln P(y_n|x_n) \\ &\Leftrightarrow \text{minimize } -\frac{1}{N}\sum_{n=1}^{N} \ln P(y_n|x_n) \\ &= \text{minimize } -\frac{1}{N}\sum_{n=1}^{N} \ln \frac{1}{h(y_i \cdot \theta^T x_i)} \end{aligned} \tag{25}$$

6 Jensen-Shannon Divergence

7 Joint Entropy

8 Conditional Entropy

Given r.v. X, the uncertainty of r.v. Y is defined as:

$$\begin{aligned} H(Y|X) &= \sum_x p(x) H(Y|X=x) \\ &= -\sum_x p(x) \sum_y p(y|x) \log p(y|x) \\ &= -\sum_x \sum_y p(x,y) \log p(y|x) \\ &= -\sum_{x,y} p(x,y) \log p(y|x) \end{aligned} \tag{26}$$

9 Gini Impurity

Gini Impurity is used in **decision tree** algorithms. It measures how impure/complex a set of data is. The notion and formula looks very similar to Entropy.

Given a set of data (A, B, B, C, D, D), randomly select a data and guess what it is. In another example, if the data set is all A's, then randomly select a data and guessing it's A, is 100% probability. According to the Gini formula shown in Equation 27, the Gini Impurity score; in this case, is 1 - 1 = 0.

As shown in Equation 1, Entropy is defined as:

$$H(p) = -\sum_i p_i \times \log_2(p_i)$$

and Gini Impurity is defined as:

$$\begin{aligned} Gini &= 1 - \sum_i p_i^2 \\ &= \sum_i p_i \times p_{-i} \end{aligned} \tag{27}$$

We can use Taylor Equation to approximate Gini Impurity equation from Entropy:
- First-order expand $f(x) = -\ln x$ at $x_0 = 1$, ignore higher-order infinitesmall, we have $f(x) \approx 1 - x$
- Thus: $H(p) = -\sum_i p_i \times \log_2(p_i) \approx -\sum_i p_i(1 - p_i)$

add taylor series

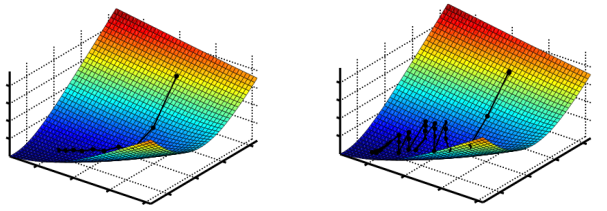
Both Entropy and Gini Impurity formulas are weighted sum over a probability distribution.

Sample Python code available at Appendix 2

10 Appendix

Listing 1: Logistic Regression Gradient Descent

```
def update_weights(features, labels, weights, alpha):
    '''
    Vectorized Gradient Descent

    Features:(200, 3)
    Labels: (200, 1)
    Weights:(3, 1)
    '''
    N = len(features)

    #1 - Get Predictions
    # predictions: (200, 1)
    predictions = predict(features, weights)

    #2 Transpose features from (200, 3) to (3, 200)
    # So we can multiply with the (200,1) cost matrix.
    # Returns a (3,1) matrix holding 3 partial derivatives --
    # one for each feature -- representing the aggregate
    # slope of the cost function across all observations
    loss = predictions - labels
    gradient = np.dot(features.T, loss)

    #3 Take the average cost derivative for each feature
    gradient /= N

    #4 - Multiply the gradient by learning rate
    gradient *= alpha

    #5 - Subtract from our weights to minimize cost
    weights -= gradient

    return weights
```



(a) GD, $\alpha = 6$, 10 steps, N=10

(b) SGD, $\alpha = 2$, 30 steps

Figure 8: GD vs. SGD. Source: Malk Magdon-Ismail

Listing 2: Calculate Gini Impurity of an array data

```
def GiniImpurity(data):
    total=len(data)
    counts={}
```

```
    # set the count of an item to 0 if not present in the dictionary.
    # increment count of item by 1 if appears more time.
    for item in data:
        counts.setdefault(item,0)
        counts[item]+=1

    impurity=0
    for j in data:
        f1 = float(counts[j]) / total
        for k in data:
            if j==k:
                continue # skip
            f2 = float(counts[k]) / total
            impurity += f1*f2
    return impurity

# run:
GiniImpurity(['b','a','c'])
# output:
# loop 1, j = b, impurity = 1/3 × 1/3 + 1/3 × 1/3
# loop 2, j = a, impurity = impurity + 1/3 × 1/3 + 1/3 × 1/3
# loop 3, j = c, impurity = impurity + 1/3 × 1/3 + 1/3 × 1/3
# impurity = 6/9 ≈ 0.6666666666666667
0.6666666666666667
```

Derivation of Logistic Cost Function Step-by-step

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \left(\log(h_\theta \left(x^{(i)} \right) \right) + (1 - y^{(i)}) \left(\log(1 - h_\theta \left(x^{(i)} \right) \right) \right] \tag{28}$$

$$\underset{\text{linearity}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\partial}{\partial \theta_j} \log \left(h_\theta \left(x^{(i)} \right) \right) + (1 - y^{(i)}) \frac{\partial}{\partial \theta_j} \left(\log(1 - h_\theta \left(x^{(i)} \right) \right) \right] \tag{29}$$

$$\underset{\text{chain rule}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\frac{\partial}{\partial \theta_j} (h_\theta \left(x^{(i)} \right)}{h_\theta \left(x^{(i)} \right)} + (1 - y^{(i)}) \frac{\frac{\partial}{\partial \theta_j} \left(1 - h_\theta \left(x^{(i)} \right) \right)}{1 - h_\theta \left(x^{(i)} \right)} \right] \tag{30}$$

$$\underset{h_\theta(x) = \sigma\left(\theta^\top x\right)}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\frac{\partial}{\partial \theta_j} \sigma \left(\theta^\top x^{(i)} \right)}{h_\theta \left(x^{(i)} \right)} + (1 - y^{(i)}) \frac{\frac{\partial}{\partial \theta_j} \left(1 - \sigma \left(\theta^\top x^{(i)} \right) \right)}{1 - h_\theta \left(x^{(i)} \right)} \right] \tag{31}$$

$$\underset{\sigma'}{=} \frac{-1}{m} \sum_{i=1}^{m} [y^{(i)} \frac{\sigma(\theta^\top x^{(i)})(1 - \sigma(\theta^\top x^{(i)})) \frac{\partial}{\partial \theta_j} (\theta^\top x^{(i)})}{h_\theta(x^{(i)})} - \tag{32}$$

$$(1 - y^{(i)}) \frac{\sigma \left(\theta^\top x^{(i)} \right) \left(1 - \sigma \left(\theta^\top x^{(i)} \right) \right) \frac{\partial}{\partial \theta_j} \left(\theta^\top x^{(i)} \right)}{1 - h_\theta \left(x^{(i)} \right)}] \tag{33}$$

$$\underset{\sigma\left(\theta^\top x\right) = h_\theta(x)}{=} \frac{-1}{m} \sum_{i=1}^{m} [y^{(i)} \frac{h_\theta \left(x^{(i)} \right) \left(1 - h_\theta \left(x^{(i)} \right) \right) \frac{\partial}{\partial \theta_j} \left(\theta^\top x^{(i)} \right)}{h_\theta \left(x^{(i)} \right)} - \tag{34}$$

$$(1 - y^{(i)}) \frac{h_\theta \left(x^{(i)} \right) \left(1 - h_\theta \left(x^{(i)} \right) \right) \frac{\partial}{\partial \theta_j} \left(\theta^\top x^{(i)} \right)}{1 - h_\theta \left(x^{(i)} \right)}] \tag{35}$$

$$\underset{\frac{\partial}{\partial \theta_j} \left(\theta^\top x^{(i)} \right) = x_j^{(i)}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \left(1 - h_\theta \left(x^{(i)} \right) \right) x_j^{(i)} - \left(1 - y^i \right) h_\theta \left(x^{(i)} \right) x_j^{(i)} \right] \tag{36}$$

$$\underset{\text{distribute}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^i - y^i h_\theta \left(x^{(i)} \right) - h_\theta \left(x^{(i)} \right) + y^{(i)} h_\theta \left(x^{(i)} \right) \right] x_j^{(i)} \tag{37}$$

$$\underset{\text{cancel}}{=} \frac{-1}{m} \sum_{i=1}^{m} [y^{(i)} - h_\theta \left(x^{(i)} \right)] \, x_j^{(i)} \tag{38}$$

$$= \frac{1}{m} \sum_{i=1}^{m} [h_\theta \left(x^{(i)} \right) - y^{(i)}] \, x_j^{(i)} \tag{39}$$

References

[1] "Journey into information theory, Khan Academy," available at https://www.khanacademy.org/computing/computer-science/informationtheory.

[2] J. D.McCaffrey, "Why you should use cross-entropy error instead of ..." available at https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural

[3] A. Geron, "Short introduction to entropy, cross-entropy, and kl-divergence," available at https://www.youtube.com/watch?v=ErfnhcEV1O8&t=393s.

[4] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[5] A. Ng, "Cost function, machine learning, stanford university," video Available at https://www.coursera.org/lecture/machine-learning/cost-function-1XG8G.

[6] S. L. Paul Penfield, "MIT 6.050j information and entropy, spring 2008," available at https://www.youtube.com/playlist?list=PLDDE03B3BDCA1D9B1.

[7] S. Raschka, "What is softmax regression and how is it related to logistic regression?" available at https://www.kdnuggets.com/2016/07/softmax-regression-related-logistic-regression.html.

[8] T. Segaran, *Programming Collective Intelligence*.

[9] Y. W. Y. Polyanskiy, "Lecture note on information theory," pdf file available at http://www.stat.yale.edu/~yw562/teaching/itlectures.pdf.