Machine Learning

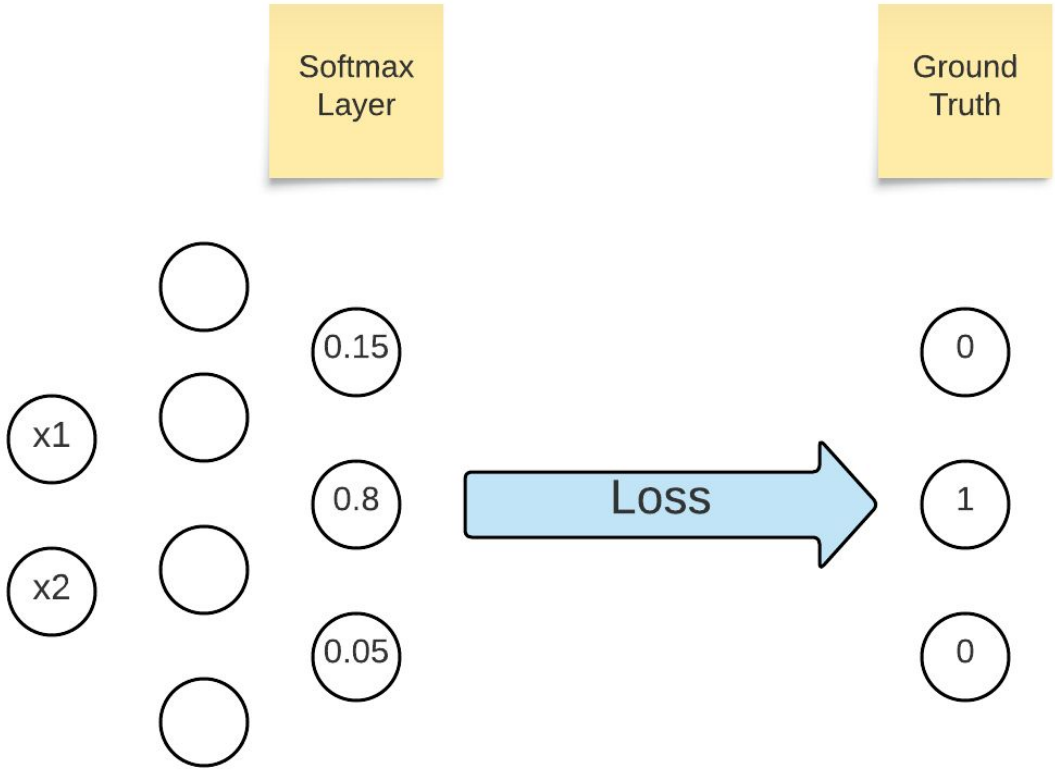
Cross Entropy

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

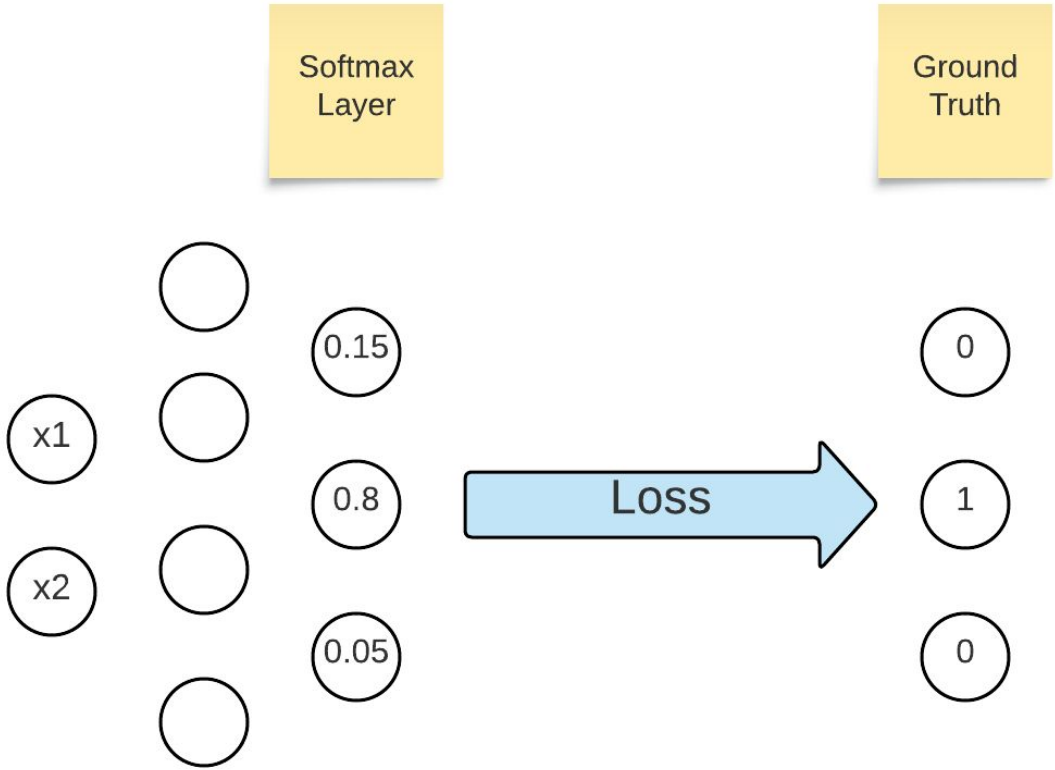
The Loss

- Now we have our predicted probability distribution
- We also have the GT probability distribution
- We need a loss function that will penalize far predictions!
- There are several ways to quantify 2 distribution : prediction vs GT



The Loss: MSE

- One can think of MSE
- We helighted how MSE even for the binary classification is not a good choice!
  - It may work but
    - Slower convergence?
    - Sub-optimal results?



Intuition

$$logloss(p,t) = -tlog(p) - (1-t)log(1-p)$$

- Recall our log-loss for binary classification
  - **Only one term** of these will be added: the one with the class
  - So intuitive extension for multi-class is only add the target class out of C classes
- This can be written as:
  - t = [0, 1, 0]
  - p = [0.15, 0.8, 0.05]
  - loss = -log(0.8)

$$loss(t,p) = -\sum_{i=1}^{C} t_i \log(p_i)$$

- For binary cases
  - t = [0, 1]
  - p = [0.3, 0.7]
  - loss = -log(0.7)

t = [1, 0]
p = [0.6, 0.4]
loss = -log(0.6)

Cross Entropy

- The previous equation is actually called cross-entropy
- There are some interesting math concepts behind it
- Interestingly the vector t for ground truth can be a real probability distribution, not just one hot encoding (add single term)
  - Try to think intuitively about it, e.g. t = [0.1, 0.1, 0.8]
  - It belongs to category of functions that quantify the difference between 2 distributions
- Let’s introduce the math behind that

$$loss(t,p) = -\sum_{i=1}^{C} t_i \log(p_i)$$

Img src

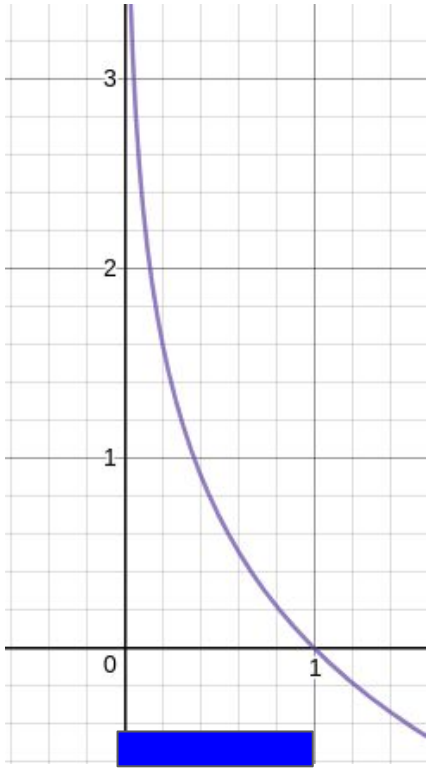
Surprise!

- To what degree you will be **surprised** from the following events!
- You live in a desert where it is sunny almost every day. Your friend guess tomorrow is sunny
- You selected a random number between 1 to 1000. You asked your friend to guess your number. He guessed it correctly!

- High probability ⇒ Low Surprise ⇒ Low Entropy
  - Weather is most probably sunny
- Low probability ⇒ High Surprise ⇒ High Entropy
  - Chance is 1/1000. Very low probability to guess correctly

Quantifying the Surprise!

- It seems the relationship between the surprise and probability is being **inverse**
  - We need something that goes toward zero if probability goes to one / And the opposite
- We already learned in log loss that **-log(p)** is a good candidate
  - So given an event of probability p, it will surprise us with -log(p)
- Entropy is that surprise concept!



-ln(p)

Entropy of a Distribution

- What if we have a categorical distribution of N classes: x1, x2, …xn
  - Sun, Cloud, Wind, Rain
- Entropy is the **expected** surprise of a distribution p of N classes?
- Optional video

$$E[X] = \sum_{i=1}^{n} p(x_i) \cdot x_i$$

$$H(X) = E[-\log(p(X))]$$

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

Cross Entropy

- Back to the loss function: we need to **quantify the difference** between 2 distributions: the predicted distribution and the ground truth distribution
  - In different words: **relative entropy** between two probability distributions over N classes
- Entropy can be extended to give us the expected surprise of the predicted distribution relative to the ground truth distribution
- In other words,
  - Apply the -log() on the predicted probability
  - Use the ground truth probability to weight the summation
    - With one-hot encoding, we use a single entropy at the target
- *Observe: equation is not symmetric*
- *Observe: CE is not a distance metric*

$$H(y,p) = -\sum_{i=1}^{C} y_i \log(p_i)$$

		CE
Ground Truth p	0.1, 0.7, 0.2	
Prediction q	0.1, 0.7, 0.2	0.801
Prediction q	0.1, 0.6, 0.3	0.828
Prediction q	0.1, 0.5, 0.4	0.898
Prediction q	0.0, 0.5, 0.5	4.077
Prediction q	0.0, 0.4, 0.6	4.197
Prediction q	0.0, 0.2, 0.8	4.625
Prediction q	0.0, 0.1, 0.9	5.086
Prediction q	0.0, 0.01, 0.99	6.679
Prediction q	0.0, 0.0, 1	27.63
Prediction q	1.0, 0.0, 0	31.08
Prediction q	0.5, 0.0, 0.5	24.38

The bigger the difference between p and q, the higher the entropy

Also it is differentiable

As a result, this is an acceptable loss function to help the network find Ws that minimize CE

Implementation

```
def cross_entropy(p, q):
    # Adding a small constant for
    # numerical stability as log(0) is undefined
    ce =-np.sum(p * np.log(q + 1e-15))

    return ce
```

$$H(p,q) = -\sum_{i=1}^{N} p_i \log(q_i)$$

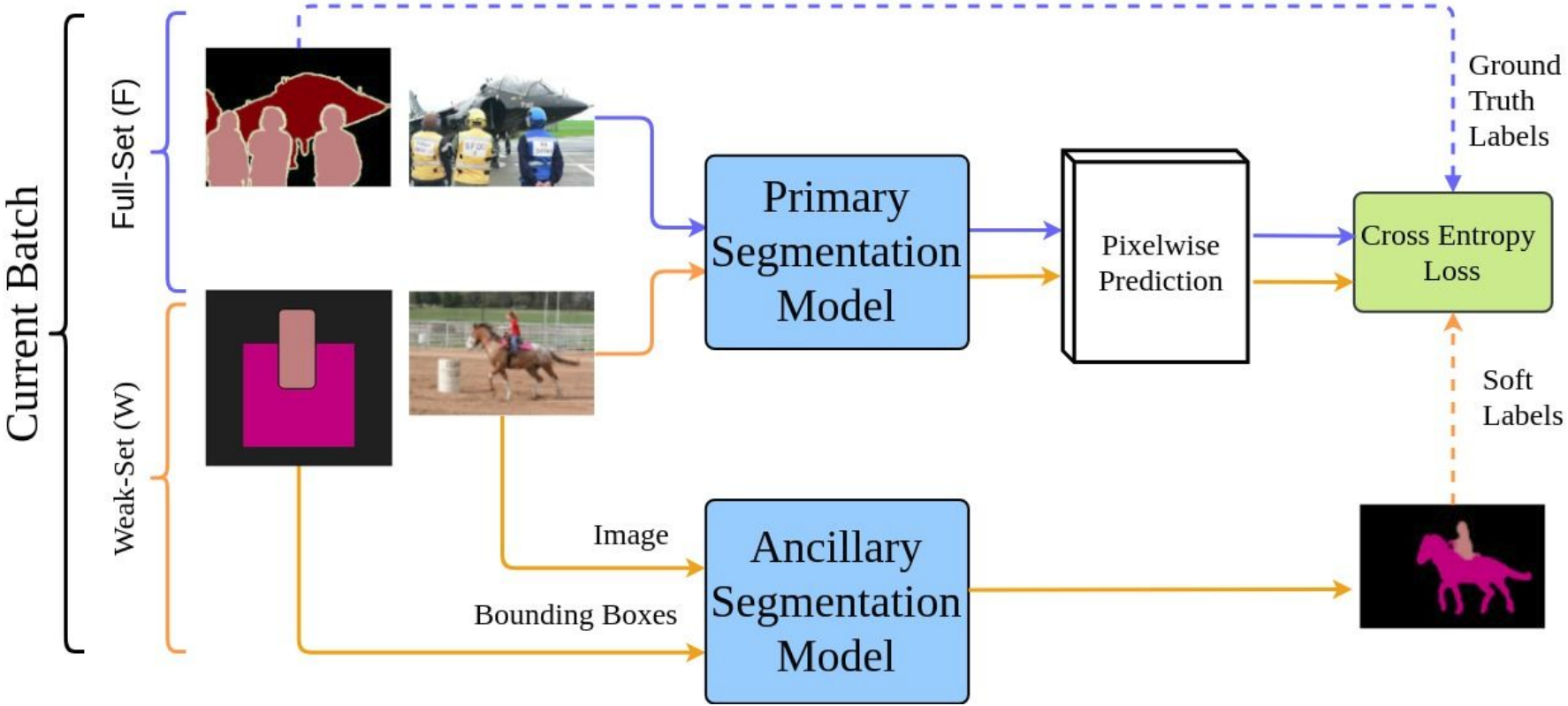
Label Smoothing

- Label smoothing is a regularization technique used in classification tasks to prevent the model from becoming **too confident** about the class labels
- Instead of using one-hot encoding, we give very small probability to the other classes
  - For example, instead of [1, 0, 0] use [0.9, 0.05, 0.05]
- How: decide factor α, e.g. 0.05
- Then the new gt (y') from old one hot encoding for N classes

$$y' = (1 - \alpha) \cdot y + \frac{\alpha}{N}$$

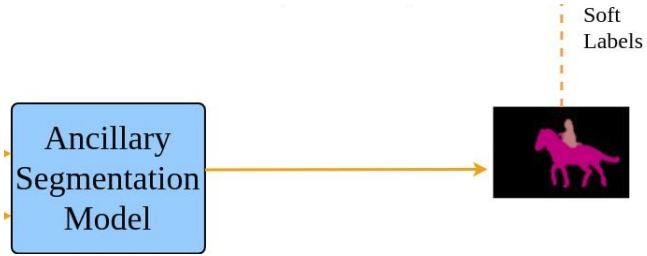
- Observe now cross-entropy sums from all terms NOT just a single one!

Recall: Soft Labels from Ancillary model



Recall: Soft Labels from Ancillary model

- In this model, we applied a trained segmentation model on the weak data
- Now, per pixel, the output is actually a distribution over N classes
  - Our pseudo-labels
- One can think in extracting the highest class and use as one-hot encoding
  - For 3 classes [0.2, 0.7, 0.1] ⇒ [0, 1, 0]
- Another promising direction is to use the output as it is
  - In this case, it is a soft (smooth) labels that consider the pixels uncertainty
- Again, this is a case where cross-entropy is using a distribution NOT just one hot encoding!



Statistical Distance

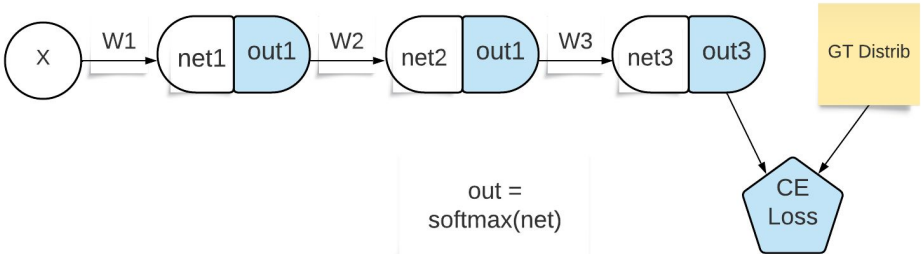
- In the big picture, there are many methods to quantify the difference between two probability distributions (hence might be used as a loss)
  - Cross-Entropy (Multi-class classification)
  - Kullback-Leibler Divergence (KL Divergence)
    - Used in variational autoencoders (**VAEs**) and in reinforcement learning
  - Earth Mover's Distance (Wasserstein **Distance**)
    - Common in **GANs**: Stable training, Meaningful Gradients, Geometric Interpretation
  - Jensen-Shannon Divergence (symmetrized version of the KL Divergence)
  - Total Variation Distance
  - Hellinger Distance
  - Bhattacharyya Distance

Entropy in ML

- Entropy is very common in many fields and ML models such as:
- Loss Function: Cross Entropy
- Decision Trees and Random Forests
  - Feature Selection and Tree Pruning
- Information Gain and Mutual Information
  - "*Mutual information is a measure of the amount of information that one random variable contains about another random variable*"
  - Feature Importance
- Cluster Purity
- Misc use cases
  - Regularization, Reinforcement Learning, Anomaly Detection, Ensemble Methods
  - You will find some/several ML papers utilizing entropy somehow

Cross Entropy with Softmax

- The last critical piece is to understand why cross-entropy/softmax are very common pair for a multi classifier!
  - In recent research literature people call it softmax classifier
- The interesting about this pair is the simplicity of the **final derivatives** at the output layer
- ∂L / ∂net3 = out3 - gt3
- Tip: it is common when we design DNN networks, we end at the logit only
  - Then the loss function takes the logit and in numerica stable way compute the loss



Categorical Focal Cross Entropy

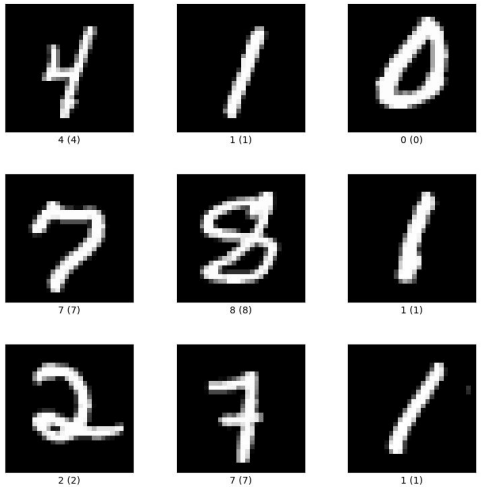
- We learned in logistic regression homework about focal loss
- We can trivially extend it with focal loss by for each class predicted probability multiply it with its alpha (class weight) and focal term
  - FL(p_t) = alpha * (1 − p_t)^gamma * CategoricalCE_vector(y_true, y_pred)

Convexity

- *As far as I know*, Hessian matrix is not positive semi-definite for all logit inputs z, which is a requirement for convexity.
  - Therefore, the cross-entropy with softmax is not convex
  - However, with DNN our network can still work fairly well
- If you have interest to derive that by yourself
  - Better try that with only 2 classes
  - Math is going to be a bit lengthy

MNIST dataset

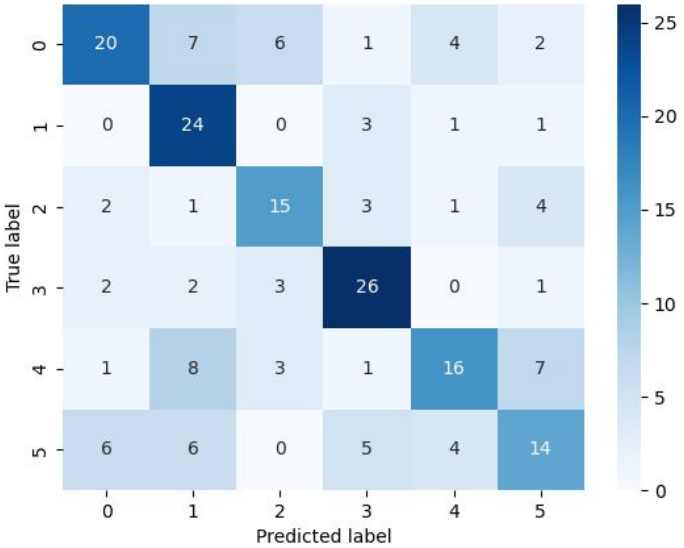
- The dataset consists of 70k handwritten digits (0 through 9) in grayscale, and each image is 28x28 (784) pixels
  - X is (70000, 784) and y is (70000) with values from [0-9]
- How do we normalize images? Most common way img / 255.0
- Tip
  - Always start with a smaller dataset version of fair size
  - Use it to validate and get good baseline
  - Then use the major dataset
  - I downloaded a good 5k sample for your HW
    - You can get performance 85%-91% with it
    - With full dataset, 98+%

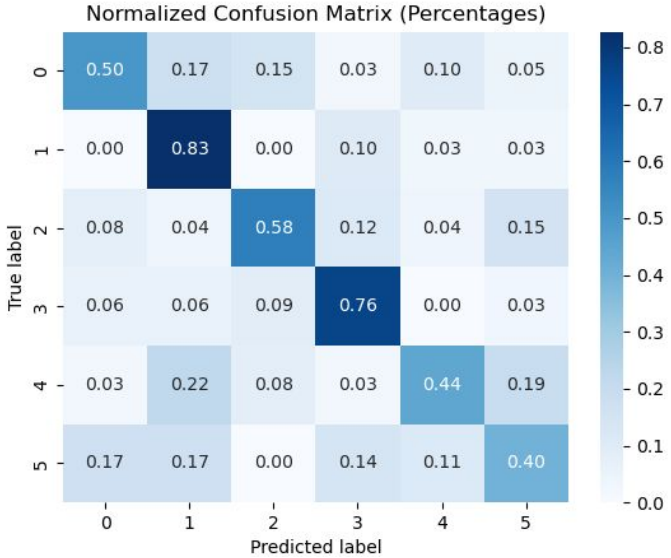


```
X = np.load('/home/moustafa/0hdd/research/ndatasets/mnist/sample/X.npy')
y = np.load('/home/moustafa/0hdd/research/ndatasets/mnist/sample/y.npy')
# We typically normalize image pixels to be in range [0-1]
X = X / 255.0

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.2, random_state=42)

model = MLPClassifier(hidden_layer_sizes=(10,), max_iter=20, alpha=1e-3,
                      solver='sgd', verbose=10, random_state=1,
                      learning_rate_init=.1)  # NN

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Test accuracy: {accuracy}")
```

Evaluation Metrics

- The same as binary classifier!
- Confusion matrix is now bigger and requires more analysis
  - Numbers vs Percentages





Tips

- Sometimes, you find some specific classes are problematic
- You may make a sub-dataset with these classes only
- Then do some train/investigations on that one
  - Ask yourself: why class A is confused with class B?
- You may combine similar classes
  - e.g. honda, toyota into car
  - e.g. phone texting, phone calling into phone usage
  - Do analysis

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."