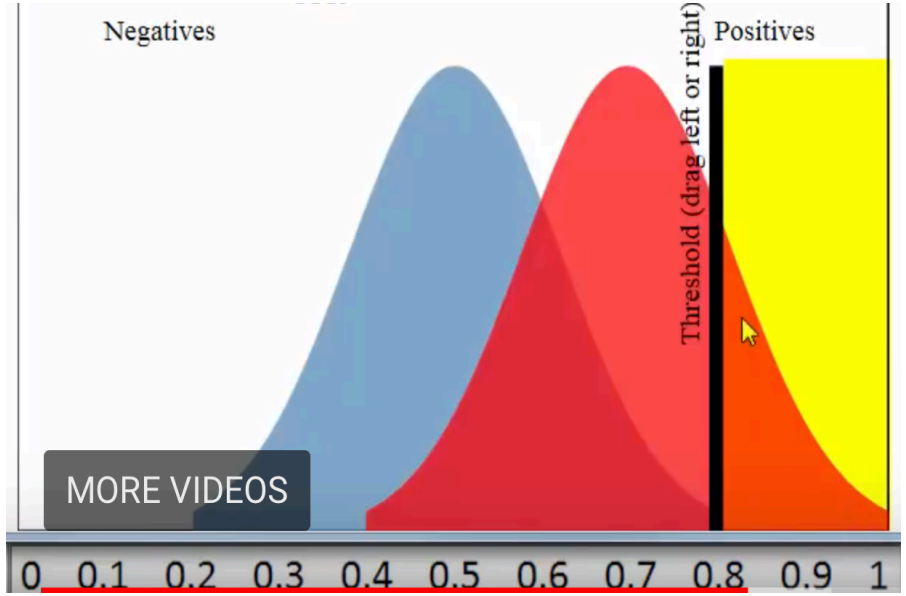
머신러닝 & 통계 기초

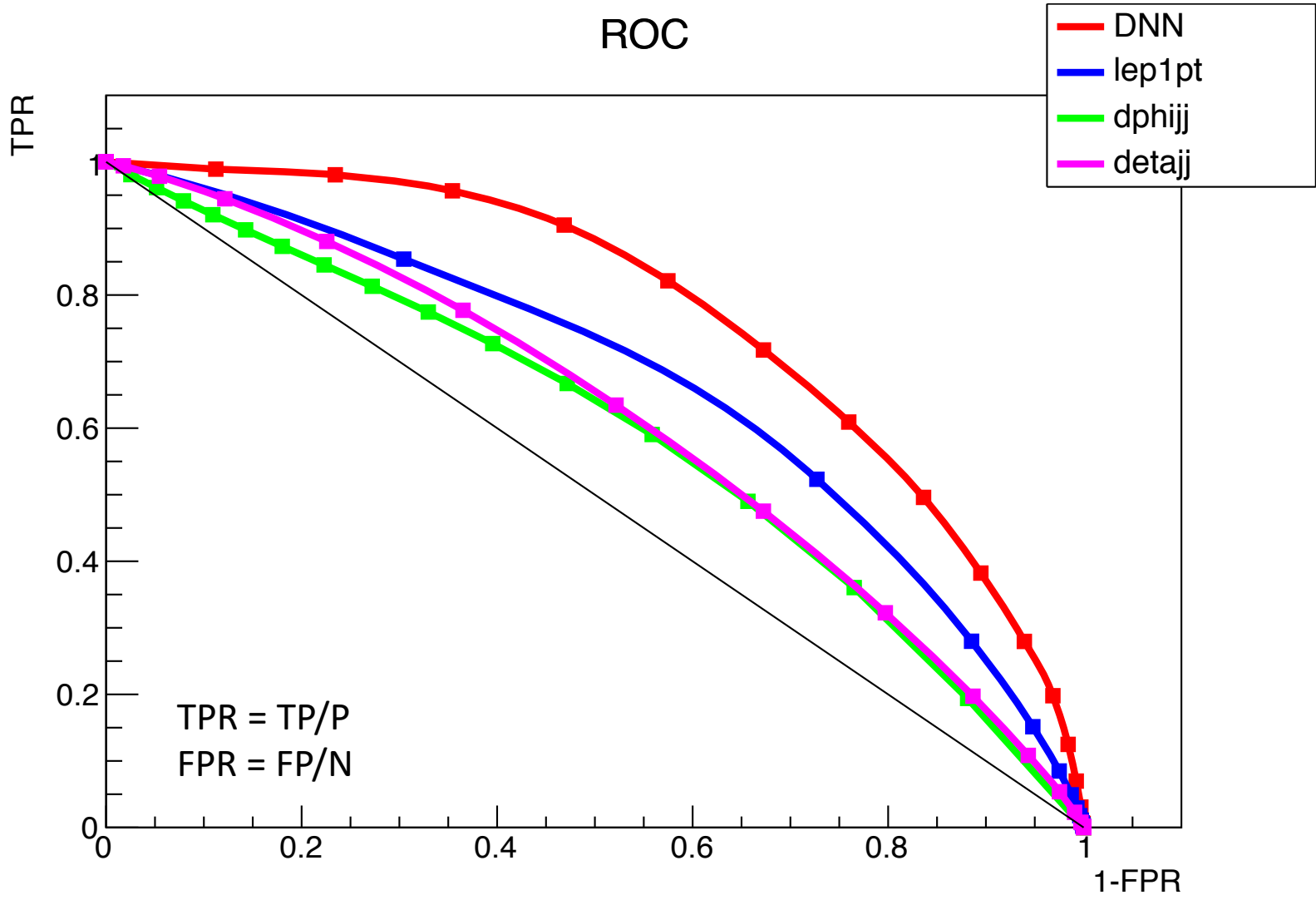
2019/11/29

이준호

ROC curve

Classifier / Predicted class	TRUE CLASS: p (positive)	TRUE CLASS: n (negative)
Y	True Positives	False Positives
N	False Negatives	True Negatives
Total	P	N





(http://web.ydu.edu.tw/~alan9956/docu/refer/roc_introduction.pdf)

Multivariate analysis

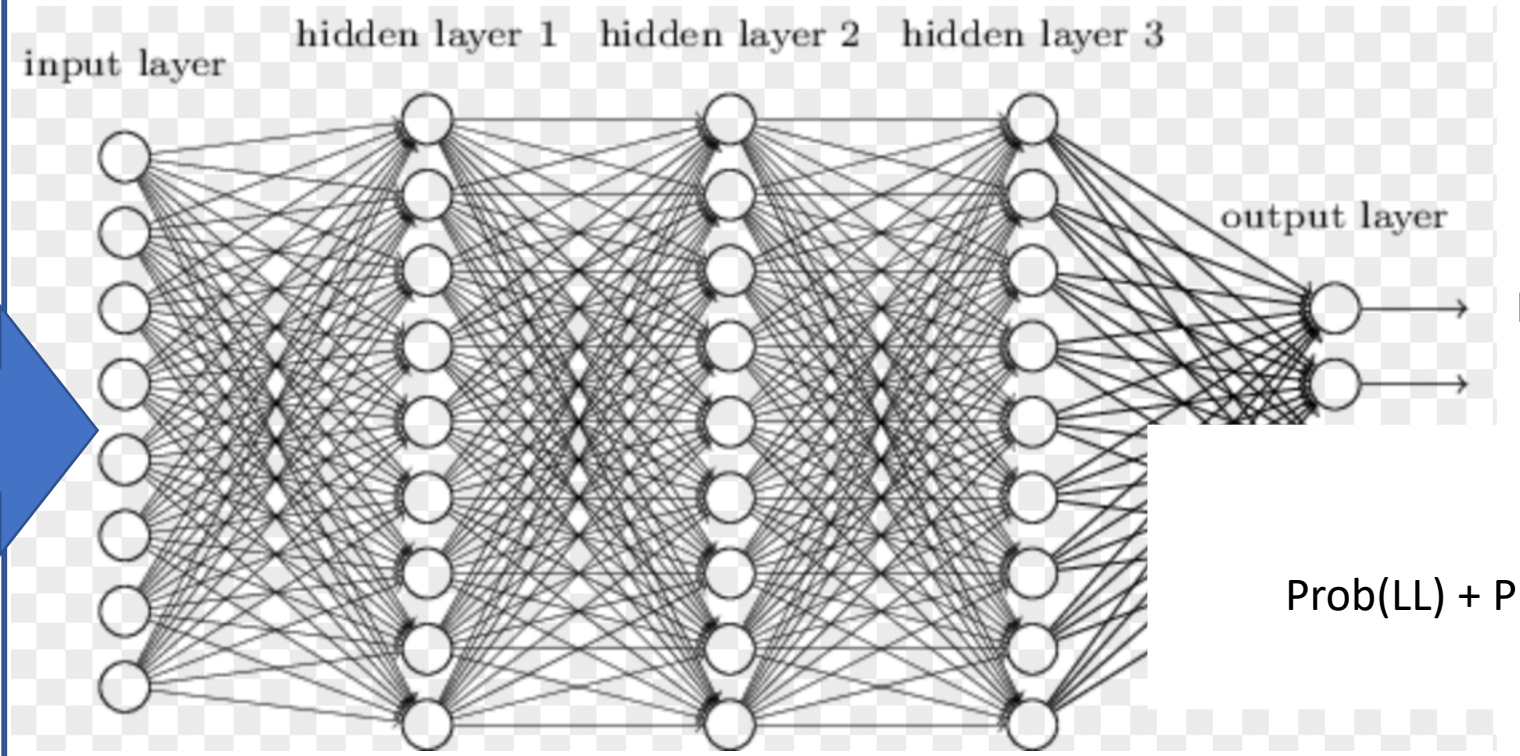
Dataset

- Train Dataset
  - Train Dataset → Obtain Trained DNN model → LL and TTTL classifier
  - Validation Dataset → Validate Training
- Test Dataset → Test performance of the model

Low level:

lep1pt,
lep1eta,
lep1phi,
lep2pt,
lep2eta,
lep2phi,
jet1pt,
jet1eta,
jet1phi.,
jet1M
jet2pt,
jet2eta,
Jet2phi,
jet2M,
MET,
METphi

High level:
dr_ll_jj,
dphijj,
detajj,
Zeppen_lep1,
Zeppen_lep2

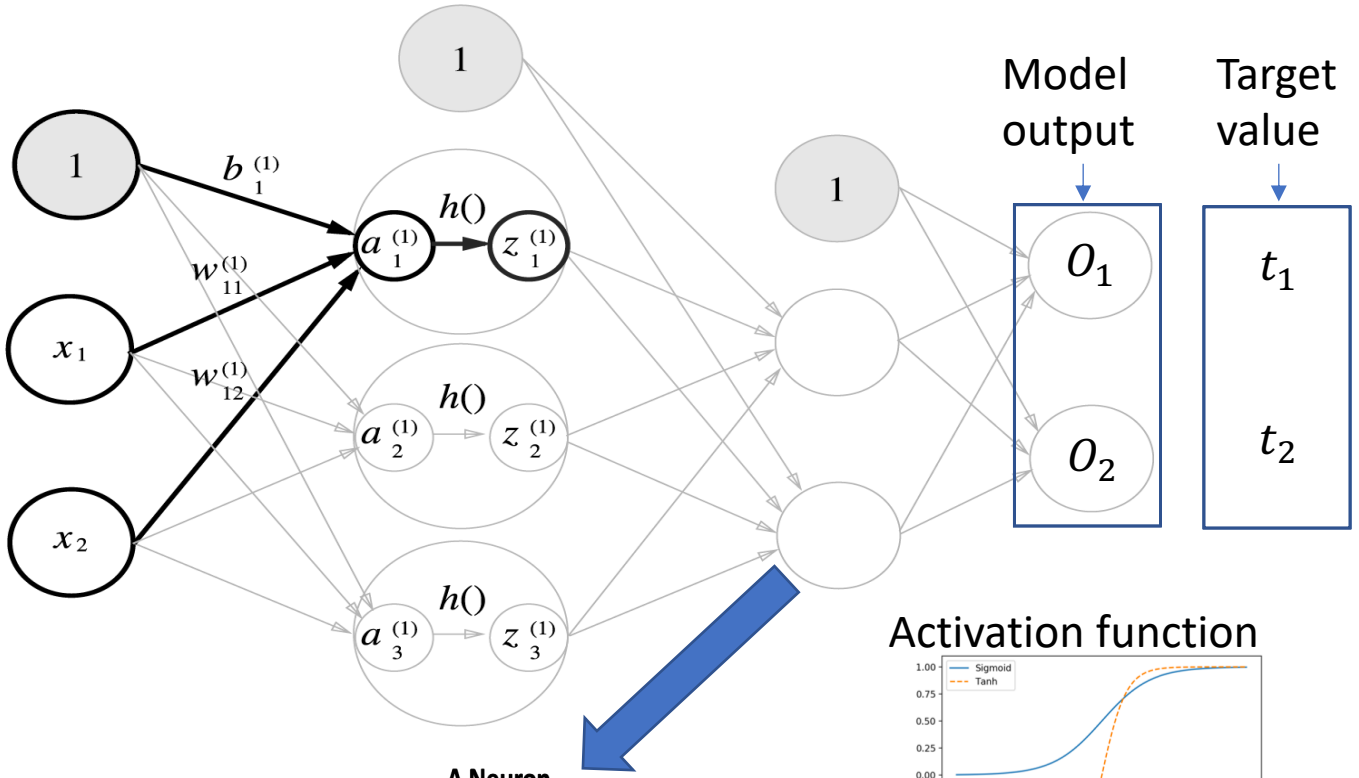


Prob(LL)

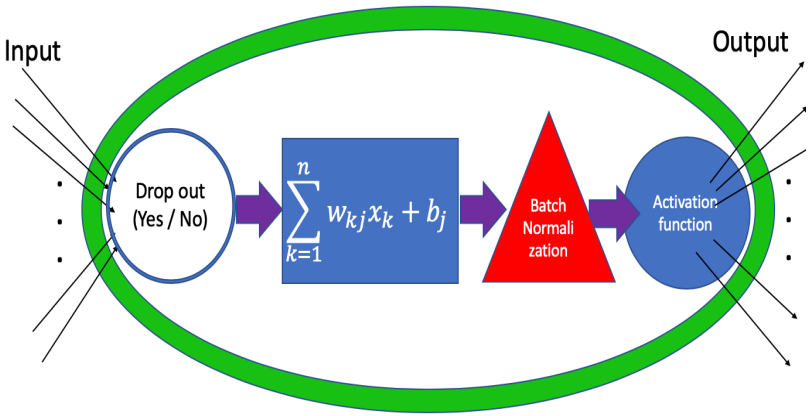
Prob(TTTL)

Prob(LL) + Prob(TTTL) = 1

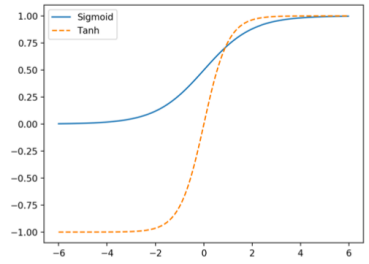
Neural network : 신경망

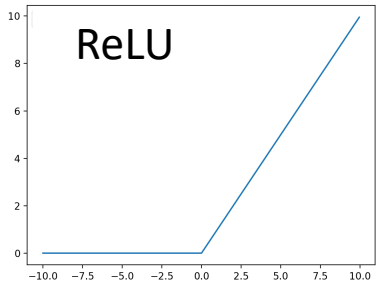


A Neuron



Activation function





- Given Dataset : $[[x_1, x_2], [t_1, t_2]]$
- Initialize weights of a NN model

1. Forward propagation
   - Activation function
   - Until reach to output layer
   - Calculates output of the model

2. Make loss function given output and Target value

$$\text{MSE} : L(\theta) = \frac{1}{n}\sum_{j=1}^{J}(t_j - O(x_j, \theta))^2$$

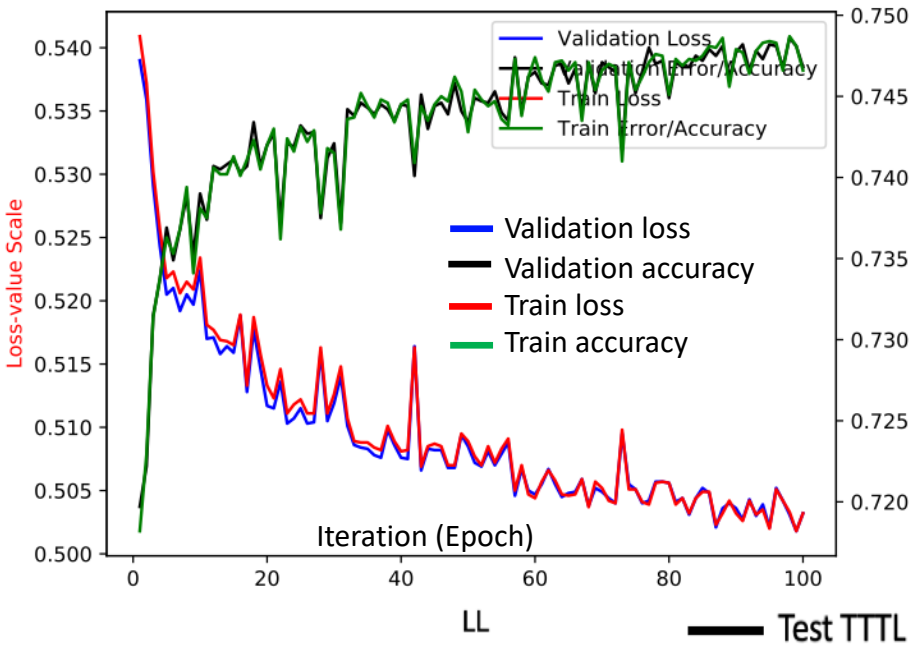
$$\text{Cross entropy} : L(\theta) = -\frac{1}{n}\sum_{j=1}[t_j log(O(x_j, \theta)) + (1 - t_j) log(1 - O(x_j, \theta))]$$

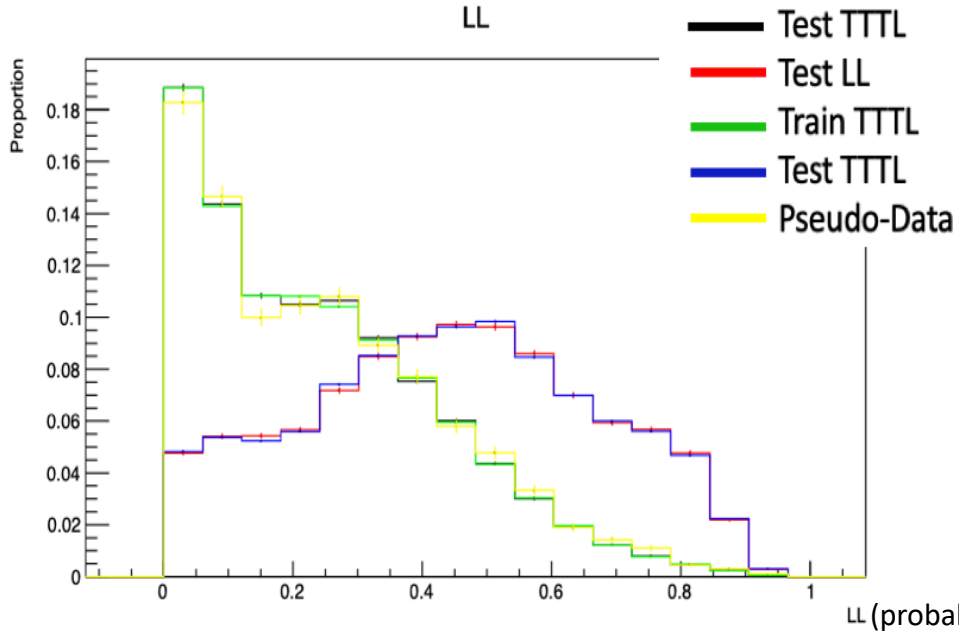
3. Back propagation : adjust weights and biases by gradients given loss function
   - Optimizer (SGD, Momentum, Adam)
   - Learning-rate (η)

$$w_{ij}^{\text{updated}} = w_{ij} + \Delta w_{ij} = w_{ij} - \eta \frac{\partial L_j}{\partial w_{ij}}$$

4. Minimize loss function by iterating 1~3.

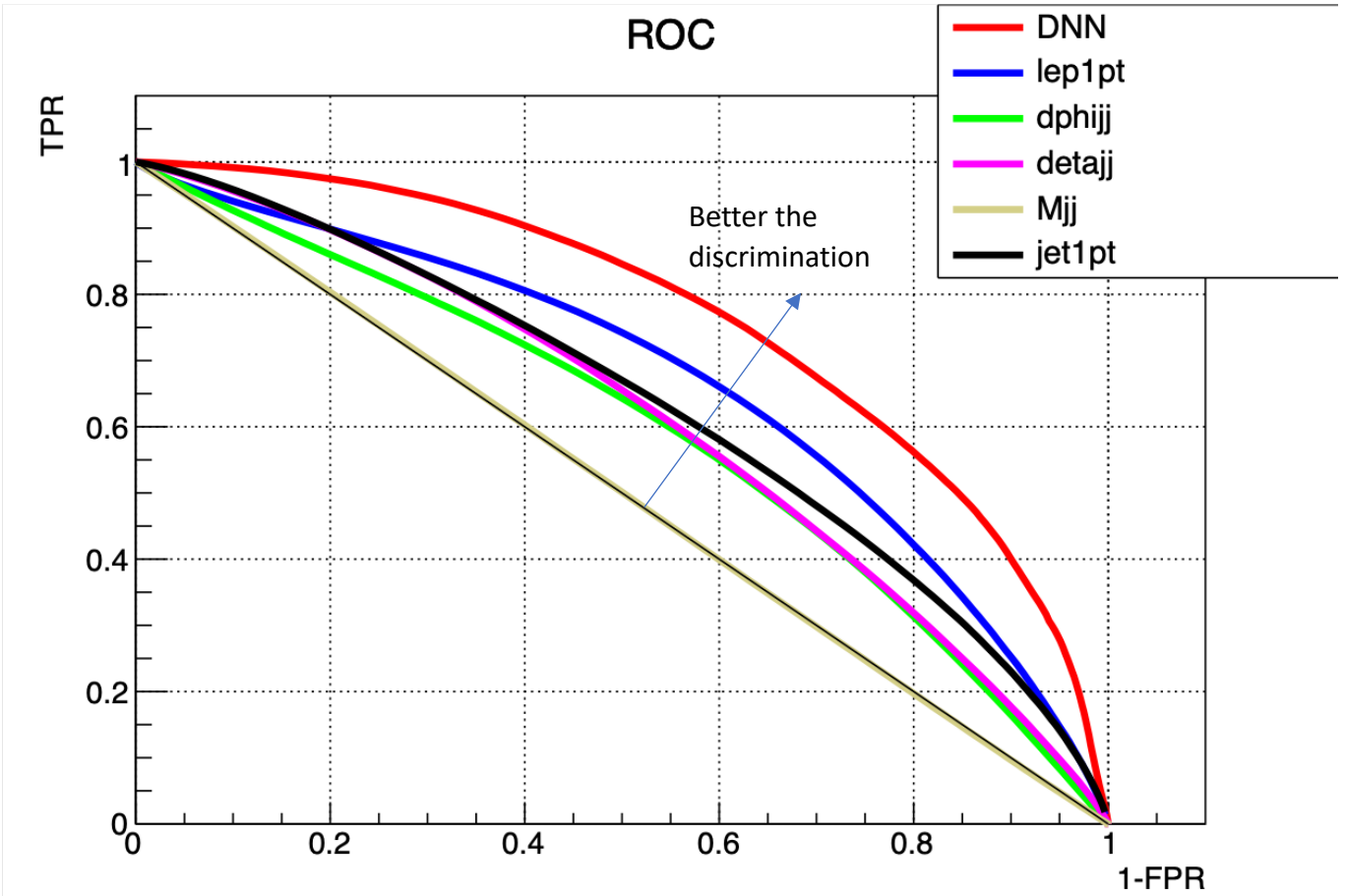
Multivariate classification (1) : Dense neural-network





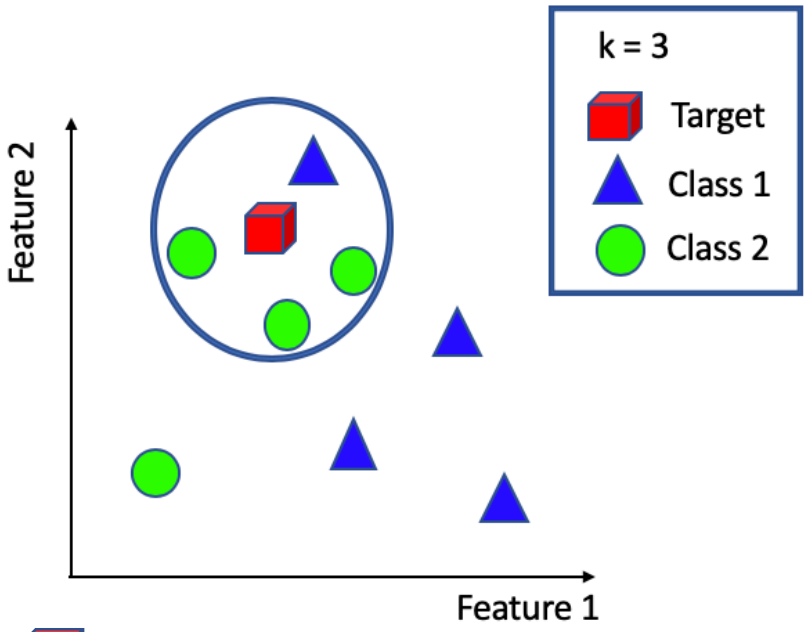
- 10 hidden-layered dense-NN
  - The best among 1~15 layered dense-NN
  - 150 neurons for each layer, ReLU activation function
  - 50% dropout and 50 epoch for early stopping

- Discrimination power improvement found from dense-NN

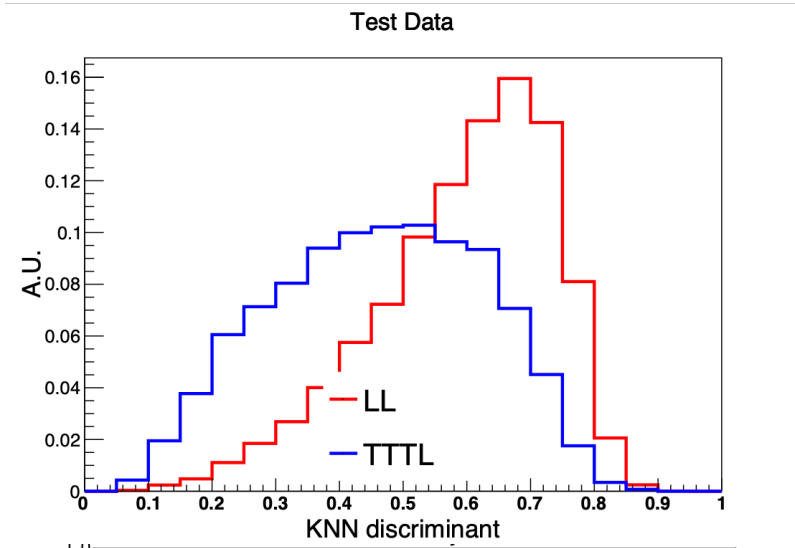


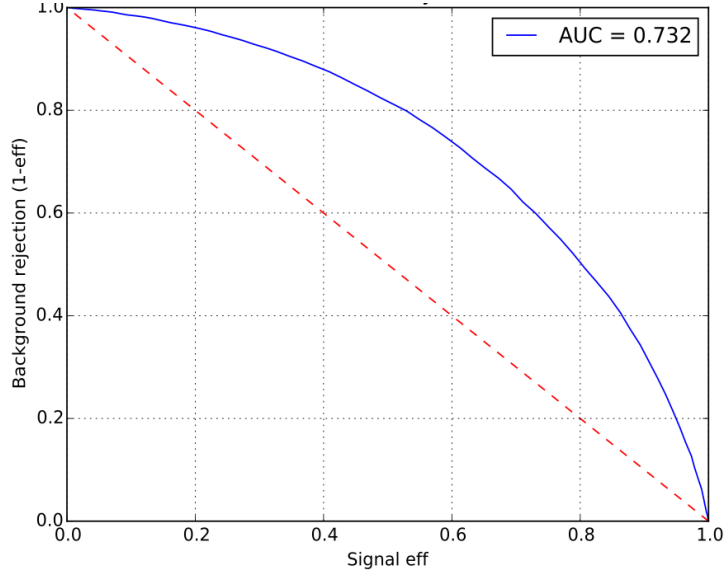
k-nearest neighborhood : 분류

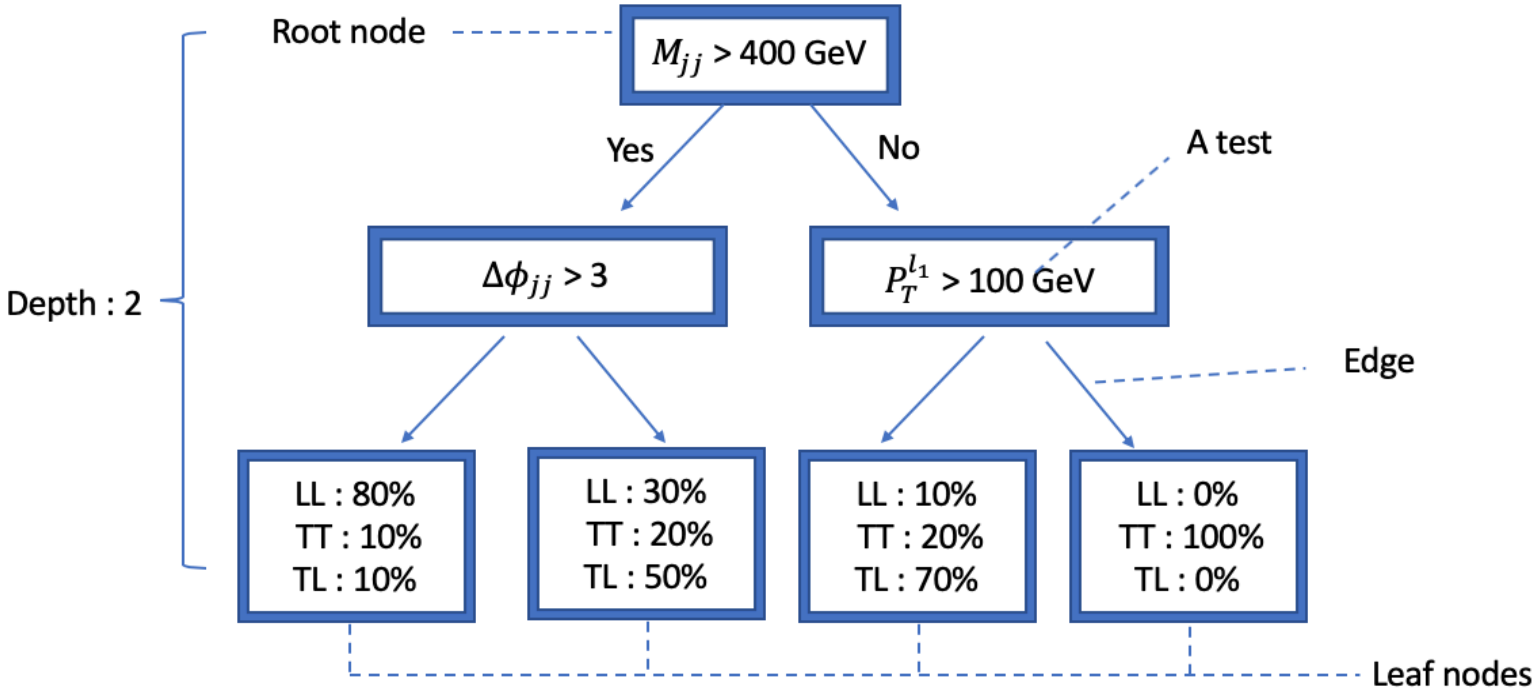
- Data itself constructs a k-NN model
- The model outputs probability on a certain data point to be classified as certain class
  - Feature space
  - With hyper-parameter ‘k’
  - Euclidean distance (optional)

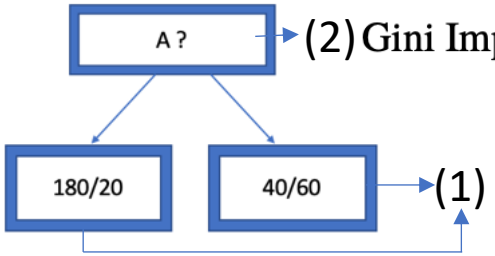


: would be inferred as Class 2







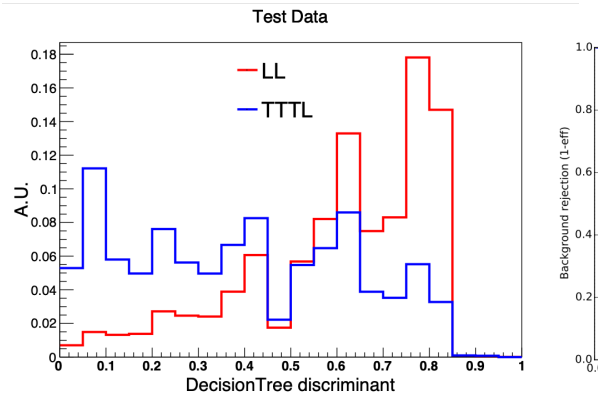


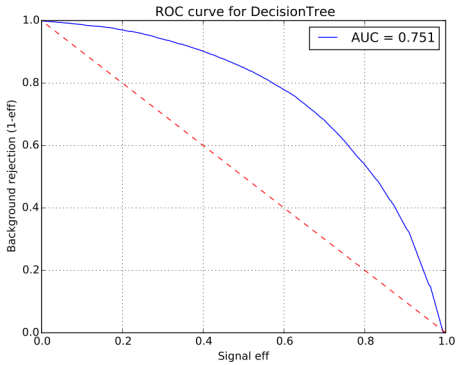
(2) $\text{Gini Impurity}_{\text{node}} = \text{W}_{\text{left}} \times \text{Gini Impurity}_{\text{left}} + \text{W}_{\text{right}} \times \text{Gini Impurity}_{\text{right}}$

$(W_{left} = \frac{N_{left}}{N_{left}+N_{right}}, W_{right} = \frac{N_{right}}{N_{left}+N_{right}})$

(1) $\text{Gini Impurity}_{\text{leaf}} = 1 - (\frac{N_{\text{true}}}{N_{\text{true}} + N_{\text{false}}})^2 - (\frac{N_{\text{false}}}{N_{\text{true}} + N_{\text{false}}})^2$

Only slightly better than k-NN

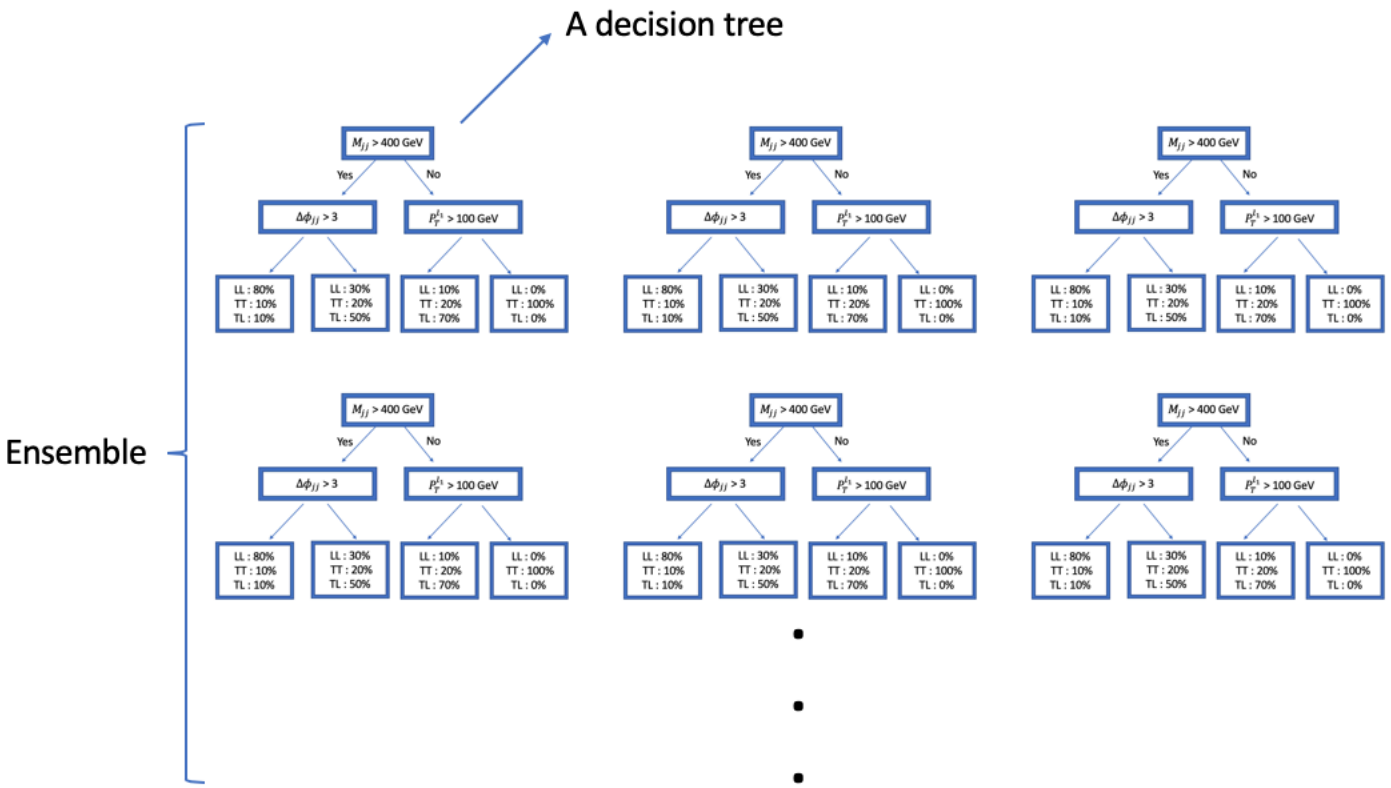




Decision tree : 분류

- Given Dataset : $[[x_1, x_2, \ldots, x_N], [t_1, t_2]]$

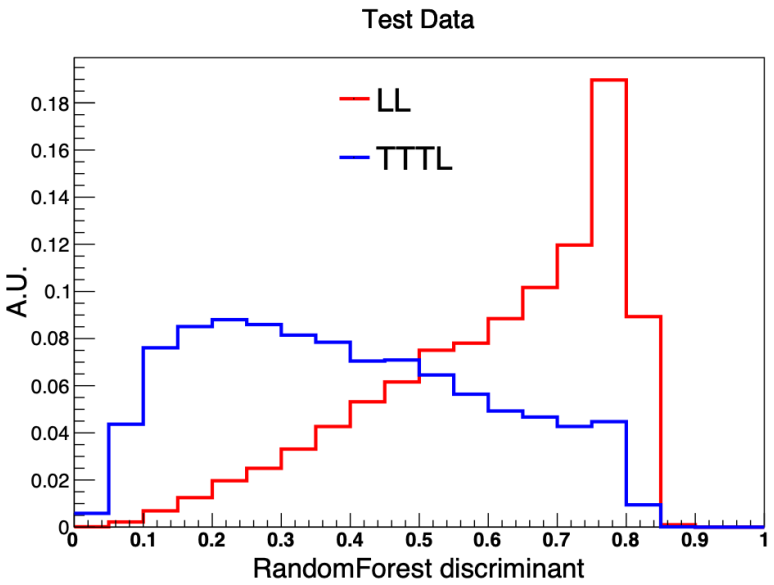
1. Find the best test which will be used for a node
   - Gini impurity leaf
     - Gini impurity node
2. If node itself has lower impurity score than any of its leaves' score, than the node become a leaf node
3. Stop separating if depth of tree reaches $N_{\max\_depth}$, which is a hyper-parameter for avoiding over-fitting.
4. Separate if not satisfies '2' and '3'.
5. Iterate 1~5.

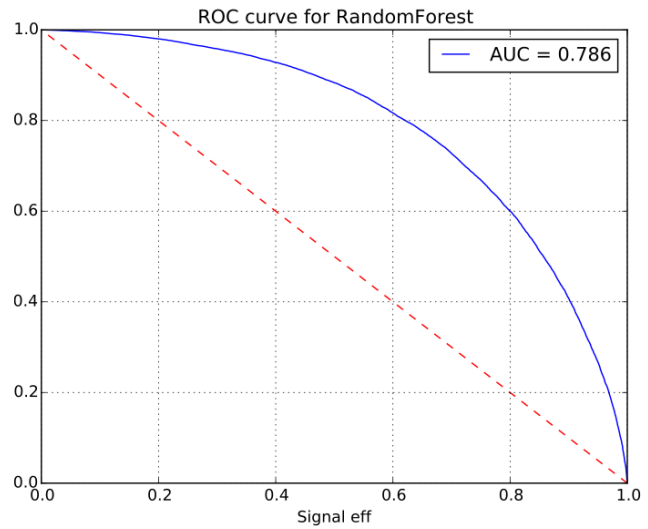


Random forest

- Given Dataset : $[[x_1, x_2, \ldots, x_N], [t_1, t_2]]$

1. Create a bootstrapped dataset.
   - Some data point might not included
2. Create a decision tree, while only taking n features, which is given hyper-parameter, when making a node.
   - Usually, n = $\sqrt{N_{feature}}$
3. Iterate 1~2 until number of trees reaches given hyper-parameter N_{tree}.

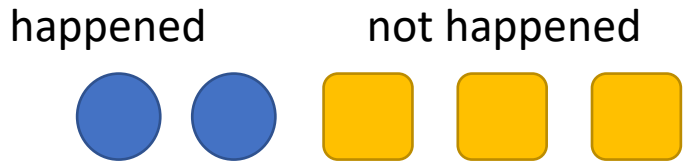




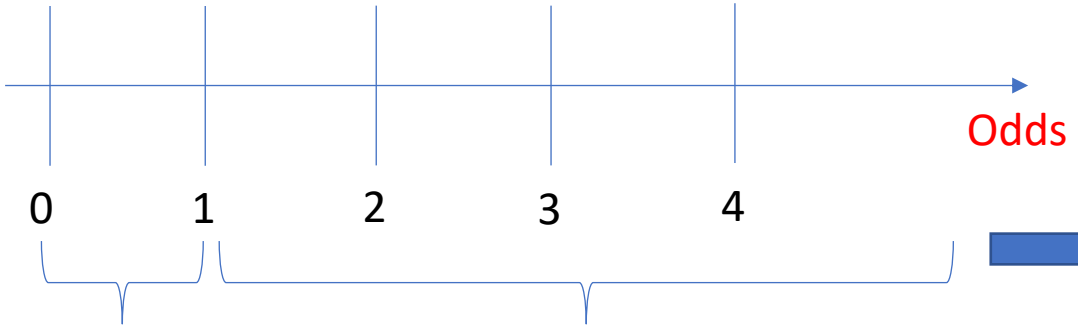
Better than k-NN, Logistic Regression, and Decision tree.

Linear regression : 선형회귀

Odds & Odds ratio

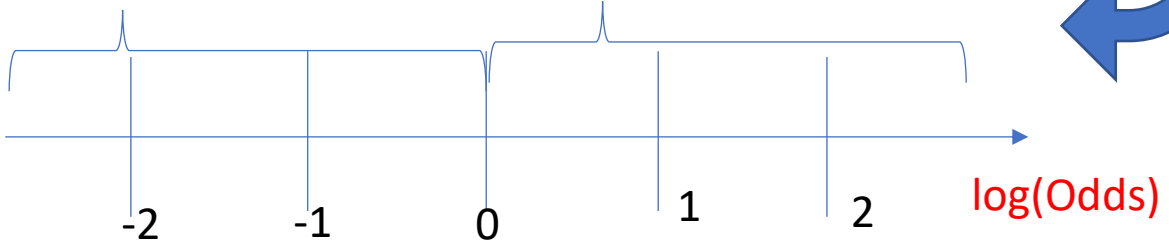


Odds = 2/3 → ‘1’ a boundary of a favor
Probability (p) = 2/5
Odds = p / (1-p) (log(Odds) = log(p/(1-p)) → logit function)



Taking log(Odds) for symmetric

Not favored Favored



Odds ratio & log(odds ratio)

		Has Cancer: Yes	Has Cancer: No	
Has the mutated gene	Yes	23	117	→ 23/117
	No	6	210	→ 6/210

Use odds ratio to determine relationship between mutated gene and cancer

- Odds ratio = $\frac{23/117}{6/210}$ = 6.88
- log(Odds ratio) = 1.93

Higher ‘odds ratio’ : better predictor (mutated gene)

Observed

Has the mutated gene	Has Cancer: Yes	Has Cancer: No
Yes	23	117
No	6	210

Odds (Wald test)

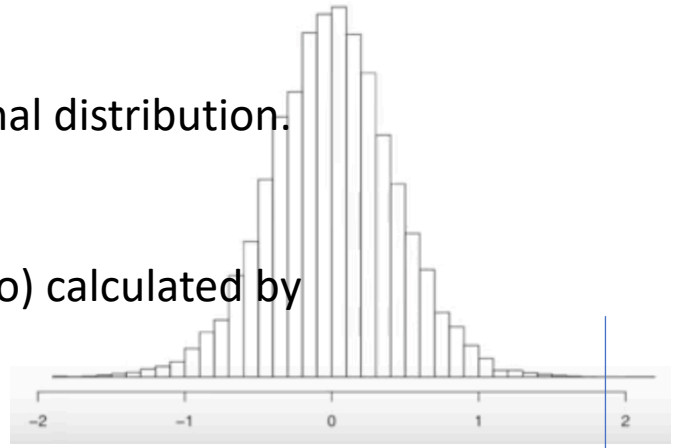
- Test if the odds ratio is statistically significant
  1. Fisher's exact test → m&m example : knowing expected distribution, how special the observation is, given by p-value
  2. Chi-square test → p-value → Compare observed values to expected values assume there is no relation between mutated gene and cancer
     - p(has cancer) = 29/356 = 0.08 (expected)
     - if no relationship, for those has mutated genes :140 * 0.08 = 11.2
     - No cancer for 128.8 people (expeted)

     →

     Expected Values

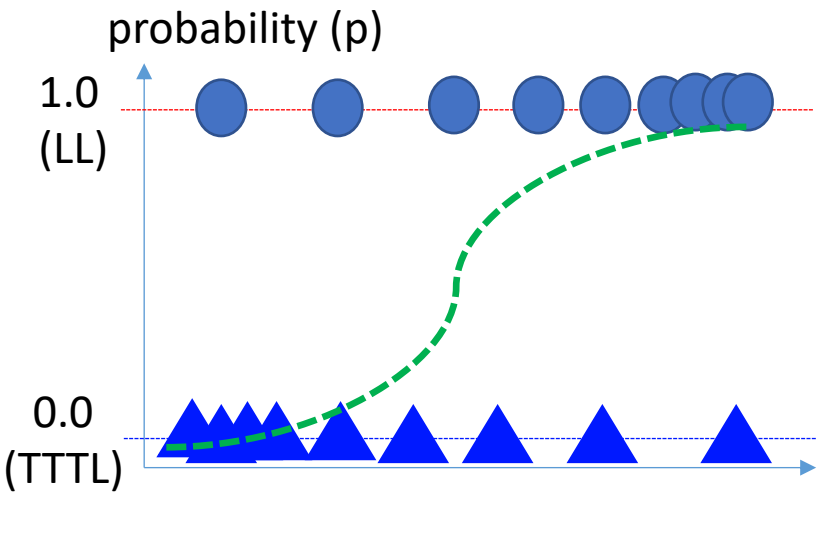
	Has Cancer: Yes	Has Cancer: No
Yes	11.2	128.8
No	17.3	198.7

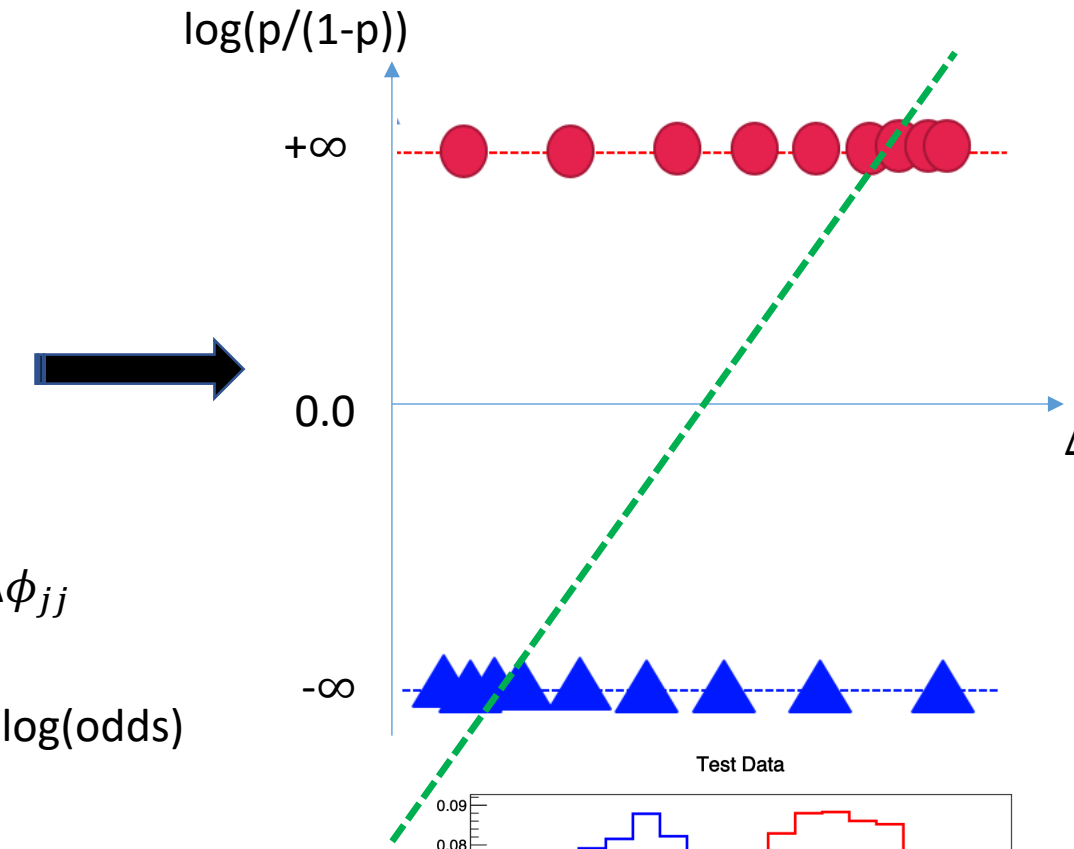
     p-value : 0.00001
  3. The Wald test → confidence interval
     - Takes advantage of log(odds ratio) is normally distributed (as well as log(odds))
     - Compute standard deviation of the normal distribution. (e.g. 0.43)
     - Compute p-value based on log(Odds ratio) calculated by observed data
       - 1.93/0.43 = 4.11 σ



log(Odds ratio) = 1.93

Logistic regression



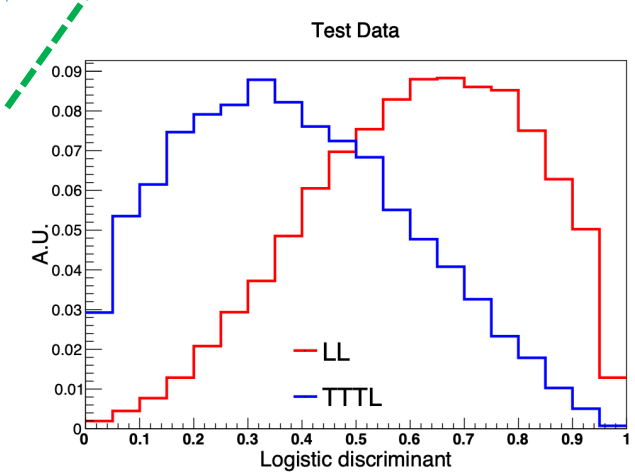


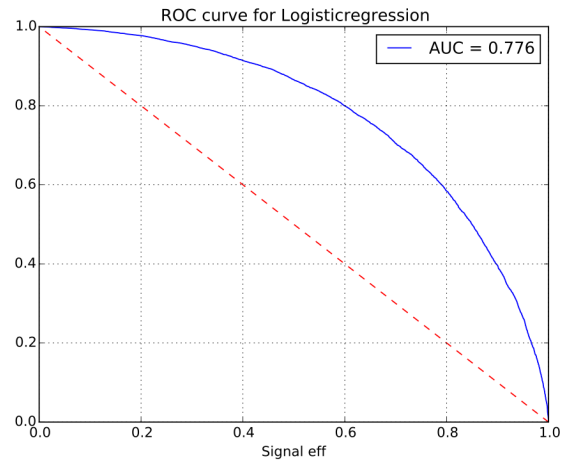
All 4-momenta taken into account, rather than just a $\Delta\phi_{jj}$.

(Slightly higher than k-NN)

- Transform Y-axis from probability (p) to log(odds) (log(p/(1-p))).
  - p = $\frac{e^{\log(odds)}}{1+e^{\log(odds)}}$
  - Turns into linear classification problem

- Construct maximum likelihood, rather than least squares, to have the best intercept and slope.





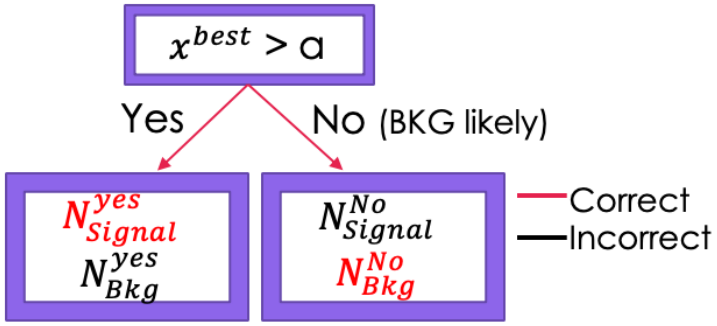
Boosted decision tree

[1] Adaptive boost (a.k.a. AdaBoost)

- Forest of stumps
- Diversity of weight for each stump
- Each stump is made by taking the previous stump's mistakes into account

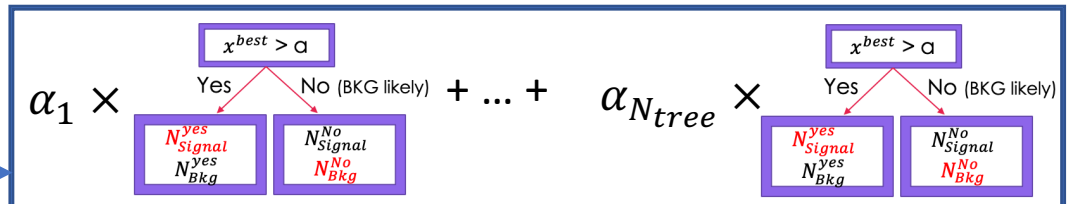
- Given Dataset : $[[x_1, x_2, \ldots, x_N], [t_1, t_2]]$

1. Give weights on each data point (w_i^k)
   - Uniform distribution if no preference for initial weighting
2. Find the best classifying variables among given X with weights, denote as x^{best}
   - Taken as a test of a stump
3. Compute misclassification rate : ϵ_k
4. Weight this stump as $\alpha_k = \beta \times ln((1-\epsilon_k)/\epsilon_k)$
5. Reweight Misclassification data points, while keeping the correctly classified ones
6. Iterate 2~5.
7. Trained model

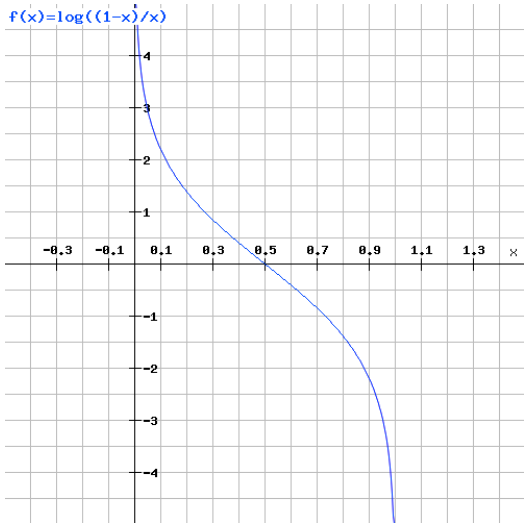


$$\epsilon_k = \frac{\sum_{i=1}^{N} w_i^k \times \text{isMisclassfied}_k(i)}{\sum_{i=1}^{N} w_i^k}$$

$$w_i^k \rightarrow w_i^{k+1} = w_i^k \times e^{\alpha_k}$$



$$\frac{1}{\sum_{k=1}^{N_{tree}} \alpha_k} \sum_{k=1}^{N_{tree}} \alpha_k T_k(i)$$



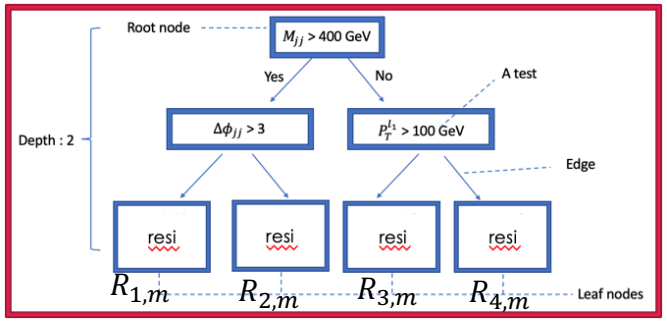
Boosted decision tree

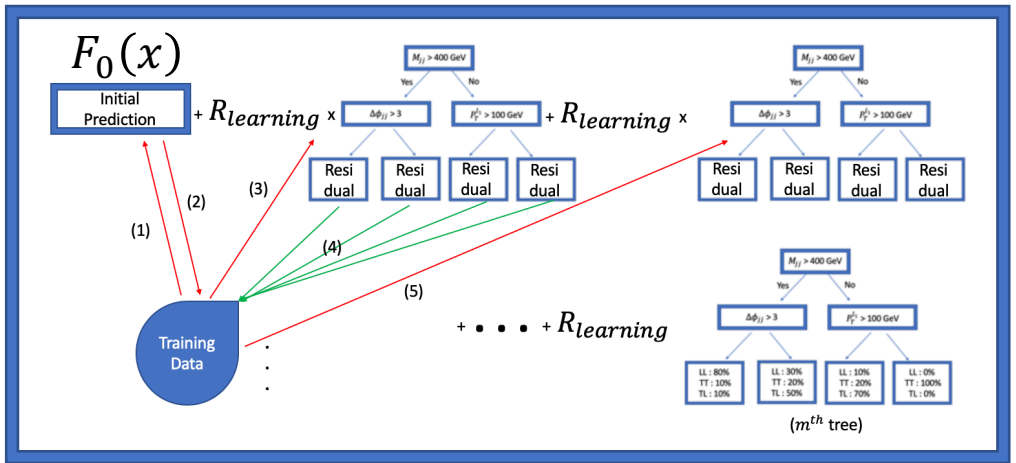
- Construct regression trees to minimize residuals
  - Make $F_0(x)$ as representative values of over all given data, and reduce residuals

[2] Gradient boost (On Regression)

- Given Dataset : $[[x_1, x_2, \ldots, x_N], [y_1, y_2, \ldots, y_N]]$

1. Define a differentiable loss function (ex: MSE) → $MSE: L(y_i, F(x)) = \frac{1}{2}\sum_{i=1}^{n}(y_i - F(x))^2$
2. Initialized model with a constant value : $F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
3. Make pseudo-residuals for each data point by $r_{i,m} = -[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}$
4. Establish m^{th} regression tree, where denote each leaf node as $R_{j,m}$ (j^{th} leaf of m^{th} tree)
5. For all $R_{j,m}$, compute $\gamma_{j,m} = \text{argmin}_\gamma \sum_{x_i \in R_{i,j}} L(y_i, F_{m-1}(x_i) + \gamma)$
6. Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1} \gamma_{j,m} I(x \in R_{j,m})$
   (ν is learning rate)
7. Iterate 3~6.
8. Trained model $F_M(x)$





Multivariate classification : Boosted decision tree

[2] Gradient boost (On Classification)

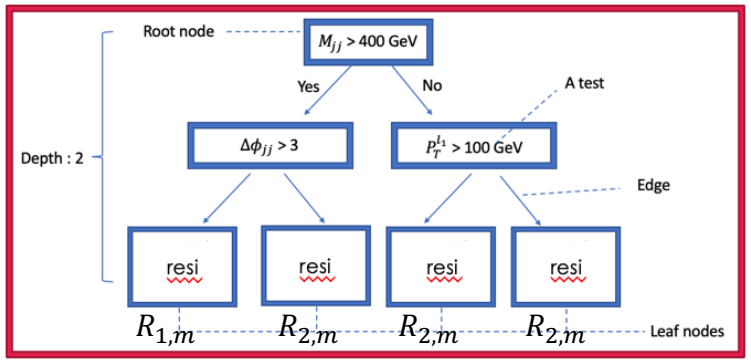
- Follow same steps with regression, usually takes minus log-likelihood as loss function :

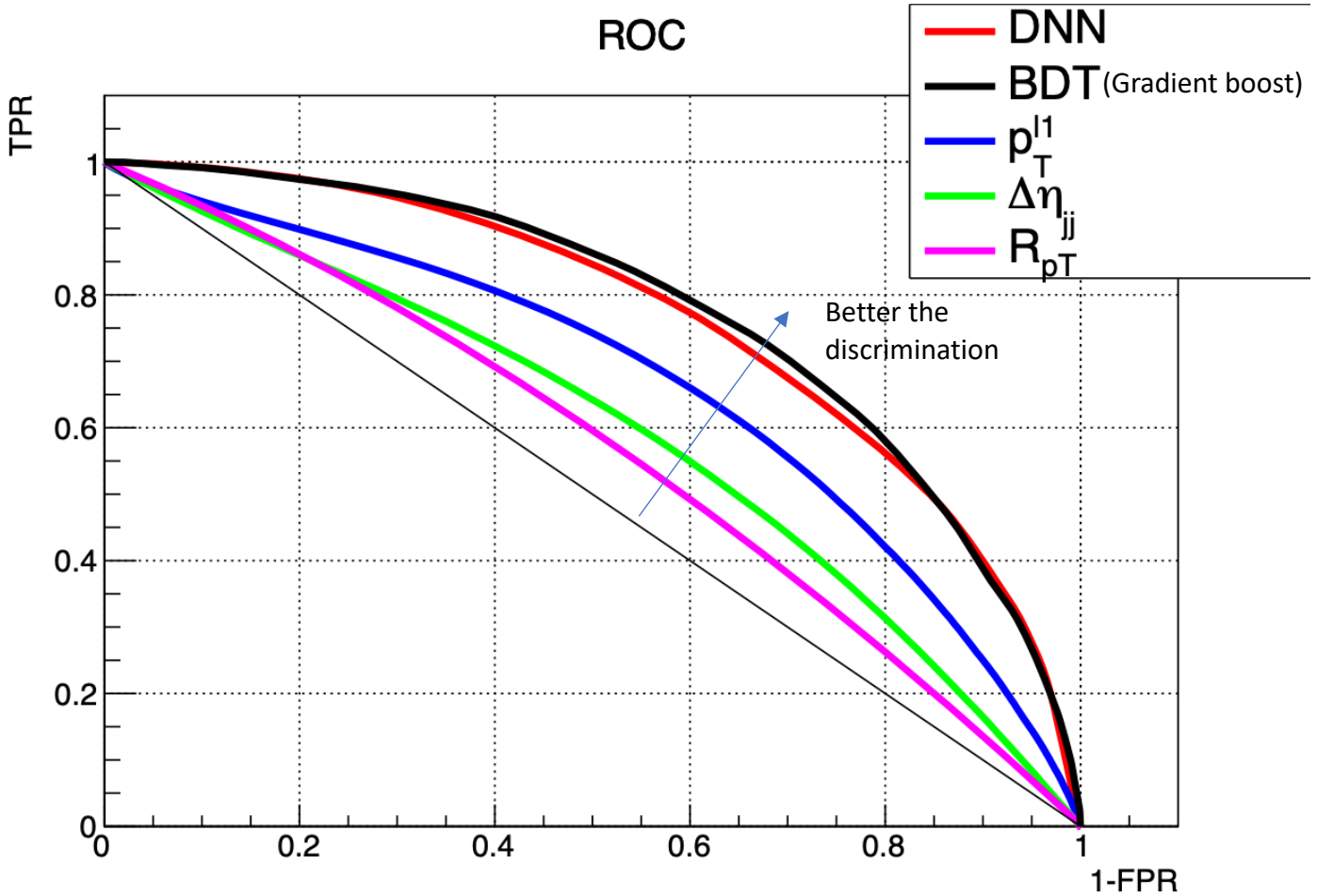
$$L(y_i, \gamma) = -\sum_{k=1}^{N} y_i \times \log(p) + (1 - y_i) \times \log(1 - p)$$

$$\text{where } \gamma = \log(\tfrac{p}{1-p}) = \log(odds)$$

- The trained model could be delivered as:

$$\gamma = F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{\mathrm{j,m}} I(x \in R_{\mathrm{j.m}})$$





1000 trees with 5 maximum depth applied for the BDT model.