FoML Assignment 1

Name - Sauradeep Debnath
Roll No - CS21MDS14025

ORDER --- all Questions serially including the Programming assignment

Q1.a – How changing K from 1 to n impacts training error in KNN Classification :

In KNN classification, when k=1, since we select only 1 closest neighbor - we simply assign a object its own label. Hence it is always correct for all the training examples. i.e. Train Accuracy = 100%. Training Error =0%

As we increase k (from 1 to n) – since the points in the neighborhood will have more & more probability to have a label different than our current point. Since we are taking a majority voting among the neighbors, the training error will keep increasing.

Initially this rise will be faster, say from k=1 (when it is 0%) to k=3(when there are 3 neighbors -the first one the point itself, the other 2 points might or might not be of the same class) as opposed to from k = 10 to 11. It finally the training error reaches its maximum as in this extreme case, all the training data points simply get the same label i.e. that of the majority class in our training data.

Thus, the change of training error could be described with the below graph:

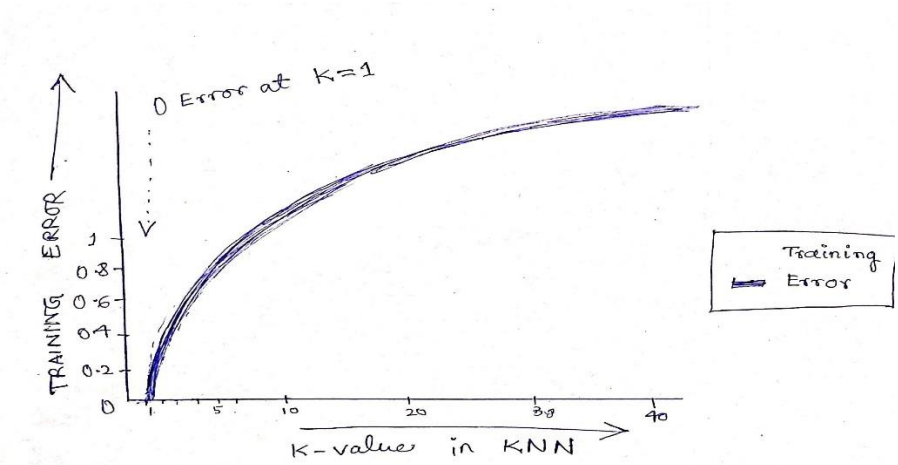


Fig: Illustration how the training error varies as K increases from 1 to n

Q1.b –

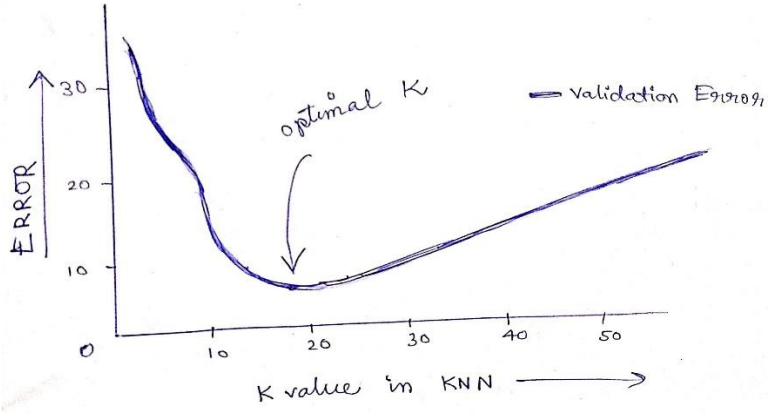
How Generalization error varies as K increases from 1 to n in KNN :

At k =1, the generalization error will be high – as we are just considering one neighbor as it will be more sensitive to noise & be overfitting (i.e. high variance)

As k increases, we are taking into account more & more points to classify the training sample. So it would be able to "generalize" better. Thus, generalization error will decrease.

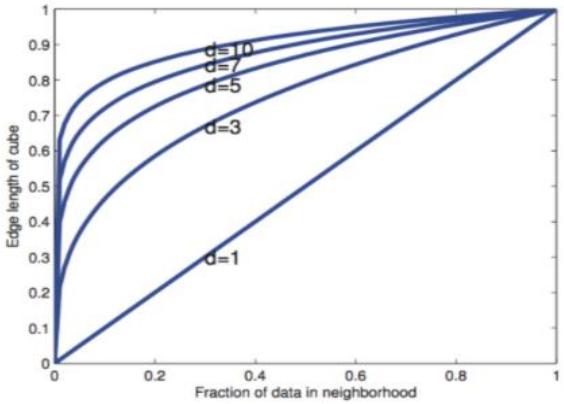
At a certain point between 1 to n , the error would reach a minima and then will start increasing. This is because now we are taking a huge part of the training set – and the chances that there are more points of a different class – will keep rising. At k=n, for every test example, we just predict the majority class.

The training error would be similar to this graph below:



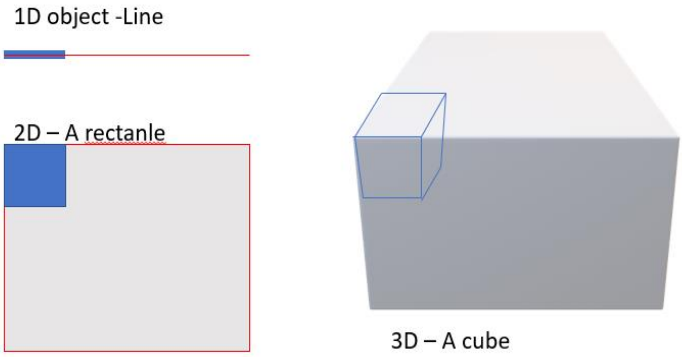
Q 1.c :2 Reasons Why KNN undesirable for high dimensional inputs:

1. **Curse of Dimensionality:**
   In the words of Schölkopf, "a high-dimensional space is a lonely place". i.e. the *density of data points keeps falling in a high D data space*. To understand this, let us look at this diagram from Kevin Murphy :



   This is the graphical representation of the formula, **edge(f) = f^(1/D)** which gives us the edge length required for covering f fraction of data points from a D dimensional Unit hypercube. Since f<1, D>1, as D increases, edge increases too, *ultimate reaching 1.0 for D=Infinity hypercube*, and all datapoints reside in the hypercube surface.

   Intuitive Example - we can see, a 0.1 unit length edge occupies f =0.1 data points of a unit line, f=0.01 in a square, & f=0.001 fraction of points in a cube.



Why this decrease in data point density impacts KNN more negatively- is that here the data points need to be close to each other in every single dimension. As D (the no. of dimensions) increase, the closest points get even more further (as compared to average distance). Thus, their predictive power becomes almost similar. Thus, the "nearest neighbors" are no longer useful in classification.

2. **Storage requires more space, query time slow –**
   The above discussed curse of dimensionality can be resolved if we have more data (thus increasing the data point density in the data space). But since KNN requires us to store all the training examples due to its "Lazy Learning" – adding a bigger dataset will add another new set of challenges i.e. higher storage & slower query time.
   For these above 2 reasons, it is not a good idea to use KNN in high-D data space.

Q 1.d - 1NN vs DT

ANS **- In general, NO. They will NOT give similar class labels as output, in general. Not unless if it is an 1- Dimensional dataset**

Explanation – For 1 D data, both DT & 1NN will select a given threshold from a Number line. Hence they could give us the same result. *But not when the dataset is more than 1 dimensional*. Let us take the example of a 2D dataspace to understand this (and the observations from this 2D space can be extrapolated to a n-Dimensional dataspace.) This is because for a 1 NN, the Decision Boundaries form a Voronoi diagram – they need not be parallel to any axis (in a x-y data space)
As example would be below, where the black dots are the data points:

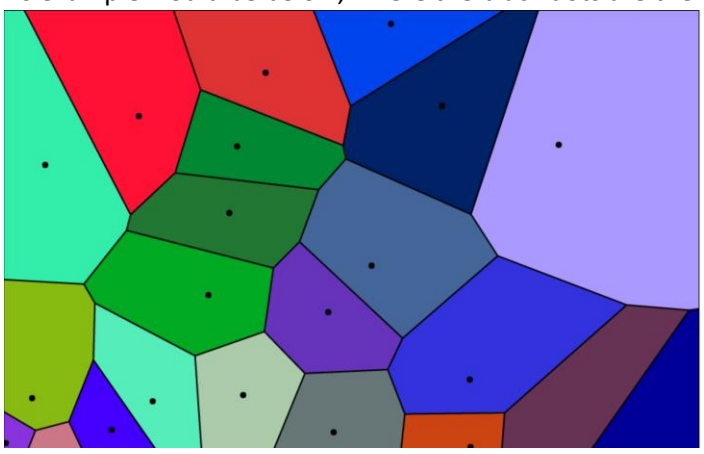
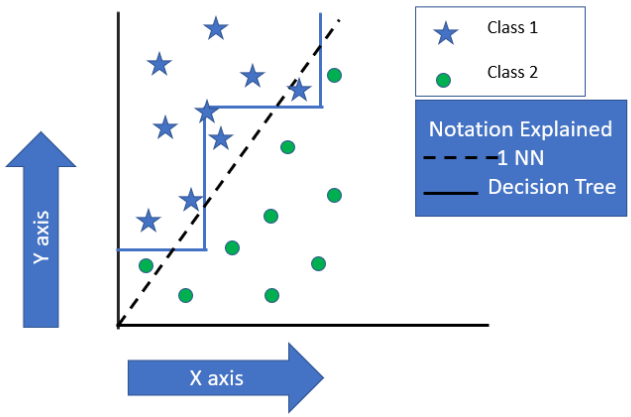
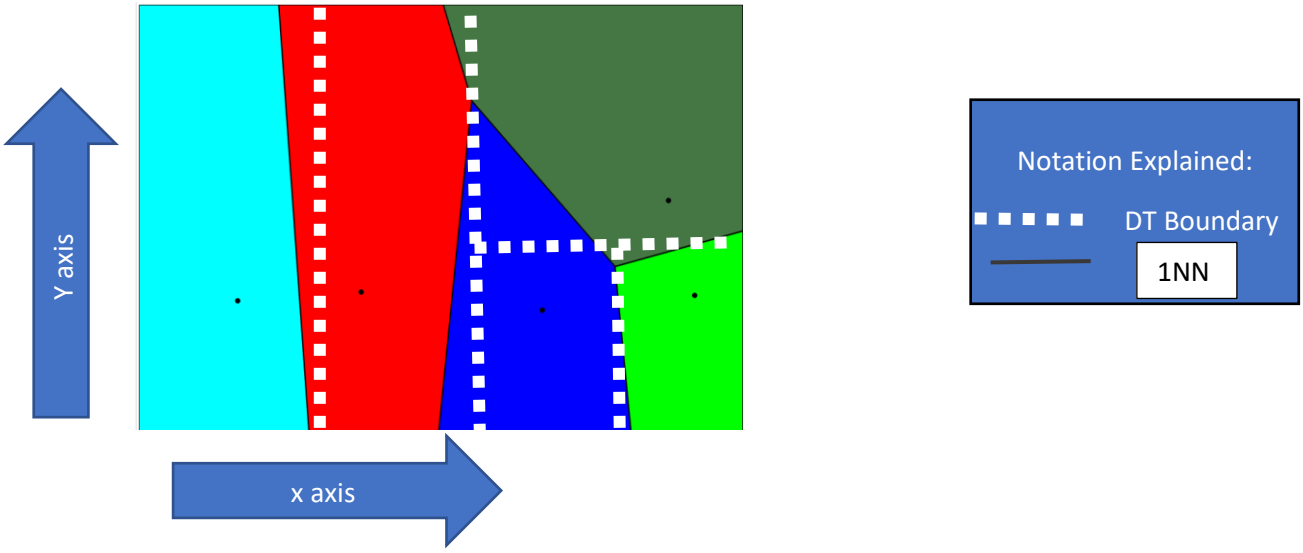


Fig – VORONOI diagrams

In case of a univariate Decision Tree, the Decision Boundary is always parallel to the x- or y-axes. Thus in some rare /unique cases (given a very specific set of data points) the results from these two methods might coincide , but not in general.
Here is an example illustrating how these two algos would classify data points differently, (i.e. assign two different labels) in general.



Below is one such **unique/exceptional** cases where 1NN & DT predict the same result. But again, this is not true for all dataset .



Q2.a -- Bayes Classifier –

As per Bayes theorem, the probability of datapoint x belonging to a class c1 is given by –

$p(c1|x) = p(x|c1).p(c1)/ \sum_{i=1}^{N} (p(x| c_i).p(c_i))$

Now, training examples {0.5; 0.1; 0.2; 0.4; 0.3; 0.2; 0.2; 0.1; 0.35; 0.25} ∈ class 1 and {0.9; 0.8; 0.75; 1.0} ∈ class 2. The label counts N1=10, N_2=4.

Hence , counting the class labels , we get the a priori probability of class 1 ,

P(c1) = Number of instances of class 1/ (total number of instances of class 1 & 2)= $N_1/(N1+N_2)$

= 10/(10+14) = .7143

the a priori probability of class 2,

P(c1) = Number of instances of class 2/ (total number of instances of class 1 & 2)

= $N_2/(N1+N_2)$= 4/(10+14) = 0.2857

The value of observation x belonging to class c, $_j$ $P(x|c_j)$ is given by the below formula:

$$p(x|c_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{1}{2}(\frac{x-\mu_j}{\sigma_j})^2}$$ ---------- Equation 1

Where, using Maximum Likelihood estimation

We have, estimator of mean of class 1, **μ1 =** $(1/N_1)\sum_{i=1}^{N1}$ (x)= (1/10) *(0.5+ 0.1+ 0.2+ 0.4+ 0.3+ 0.2+ 0.2+ 0.1+ 0.35+ 0.25) = 0.26 where $\sum_{i=1}^{N1}$ (x) denotes summing over all instances of class 1 (and N1 = number of instances of class 1)

And **μ2 =** $(1/N_2)\sum_{i=1}^{N2}$ (x)= (1/4) *(0.9+0.8+0.75+1.0) = 3.45/4 = 0.8625

And estimator of variance of class 1, $\sigma1^2$ = $(1/N_1)\sum_{i=1}^{N1}$ $(xi - \mu1)^2$ = ((0.5-.26)^2 +(0.1-.26)^2 +(0.2-.26)^2 +(0.4-.26)^2 +(0.3-.26)^2 +(0.2-.26)^2 +(0.2-.26)^2 +(0.1-.26)^2 +(0.35-.26)^2 + (0.25-.26)^2)/10

=0.149/10 = 0.0149 (same as the value given in the question)

And $\sigma2^2$= $(1/N_2)\sum_{i=1}^{N2}$ $(xi - \mu2)^2$ = ((0.9-0.8625)^{2}+(0.8-0.8625)^{2}+(0.75-0.8625)^{2}+(1.0-0.8625)^{2})/4 = 0.036875/4 = 0.0092

Plugging in all the values in Equation 1,

P(0.6|c1) = √ (1/2 π $\sigma1^2$)* exp (-0.5* ((0.6- μ1)/ σ1)^2) =) = 1/√(2 π * 0.0149)* exp (-0.5* ((0.6-0.26)^2/ 0.0149)) = 0.0675

P(0.6|c2) = 1/√(2 π * 0.0092)* exp (-0.5* ((0.6- 0.8625)^2/ 0.0092)) = 0.0983

From Bayes Theorem,

p(c1|x) = p(x|c1).p(c1)/ $\sum_{i=1}^{N}$ (p(x| c_i).p(c_i))

p(c1|0.6) = 0.0675 *0.7143/ (0.0675*0.7143 +0.0983*0.2857) = **0.63192** **ANSWER**

Q2.b. Text Classifier –

This is a multivariate Bernoulli Naïve Bayes text classification problem.

The BOW matrices given in the question is –

$$x_{\text{politics}} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$x_{\text{sport}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Let us assume that out of the 8 words given, xj will denote the existence of the j^{th} word.

In politics the distribution of word j is –(i.e count of xj per column)

[2 1 1 5 5 1 4 5]

In sports, the distribution is –

[4 4 1 4 1 1 0 1]

Using the Bayes theorem,

P(y=politics |[x = (1; 0; 0; 1; 1; 1; 1; 0)])

= P([x = (1; 0; 0; 1; 1; 1; 1; 0)] | y=politics) * P(y=politics) / P([x = (1; 0; 0; 1; 1; 1; 1; 0)]

Now , looking at the label count , we have –

P(y=politics) = 6/13

P(y=sports) = 7/13

From the politics matrix, we can calculate the probability of each word xj being present given the topic is “politics”

P(x1=1|y=politics) = 2/6

P(x2=1|y=politics) = 1/6; P(x2=0|y=politics) =1-1/6

P(x3=1|y=politics) = 1/6; ; P(x3=0|y=politics) =1-1/6

P(x4=1|y=politics) = 5/6

P(x5=1|y=politics) = 5/6

P(x6=1|y=politics) = 1/6

P(x7=1|y=politics) = 4/6

P(x8=1|y=politics) = 5/6; ; P(x8=0|y=politics) =1-5/6

Therefore P([x = (1; 0; 0; 1; 1; 1; 1; 0)] | y=politics) * P(y=politics)

= $\prod_j$(p(xj =i|y=politics). p(y=politics) ----------(*Assuming Conditional Independence*)

=2/6*(1-1/6)*(1-1/6)*5/6*5/6*1/6*4/6*1/6 *6/13

= 0.001374

Again, In politics the distribution of word j is –

[2 1 1 5 5 1 4 5]

In sports , the distribution is –

[4 4 1 4 1 1 0 1]

Now probability of the given instance,(*assuming all xj(s) are independent*)

P([x = (1; 0; 0; 1; 1; 1; 1; 0)]

= 6/13 * (1- 5/13)*(1- 2/13)*9/13*6/13*2/13*4/13*(1- 6/13)

= 0.001957

The P(y=politics|x) = P(x|y=politics) P(y=politics)/P(x)

= 0.001374/ 0.001957

=0.70209 --- which is the probability of the given x instance belonging to the class politics

Again if we do not assume xj(s) are independent, we can use the equation from Bayes Theorem,

P(y=politics|x) = P(x|y=politics) P(y=politics)/ (P(x|y=politics) P(y=politics) + P(x|y=sports) P(y=sports)) ……. Equation 1

Therefore P([x = (1; 0; 0; 1; 1; 1; 1; 0)] | y=sports) * P(y=sports)

= $\prod_j$(p(xj =i|y=sports). p(y=sports) ----------(*Assuming Conditional Independence*)

NOTE on selection of Smoothing constant - our given question is a multivariate Bernoulli (i.e. for each word xj , we have a value 0 or 1) rather than being a multinomial problem (in which case we predict a word out of Vocabulary V for each i-th position – which is NOT our present problem) – Hence, we need to add 2 in the denominator & not the feature count (as per the Pseudo code from Stanford - https://nlp.stanford.edu/IR-book/html/htmledition/the-bernoulli-model-1.html & elaborated/ analyzed further in the Q& A in https://stackoverflow.com/questions/40448784/laplace-smoothing-for-bernoulli-model-for-naive-bayes-classifier - for the Bernoulli case, even for original numerator = denominator, the smoothed probability is ½ - which makes sense – as each feature can take 2 values -0 or 1) . Based on these, it is clear that we need to add 2 (not 8 – feature count) in the denominator.

Since the 7th word x7 is present 0 times for sports, & the original Sports word distribution being [4 4 1 4 1 1 0 1] & new test instance x = (1; 0; 0; 1; 1; 1; 1; 0)----- doing Add-one Laplace Smoothing, i.e adding 1 to numerator & k=2 in denominator (as each xj feature can take two values i.e. {0,1})

= (4+1)/(7+2)* (1-5/9) *(1-2/9)* (4+1)/(7+2) *(1+1)/(7+2)* (1+1)/(7+2)* (1+0)/(7+2) *(1-2/9) *7/13

= 5/9* (1-5/9) *(1-2/9)* 5/9 *2/9* 2/9* 1/9 *(1-2/9) *7/13

= 0.000245

From Equation 1 above, probability that given the new test instance- it belongs to class “politics”,

P(y=politics|x) = 0.001374/ (0.001374 + 0.000245) = 0.84867 **ANSWER**

QUESTION 3 : DECISION TREE IMPLEMENTATION:

Important Helper functions implemented :
get_impurity() – calculate impurity using Gini/ Entropy
information_gain() – Gets either the information gain as per Entropy OR the reduction in gini impurity when a parent node is divided into child nodes
get_best_threshold(),check_all_features(), get_best_threshold()- together they find the best splitting condition based on the input/independent features. Vectorized implementation to optimize the run time.
build_decision_tree(), classify_test_instance()—used to build DT recursively & classify using it
TO BUILD the Decision Tree & Predict on TEST DATA- we have 2 options :
run_decision_tree() → runs decision tree for a fixed test-train set
run_decision_tree_cross_val()→ runs 10 fold cross validation
--- **NOTE**- run_decision_tree_cross_val() first runs all the 10 fold cross validation & then prints the accuracies. So, after running it, we need to wait a while for all that 10 fold validation to complete

HOW TO RUN THE CODE-

1. Place the ML_Assignment_Decision_Tree_cs21mds14025.ipynb file in the same directory as the wine dataset ("wine-dataset.csv")
2. Run all cells up to & including run_decision_tree_cross_val()
3. For initial accuracy using Entropy – run the following code-

```
if __name__ == "__main__":
    ig_threshold =0 #0.01
    criteria ='entropy'# 'entropy' 'gini'
    minimum_leaf =5
    test_set = run_decision_tree(ig_threshold, criteria, minimum_leaf)
```

4. For initial 10 Fold accuracy using Entropy – run the following code-

```
start = time.time()
if __name__ == "__main__":
    ig_threshold =0
    criteria ='entropy'# 'entropy' 'gini'
    minimum_leaf =5
    test_set = run_decision_tree_cross_val(ig_threshold, criteria, minimum_leaf)
start = time.time()# EXECUTION TIME in SECONDs
```

5. For the **best result** in **cross validation**, run the code block next to this Header *(please search for "RUN This Block Only - Search Tag - cs21mds14025"* text in the notebook). **Alternately**, run the following code in a new cell -

```
start = time.time()
if __name__ == "__main__":
    ig_threshold =0.005 # better than gini 0
    criteria ='gini'# 'entropy' 'gini'
    minimum_leaf = 5
```

```
    test_set = run_decision_tree_cross_val(ig_threshold, criteria, minimum_leaf) #.8350
end = time.time()
print(end-start)
```

The accuracy of **Initial Implementation for Cross Validation** using Entropy is – **0.8322**

Info Gain Threshold	Impurity	#Leaves	Cross Val Accuracy	Accuracy	Comment
0	entropy	5	NA	0.818	
0	gini	5	NA	0.8139	
0.01	entropy	5	NA	0.816	Decreased Accuracy
0	entropy	5	0.8322	NA	Initial Accuracy
0.01	entropy	5	0.8336	NA	IMPROVED Accuracy - Best Entropy
0.03	entropy	5	0.804	NA	
0.001	entropy	5	0.8322	NA	
0.0005	entropy	5	0.8322	NA	
0.02	entropy	5	0.8154	NA	
0.015	entropy	5	0.8303	NA	
0.008	entropy	5	0.8328	NA	Improved Accuracy
0.012	entropy	5	0.8322	NA	
0.006	entropy	5	0.8326	NA	Improved Accuracy
0	gini	5	0.8279	NA	Gini Initial Accuracy
0.007	gini	10	0.8336	NA	Improved Accuracy
0.005	gini	10	0.835	NA	
0.005	gini	5	0.835	NA	**BEST ACCURACY**
0.006	gini	5	0.835	NA	
0.004	gini	5	0.834	NA	
0.005	entropy	5	0.8326		IMPROVED Accuracy

Improvement Strategies & Their Explanations –

2 strategies that worked in improving the accuracy –

1. Pruning Using Minimum Information Gain/ Gini Impurity Reduction Threshold
2. Gini Impurity with the above Impurity based Pruning

   What Did not work – No of Leaves based Pre-Pruning

   Details -

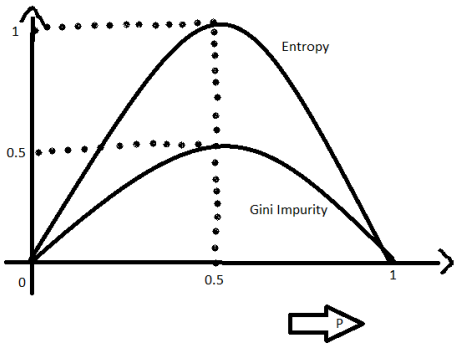
   1. **Post Pruning** based on Information Gain/ Gini Impurity Reduction Threshold – Increased to .8336 using threshold = 0.006.

   Explanation- This is because Pruning is **reducing overfitting & increases generalization** . Thus it is performing better on a new /unseen test instance.

   2. **Different Impurity measures** – for gini , with no pruning we only get 0.8279 . But this increases to 0.8350 with a Info gain minimum threshold of .005 (i.e. Gini + impurity based pruning)
   3. **Pre Pruning based on no. of leaves**— I experimented with threshold 5 & 10. Could not see any impact. This is probably because pre-pruning usually does not contribute more to improving the performance. Post pruning usually does, which we can see both intuitively & empirically (as we go all way down to fit the training data & later discard some of the lowest levels -hence better generalization capability.

   Why is Gini performing Slightly better?

Explanation _ As the formula & graphs of gini impurity & entropy are very similar, they give us almost similar results



$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

Image Source/credit – geeks for geeks

Without Pruning, Entropy performs better than Gini – because it handles imbalanced data slightly better. This is likely due to the fact that in gini, due to the square of probabilities – the lower probabilities are ignored.

Gini is of course faster than entropy, due to using the square probability in the formula rather than log probability.

However as various Pruning criteria were explored, Gini with Pruning is working slightly better than Entropy with Pruning- however the difference is negligible and is likely due to 1. the specific distribution of data & the specific dataset ("no free lunch theorem" – i.e. there is no one algo that works best for all possible problems- it varies) . 2. Secondly, the present implementation is inspired by CART. CART uses gini impurity for split and its implementation is likely optimized by using Gini.