CSCE 633

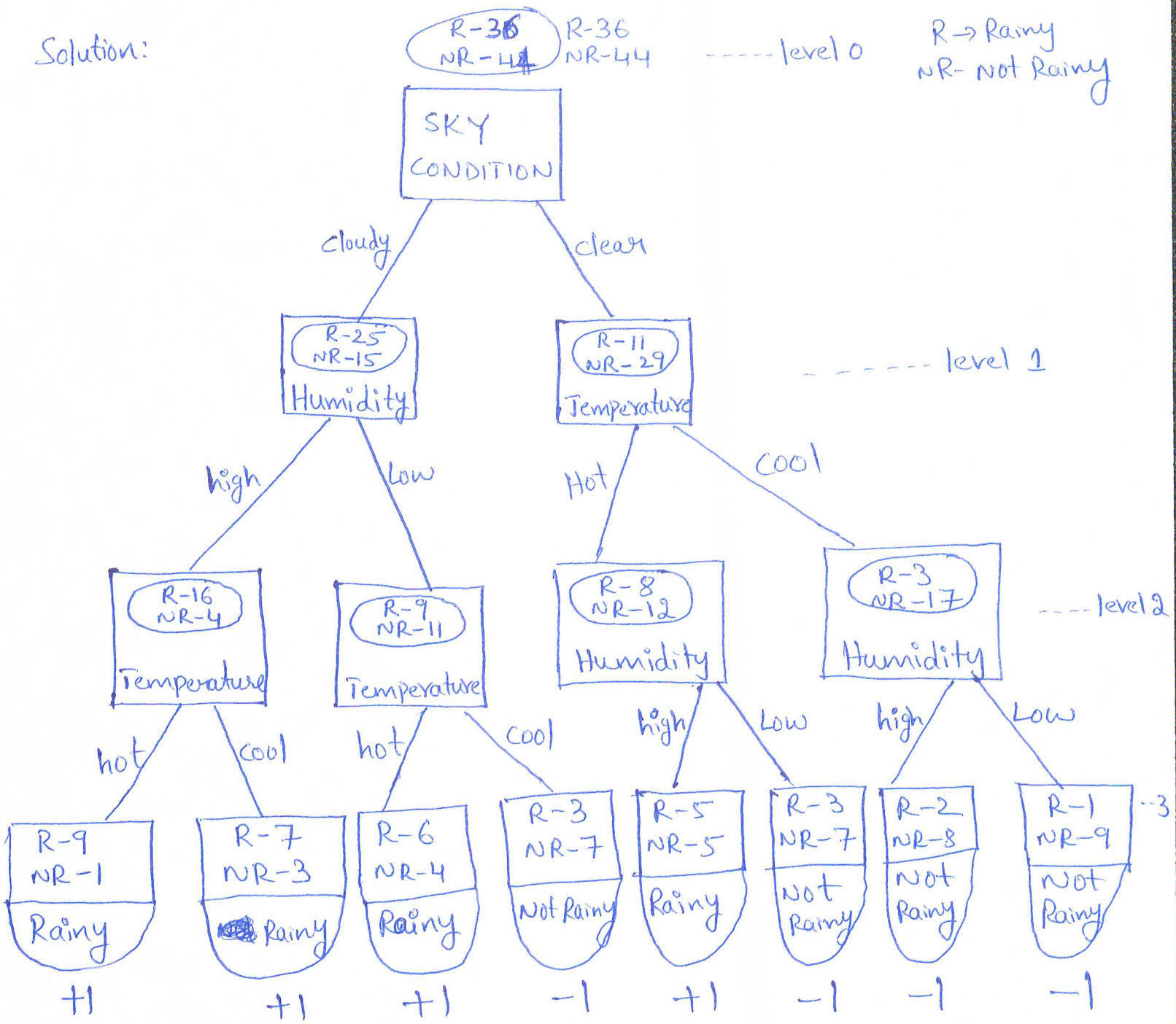
Homework # 2

Fall 2017

Pentyala, Shiva Kumar UIN: 127003995
10-23-2017

Question-1: Decision Tree

Solution:



If we assume labels $\{+1, -1\}$ for Rainy and not Rainy respectively.

As the attributes are not sufficient to expand the tree, In the leaf layer classes are assigned based on Majority voting.

Algorithm:

```
ID3 (Examples, Target_Attribute, Attributes)
    Create a root node for the tree
    If all examples are positive, Return the single-node tree Root, with label = +.
    If all examples are negative, Return the single-node tree Root, with label = -.
    If number of predicting attributes is empty, then Return the single node tree Root,
    with label = most common value of the target attribute in the examples.
    Otherwise Begin
        A ← The Attribute that best classifies examples.
        Decision Tree attribute for Root = A.
        For each possible value, v_i, of A,
            Add a new tree branch below Root, corresponding to the test A = v_i.
            Let Examples(v_i) be the subset of examples that have the value v_i for A
            If Examples(v_i) is empty
                Then below this new branch add a leaf node with label = most common target value in the examples
            Else below this new branch add the subtree ID3 (Examples(v_i), Target_Attribute, Attributes - {A})
    End
    Return Root
```

Calculations:

Information gain (I) = Entropy (Parent) − [average weighted Entropy of children]

The Entropy of a random variable X is given as:

$$H(X) = -\sum P(x_i) \log_2 (P(x_i))$$

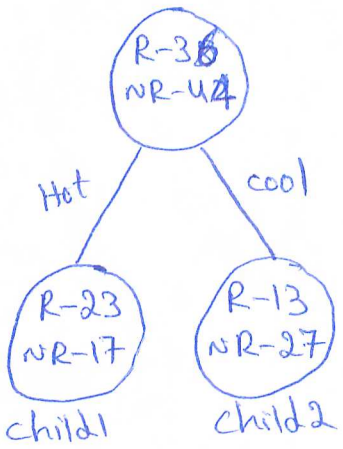
X is a discrete random variable with $\{x_1, \ldots x_N\}$ outcomes, each occurring with probability $P(x_1), \ldots P(x_N)$.

H_{L-0} = Initial Entropy at level 0 = $-\left(\frac{38}{80}\log_2\left(\frac{38}{80}\right) + \frac{42}{80}\log_2\left(\frac{42}{80}\right)\right)$

↓ complete training data

$= 0.99820$ bits

If Root is temperature:



$$H_{C_1} = H_{child1} = -\left[\frac{23}{40}\log\left(\frac{23}{40}\right) + \frac{17}{40}\log\left(\frac{17}{40}\right)\right] = 0.98371$$

$$H_{C_2} = H_{child2} = -\left[\frac{13}{40}\log\left(\frac{13}{40}\right) + \frac{27}{40}\log\left(\frac{27}{40}\right)\right] = 0.90974$$

weighted average H (children) $= \frac{1}{2}\left(H_{C_1} + H_{C_2}\right)$

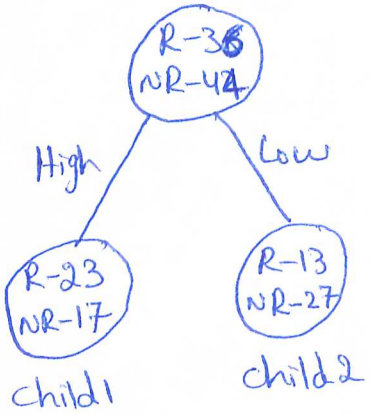
$$\Rightarrow H_{avg} = 0.946725$$

Information gained $= I = H_{L-O} - H_{avg}(\text{children})$

$$= 0.99820 - 0.946725$$

$$\Rightarrow I = 0.051475 \longrightarrow (1)$$

If Root is Humidity:



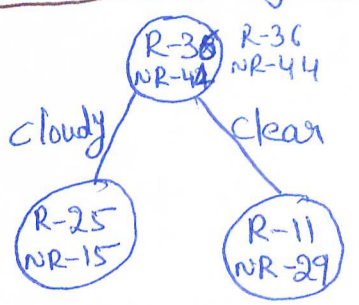
$$H_{C_1} = -\left[\frac{23}{40}\log\left(\frac{23}{40}\right) + \frac{17}{40}\log\left(\frac{17}{40}\right)\right] = 0.98371$$

$$H_{C_2} = -\left[\frac{13}{40}\log\left(\frac{13}{40}\right) + \frac{27}{40}\log\left(\frac{27}{40}\right)\right] = 0.90974$$

$$H_{avg} = 0.946725$$

$$I = 0.99820 - 0.946725 = 0.051475 \Rightarrow (2)$$

If Root is Sky Condition:



$$H_{C_1} = -\left(\frac{25}{40}\log\left(\frac{25}{40}\right) + \frac{15}{40}\log\left(\frac{15}{40}\right)\right) = 0.95443$$

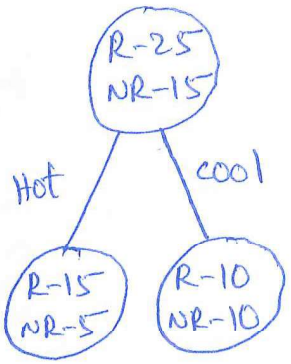
$$H_{C_2} = -\left(\frac{11}{40}\log\left(\frac{11}{40}\right) + \frac{29}{40}\log\left(\frac{29}{40}\right)\right) = 0.84855$$

$$I = 0.99820 - 0.90149 = 0.09671 \longrightarrow (3)$$

From ①, ②, ③ we see that, when Root is Sky Condition we are getting maximum Information Gain.

Now let's move to level-1: It has 2 branches. Consider branch 1 first.

If Level-1, branch-1 Splitting is Temperature:



$H_{c_1} = 0.81128$; $H_{c_2} = 1$

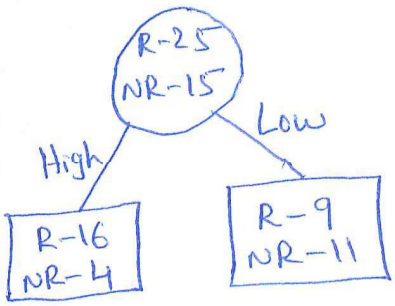
$$I = H_{\text{level-1, branch-1 (parent)}} - \frac{1}{2}\left[H_{c_1} + H_{c_2}\right]$$

$$H_{\text{Level-1, branch-1 (Parent)}} = -\left[\frac{25}{40}\log\left(\frac{25}{40}\right) + \frac{15}{40}\log\left(\frac{15}{40}\right)\right]$$

$$= 0.95443 \text{ bits}$$

$$I = 0.95443 - 0.90564 = 0.04879 \rightarrow (4)$$

If level-1, branch-1 splitting is Humidity:



$H_{c_1} = 0.72193$; $H_{c_2} = 0.99277$

$I = 0.95443 - 0.85735 = 0.09708 \rightarrow (5)$

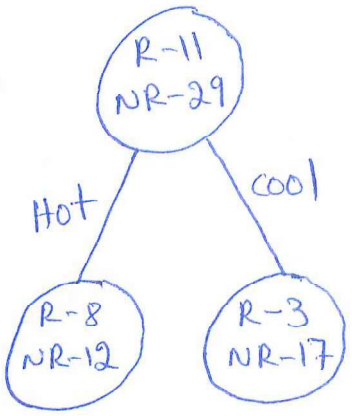
⇒ If Level-1, branch-1 splitting is Humidy then I is more.

Consider Level-1 - branch-2:

$$H_{\text{Level-1, branch-2}} = -\left[\frac{11}{40}\log\left(\frac{11}{40}\right) + \frac{29}{40}\log\left(\frac{29}{40}\right)\right]$$

$= 0.84855$

If Level-1, branch-2 splitting is Temperature:

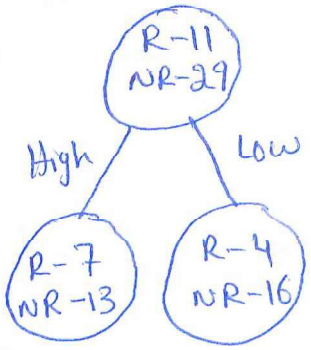


$$H_{C_1} = -\left[\frac{8}{20}\log\frac{8}{20} + \frac{12}{20}\log\frac{12}{20}\right] = 0.97095$$

$$H_{C_2} = -\left[\frac{3}{20}\log\frac{3}{20} + \frac{17}{20}\log\frac{17}{20}\right] = 0.60984$$

$$I = 0.84855 - 0.790395 = 0.058155 \rightarrow (6)$$

If Level-1, branch-2 splitting is Humidity:



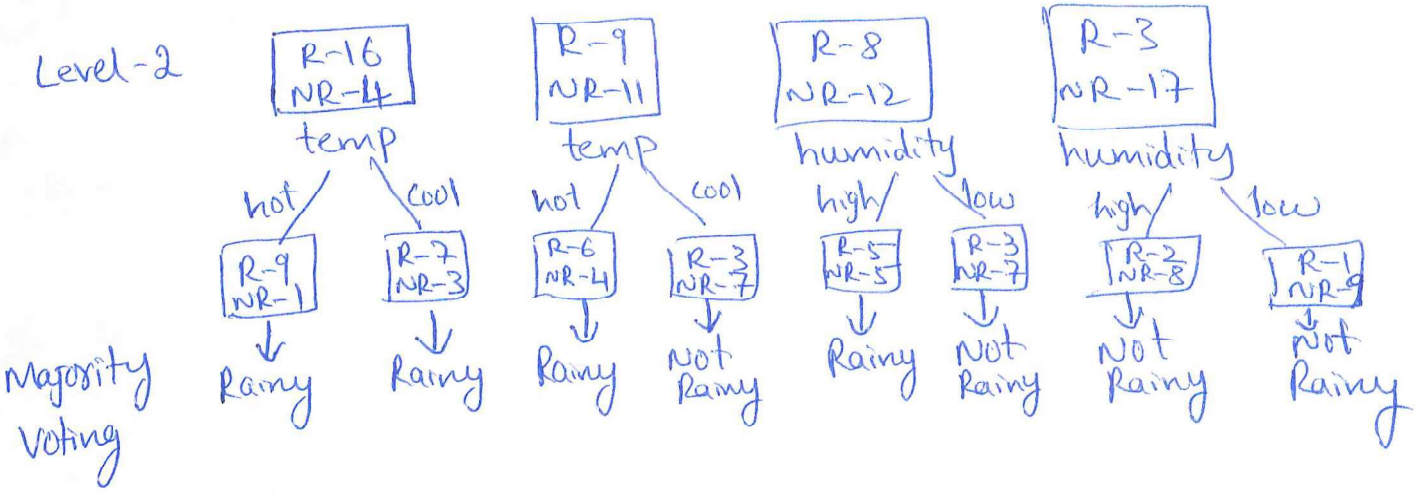
$H_{C_1} = 0.93407$ bits ; $H_{C_2} = 0.72193$ bits

$$I = 0.84855 - 0.828 = 0.02055 \rightarrow (7)$$

we can see that I is high if splitting is based on Temperature.

Now let's move to Level-3:

In level-3, there are 4 nodes. According to the given algorithm for splitting node1, node2 should be Temperature and node3, node4 have to be humidity as they are only left out attributes.



1b) Let X be a discrete random variable with $\{x_1, \cdots x_K\}$ outcomes, each occuring with probability $P(x_1), \cdots P(x_K)$. The Entropy of the random variable X is:

$$H(X) = -\sum_{i=1}^{K} P(x_i) \log\left(P(x_i)\right) = -\sum_{i=1}^{K} P_i \log P_i$$

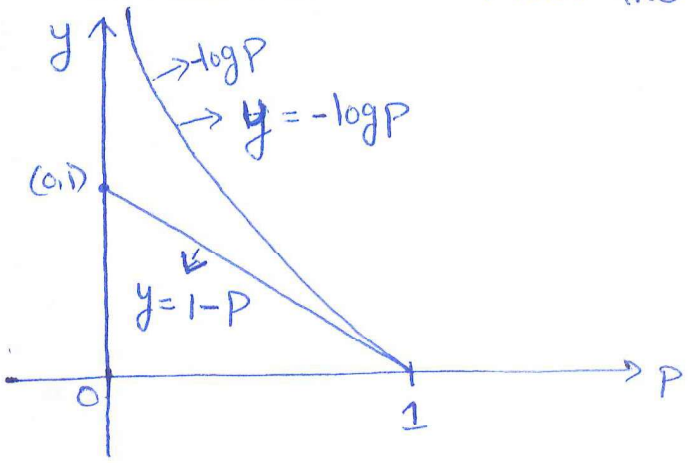
Gini Index, $$\phi(X) = \sum_{i=1}^{K} P(x_i)\left(1 - P(x_i)\right) = \sum_{i=1}^{K} P_i(1-P_i)$$

$$H(X) = \sum P(-\log P) \; ; \; \phi = \sum P(1-P) \quad \text{where } 0 \le P \le 1$$

compare these 2 terms

when $P=0 \Rightarrow H(X)=0 \; ; \; \phi(X)=0$
when $P=1 \Rightarrow H=0 \; ; \; \phi=0$
$\Rightarrow$ It's sufficient to show that $1-P \le -\log P \; \forall P \in [0,1]$

For $P \in (0,1)$:



From the below Figure, $\boxed{H(X) \ge \phi(X) \text{ for } P \in [0,1]}$

Alternatively,

Consider: $f(P) = 1 - P + \log P$

$$f'(P) = -1 + \frac{1}{P}$$

when $P \in (0,1) \Rightarrow f'(P) > 0$

$\Downarrow$

f is an increasing function

$\Rightarrow$ For $P \in (0,1) \Rightarrow f(P) < f(1)$

$\Rightarrow$ For $P \in [0,1] \Rightarrow 1 - P + \log P \le 0$

$\Rightarrow 1 - P \le -\log P$. Hence Proved

Question-2: Kernel Ridge Regression

$$J(w) = (y - Xw)^T (y - Xw) + \lambda \|w\|_2^2$$

$$= y^T y - 2w^T (X^T y) + w^T (X^T X) w + \lambda w^T w$$

$$= y^T y - 2w^T (X^T y) + w^T (X^T X + \lambda I_{D \times D}) w$$

Given that, Non linear mapping is applied to each feature x_n.

$$\Rightarrow \quad x_n \rightarrow \phi_i = \phi(x_n) \in R^T \; ; \; x_n \in R^D \; ; \; T \gg D$$

So, In the above expression X will be replaced with a new matrix ϕ which is, $\phi \in R^{N \times T}$ where N is the total number of samples.

$$\Rightarrow J(w) = y^T y - 2w^T (\phi^T y) + w^T (\phi^T \phi + \lambda I_{T \times T}) w$$

$$\frac{\partial J(w)}{\partial w} = -2\phi^T y + 2(\phi^T \phi + \lambda I_{T \times T}) w$$

optimization $\rightarrow \frac{\partial J(w)}{\partial w} = 0$

$$\Rightarrow w(\phi^T \phi + \lambda I_{T \times T}) = \phi^T y$$

$$\Rightarrow w^* = (\phi^T \phi + \lambda I_{T \times T})^{-1} \phi^T y$$

$$w^* = (\phi^T \phi + \lambda I_{T \times T})^{-1} \phi^T y$$

Matrix Inversion lemma is given by,

$$\left(P^{-1} + B^T R^{-1} B\right)^{-1} B^T R^{-1} = PB^T \left(BPB^T + R\right)^{-1}$$

Applying this lemma to our expression, and using Identity matrix properties, $I^{-1} = I$ and $A \cdot B = AI \cdot B$; $AI = IA = A$

$$\Rightarrow w^* = \left(B^T R^{-1} B + \lambda P^{-1}\right)^{-1} B^T R^{-1} y$$

$$= PB^T \left(BPB^T + R\right)^{-1} y$$

where $\phi = B \in R^{N \times T}$

$I_{T \times T} = P \in R^{T \times T}$

$I_N = R \in R^{N \times N}$

$$= B^T \left(BB^T + \lambda R\right)^{-1} y$$

$$\boxed{w^* = \phi^T \left(\phi \phi^T + \lambda I_N\right)^{-1} y}$$

2-b) we finally need to show that we never actually need access to the feature vectors, which could be infinitely long (which would be rather impractical). what we need in practice is the predicted value for a new test point or just the ~~dot~~ dot product. This is computed by projecting it on the solution w,

$$y = w^{*T} \phi(x) = \left[\phi^T \left(\phi \phi^T + \lambda I_N\right)^{-1} y\right]^T \phi(x)$$

$$= y^T \left(\phi^T \phi + \lambda I_N\right)^{-1} \phi^T \phi(x)$$ [using $(AB)^T = B^T A^T$; $(A^{-1})^T = (A^T)^{-1}$]

$$\Rightarrow y = y^T \left(\phi^T \phi + \lambda I_N\right)^{-1} \phi^T \phi(x)$$

Kernel matrix K, where $K \in R^{N \times N}$ is defined as:

$$k_{ij} = \phi_i^T \phi_j \; ; \; (k(x))_n = \phi^T(x_n)\, \phi(x)$$

$$\Rightarrow \boxed{y = y^T \left(K + \lambda I_N\right)^{-1} k(x)}$$

Bonus: 2-C)

Linear ridge regression: closed form solution is: $w^* = \left(X^T X + \lambda I_{D \times D}\right)^{-1} X^T y$

$$w^* = \left(X^T X + \lambda I_{D \times D}\right)^{-1} X^T y$$

where $X \in R^{N \times D}$

Multiplying $X^T X \rightarrow O(ND^2)$

Inverting $X^T X + \lambda I_D \longrightarrow O(D^3)$

Multiplying $X^T y \rightarrow O(ND)$

Multiplying $\left(X^T X + \lambda I_{D \times D}\right)^{-1}$ with $X^T y \longrightarrow O(D^2)$

so, asymptotically $O(D^3)$ dominates everything else if $D > N$

$\Rightarrow$ Computational complexity of Linear ridge regression $\rightarrow O(D^3)$ if $D > N$ and if $N > D$ then its $O(ND^2)$

Kernel ridge regression:

$$y = y^T (K + \lambda I_N)^{-1} K(x) \quad ; \quad \beta = (\lambda I + K)^{-1} y$$

here we need to compute $N \times N$ matrix but not $D \times D$ matrix

So, whenever we move to any Infinite Dimensional feature space also, we just compute $N \times N$ matrix Inversion which is Independent of the order of Dimensionality.

Time Complexity $= O(N^3)$

← END OF MATHEMATICAL PROBLEMS :) →

Q-1 c) Classifying benign vs malignant tumors:

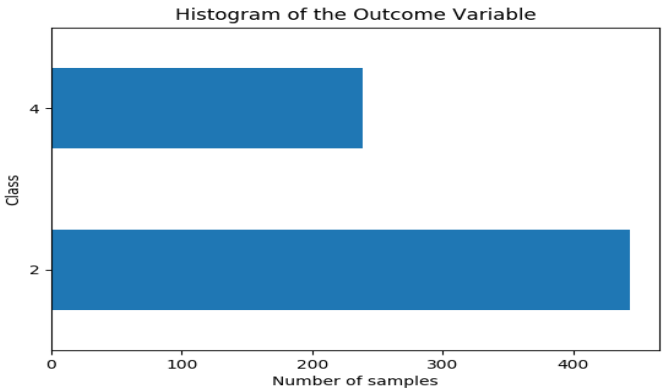
(i) Compute the number of samples belonging to the benign and the number of samples belonging to the malignant case. What do you observe? Are the two classes equally represented in the data? Separate the data into a train (2/3 of the data) and a test (1/3 of the data) set. Make sure that both classes are represented with the same proportion in both sets.

Sol. Total Number of Samples given= 683

Number of samples belonging to the benign = 444 (label -2)

Number of samples belonging to the malignant = 239 (label – 4)

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
>
2   444
4   239
Name: class, dtype: int64
```



Clearly, we can see that Number of samples belonging to the malignant and benign are different. They are not equally represented. Number of samples belonging to benign (label -2) are higher.

Train, Test Splitting:

Data	label - 4	label - 2	Ratio of 2 and 4
Train = 455 (=2/3 of the data)	159	296	1.8616
Test = 228 (=1/3 of the data)	80	148	1.85

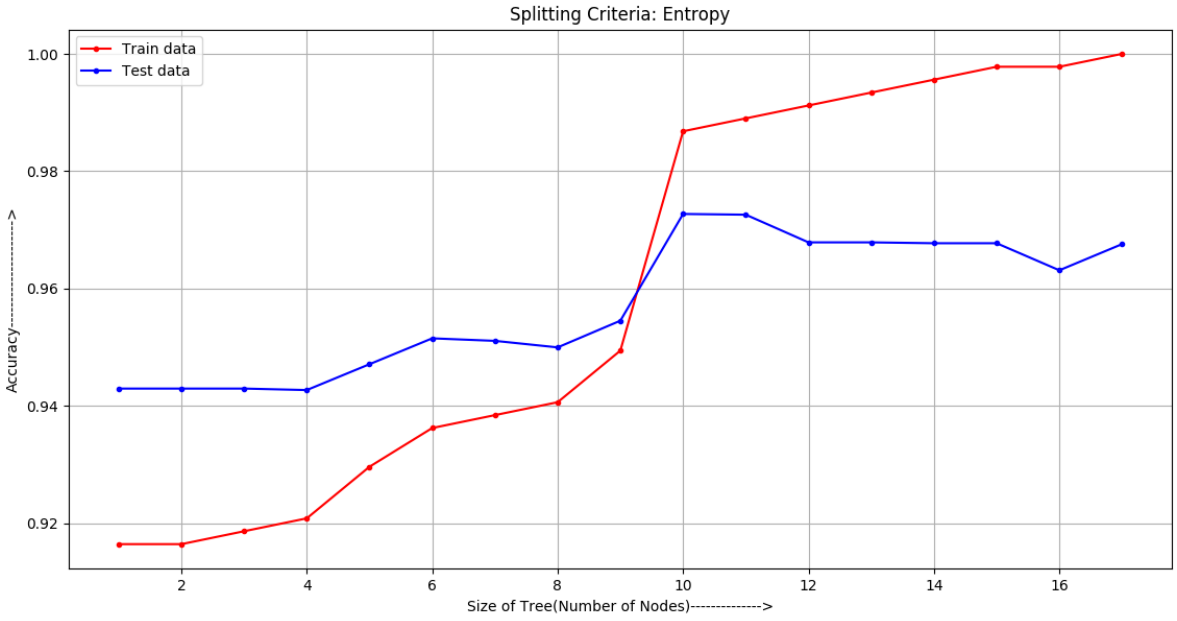
(c.ii) Implement two decision trees using the training samples. The splitting criterion for the first one should be the entropy, while for the second one should be the Gini index. Plot the accuracy on the train and test data while the number of nodes in the tree increases for both splitting criteria. Do you observe any differences in practice?

Algorithm:

```
ID3 (Examples, Target_Attribute, Attributes)
    Create a root node for the tree
    If all examples are positive, Return the single-node tree Root, with label = +.
    If all examples are negative, Return the single-node tree Root, with label = -.
    If number of predicting attributes is empty, then Return the single node tree Root,
    with label = most common value of the target attribute in the examples.
    Otherwise Begin
        A ← The Attribute that best classifies examples.
        Decision Tree attribute for Root = A.
        For each possible value, v_i, of A,
            Add a new tree branch below Root, corresponding to the test A = v_i.
            Let Examples(v_i) be the subset of examples that have the value v_i for A
            If Examples(v_i) is empty
                Then below this new branch add a leaf node with label = most common target value in the examples
            Else below this new branch add the subtree ID3 (Examples(v_i), Target_Attribute, Attributes – {A})
    End
    Return Root
```

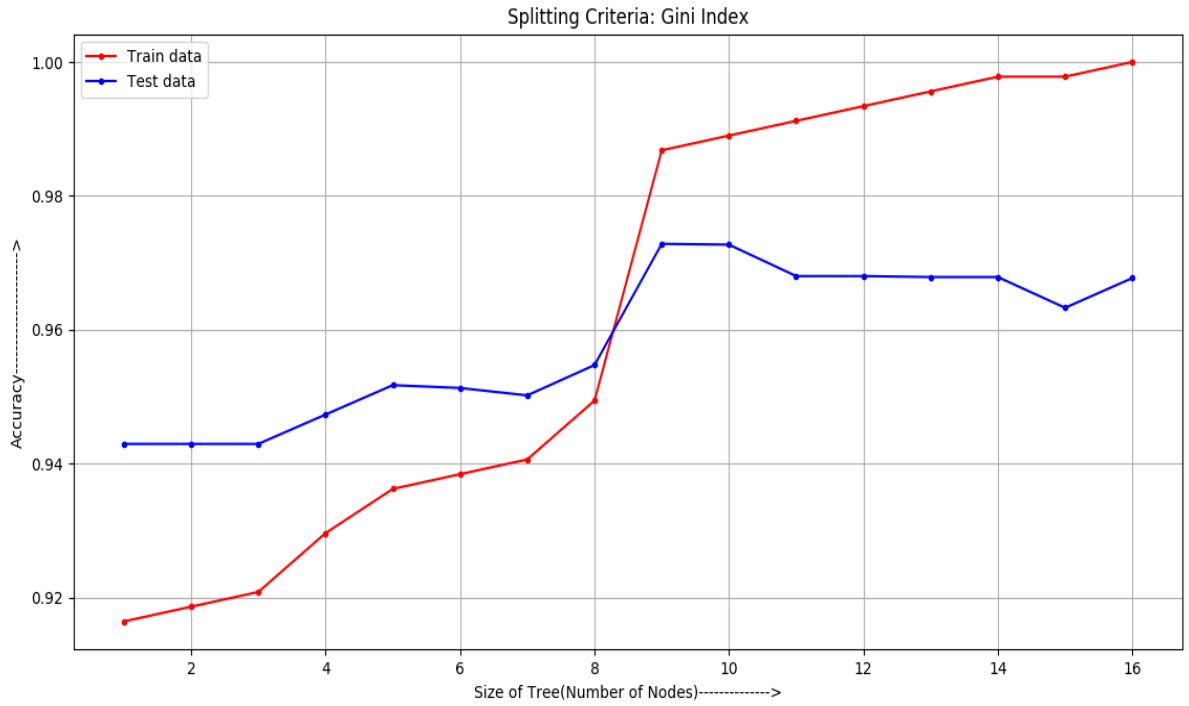
Plots:

Entropy Plot:



```
C:\Users\MBD\AppData\Local\Programs\Python\Python36\python.exe
[0.9429824561403509, 0.9429824561403509, 0.9429824561403509, 0.9427312775330396, 0.947136563876652, 0.9515418502202643, 0.9511111111111111, 0.95, 0.9545454545454546, 0.
9727272727272728, 0.9726027397260274, 0.9678899082568807, 0.9678899082568807, 0.967741935483871, 0.967741935483871, 0.9631336405529954, 0.9675925925925926]
>>>
```

Gini Index Plot:




```
C:\Users\MBD\AppData\Local\Programs\Python\Python36\python.exe
[0.9429824561403509, 0.9429824561403509, 0.9429824561403509, 0.9473684210526315, 0.9517543859649122, 0.9513274336283186, 0.9502262443438914, 0.9547511312217195, 0.97285
06787330317, 0.9727272727272728, 0.9680365296803652, 0.9680365296803652, 0.9678899082568807, 0.9678899082568807, 0.963302752293578, 0.967741935483871]
>>>
```

Conclusion: Gini Index produced little Higher Accuracies than that of the Entropy based splitting

Source code:

As suggested by Dr. Chaspari, I'm only pasting the most important parts of the code. For full code, please visit my GitHub link

Decision Tree:

```
class DecisionTree():

    def learn(self, training_set, attributes, target):
        self.tree_full = build_tree(training_set, attributes, target)
    def learn_nodewise(self, training_set, attributes, target,
node_count,best_count):
        self.tree_p = Nodewise_acc_tree(training_set, attributes,
target,node_count,best_count)
```

```
def build_tree(data, attributes, target):  #target is the last class column
name, data is the current data at that level

    data = data[:]
    vals = [record[attributes.index(target)] for record in data] #values of clas
    default = majorClass(attributes, data, target) #majority voting

    if not data or (len(attributes) - 1) <= 0:
        return default
    elif vals.count(vals[0]) == len(vals): #if all values are same
    else:
        best = attr_choose(data, attributes, target)    #gives best attribute
        best_fulltree.append(best)
        tree = {best:{}}

        for val in get_values(data, attributes, best):  #best has how many
possible values -  val in ['hot','cool']
            new_data = get_data(data, attributes, best, val)  #gives new node
data.like [[],[],[],[]]
            newAttr = attributes[:]
            newAttr.remove(best)
            subtree = build_tree(new_data, newAttr, target)
            tree[best][val] = subtree

    return tree
```

Information gain and Entropy:

```
def attr_choose(data, attributes, target): #chooses best attribute based on
information gain

    best = attributes[0]
    maxGain = 0;

    for attr in attributes[:-1]:  #for each attribute information gain is
calculated
        newGain = info_gain(attributes, data, attr, target)
        if newGain>maxGain:
            maxGain = newGain
            best = attr

    return best
```

```
def info_gain(attributes, data, attr, targetAttr):
    freq = {}
    subsetEntropy = 0.0
    i = attributes.index(attr)

    for entry in data:
        if (entry[i] in freq):
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]]  = 1.0
    for val in freq.keys():
        valProb        = float(freq[val]) / sum(freq.values())
        dataSubset     = [entry for entry in data if entry[i] == val]
        subsetEntropy += valProb * entropy(attributes, dataSubset, targetAttr)

    return (entropy(attributes, data, targetAttr) - subsetEntropy)
```

```
def entropy(attributes, data, targetAttr): #summation of - p log (p) bits
  freq = {}
  dataEntropy = 0.0
  for entry in data:
      if (entry[-1] in freq):
           freq[entry[-1]] += 1.0
      else:
           freq[entry[-1]]  = 1.0

  for freq in freq.values():
    dataEntropy += (-freq/float(len(data)))*math.log((freq/float(len(data))),2)

    return dataEntropy
```

Bonus: Pruning using Lower Threshold and Finding Accuracy as Number of Nodes Increase

```
def Nodewise_acc_tree(data, attributes, target, node_count,best_count):
#how many nodes u want to select in the best
    data = data[:]
    vals = [record[attributes.index(target)] for record in data]  #values
of class
    default = majorClass(attributes, data, target) #majority voting
    if len(best_count)==node_count:
        return default
    if not data or (len(attributes) - 1) <= 0:
        return default
    elif vals.count(vals[0]) == len(vals): #if all values are same then
        return vals[0]
    else:
        best = attr_choose(data, attributes, target)#gives best attribute
        best_count.append(best)
        tree = {best:{}}
        for val in get_values(data, attributes, best):
            new_data = get_data(data, attributes, best, val)
            newAttr = attributes[:]
            newAttr.remove(best)
            subtree = Nodewise_acc_tree(new_data, newAttr,
target,node_count,best_count)
            tree[best][val] = subtree
    return tree
```

```
for node_count in range(1,len(best_fulltree)+1):
        tree.learn_nodewise( training_set, attributes,
target,node_count,best_count=[])
        results = []
        for entry in test_set:  #for each sample go from top root to leaf
            tempDict = tree.tree_p.copy()
            result = ""
            while(isinstance(tempDict, dict)):root =
Node(list(tempDict.keys())[0], tempDict[list(tempDict.keys())[0]])
tempDict = tempDict[list(tempDict.keys())[0]]
                index = attributes.index(root.value)
                value = entry[index]
                if(value in tempDict.keys()):child = Node(value,
tempDict[value])
                    result = tempDict[value]
                    tempDict = tempDict[value]
                else:
                    result = "Null"
                    break
            if result != "Null":
                results.append(result == entry[-1])

        accuracy = float(results.count(True))/float(len(results))
        accuracy_test.append(accuracy)
```

Question 3: Support Vector Machines

(a) Data pre-processing:

Total number of samples given = 11055

If a feature f have value {−1, 0, 1}, we create three new features. Only one of them can have value 1 and $f_{i,x} = 1$ if and only if $f_i = x$.

I transformed the original feature with value 1 into [0, 0, 1]. In the given dataset, the features 2, 7, 8, 14, 15, 16, 26, 29 take three different values {−1, 0, 1}. So each of them is transformed to three 0/1 features.

Source Code:

```
df=pd.read_csv("hw2_question3.csv")
Transform= [2,7,8,14,15,16,26,29]


df2=df.values.tolist()

for i in Transform:
    for j in range(len(df2)):
        if (df2[j][i-1]== -1):
            df2[j][i-1]=[1,0,0]
        elif (df2[j][i-1]==0):
            df2[j][i-1]=[0,1,0]
        else:
            df2[j][i-1]=[0,0,1]
```

As a result, new features will be more in number than that of the original features and now all the features will have either 0 or 1 values.

Now we split the data for training and testing.

Train data = 2/3 of original data

Test data= 1/3 of original data

Now we are ready to perform Linear SVM and Kernel SVM.

(b) Use linear SVM in LIBSVM:

Experimented with different values of misclassification cost C, applying **3-fold** cross validation on the train set.
Best C=**67**
Cross validation accuracy = **94.05698** %
Average training time = **13.05** s
Accuracy on test set using the best C = **94.2741** %

```
optimization finished, #iter = 981941
nu = 0.141879
obj = -34487.999366, rho = -0.277104
nSV = 374, nBSV = 328
Total nSV = 374
Cross Validation Accuracy = 93.7313%
Best C is 67 , Cross Validation Accuracy is 94.056988
Average Training time = 13.051512 seconds
.....................................................
.....................................................
.....................................................
.....................................................
.....................................................
.....................................................
.........................*...........................
optimization finished, #iter = 1109643
nu = 0.137122
obj = -33844.008329, rho = -0.434430
nSV = 539, nBSV = 489
Total nSV = 539
Accuracy = 94.2741% (3474/3685) (classification)
>>>
```

```
def cross_val(y_train,x_train):
    prob  = svm_problem(y_test, x_test)
    max=0
    c_p=0
    for c in range(1,100):
        p=str(c)
        print (p)
        param = svm_parameter('-t 0 -c '+p+' -v 3')
        acc = svm_train(prob,param)
        if acc>max:
            max=acc
            c_p=p
    print ('Best C is %s , Cross Validation Accuracy is %f'%(c_p,max))
    return c_p


def linear_svm(y_test,x_test,c_p): #gives best p
    prob  = svm_problem(y_test, x_test)  #t is 0 for linear svm  #set c
as hyperparameter
    param = svm_parameter('-t 0 -c '+c_p)
    model = svm_train(prob,param)  #v is cross validation 3 fold.
    p_labels, p_acc, p_vals = svm_predict(y_test,x_test, model)
```

(c) <u>Polynomial kernel SVM in LIBSVM:</u>

Polynomial kernel function with degree of d

$$K(\mathbf{x_n}, \mathbf{x_m}) = (\mathbf{x_n}^T \mathbf{x_m} + c)^d$$

for $c \geq 0$ and d a positive integer.

In LIBSVM, 1 – Polynomial kernel is: (gamma*u'*v + coef0)^degree

Parameters: gamma (-g) , coef0 (-r), degree (-d), C. **3 Fold** Cross validation is done.

Best C = **4** ; Best degree = **3** ; Coef0 = 4; g = 4; Cross validation accuracy = **91.6309** %

Accuracy on test set using the best parameters = **92.545** %

```
def cross_val(y_train,x_train):
    prob  = svm_problem(y_test, x_test)
    max=0
    c=d=r=g=1
    for c in range(1,6):
        c=str(c)
        for d in range(1,6):
            d=str(d)
            for r in range(1,5):
                r=str(r)
                for g in range(1,5):
                    g=str(g)
                    param = svm_parameter('-t 1 -c '+c+' -d '+d+' -h 0'+'
-v 3'+' -r '+r+' -g '+g)
                    acc = svm_train(prob,param)
                    if acc>max:
                        max=acc
                        best_c = c
                        best_d=d
                        best_r=r
                        best_g=g
    print ('Best C is %s , Best d is %s, Best r is %s, Best g is %s,
Cross Validation Accuracy is %f'%(best_c,best_d,best_r,best_g,max))
    return [best_c,best_d,best_r,best_s]


def polynomial_svm(y_test,x_test,b_c,b_d,b_r,b_g): #gives best p
    prob  = svm_problem(y_test, x_test)
    param = svm_parameter('-t 1 -c '+b_c+' -d '+b_d+' -h 0 '+' -r '+b_r+'
-g '+b_g)
    model = svm_train(prob,param)  #v is cross validation 3 fold.
    p_labels, p_acc, p_vals = svm_predict(y_test,x_test, model)
p_labels, p_acc, p_vals = svm_predict(y_test,x_test, model)
```

(c) RBF kernel SVM in LIBSVM:

Gaussian kernel, RBF (radial basis function) kernel, or Gaussian RBF kernel

$$K(\mathbf{x_n}, \mathbf{x_m}) = \exp\left(-\frac{\|\mathbf{x_n} - \mathbf{x_m}\|_2^2}{2\sigma^2}\right)$$

parameters to be tuned: d, c, σ^2, etc.

They are Hyperparameters and are often tuned on hold-out data or with Cross-validation.

In LIBSVM, 2 -- radial basis function: exp(-gamma*|u-v|^2)

Parameters: gamma (-g), C ; **3 Fold** Cross validation is done.

Best C = **54** ; Best gamma =**0.1** ; Cross validation accuracy = **95.6309** %

Accuracy on test set using the best C, gamma = **99.403 %**

```
def cross_val(y_train,x_train):
    prob  = svm_problem(y_test, x_test)
    max=0
    c=g=1
    for c in range(1,20):
        c=str(c)
        for g in range(1,20):
            g=str(g)
            param = svm_parameter('-t 2 -c '+c+' -g '+g+' -v 3'+' -h 0')
            acc = svm_train(prob,param)
            if acc>max:
                max=acc
                best_c = c
                best_g=g
    print ('Best C is %s , Best Gamma is %s, Cross Validation Accuracy is
%f'%(best_c,best_g,max))
    return [best_c,best_d]


def polynomial_svm(y_test,x_test,b_c,b_g):
    prob  = svm_problem(y_test, x_test)
    param = svm_parameter('-t 2 -c '+b_c+' -d '+b_g+' -h 0')
    model = svm_train(prob,param)  #v is cross validation 3 fold.
    p_labels, p_acc, p_vals = svm_predict(y_test,x_test, model)
```

```
optimization finished, #iter = 3468
nu = 0.012582
obj = -1773.646167, rho = -0.051947
nSV = 1824, nBSV = 14
Total nSV = 1824
Cross Validation Accuracy = 89.6336%
Best C is 54 , Best Gamma is 0.1, Cross Validation Accuracy is 95.630936
Average Training time = 6.347664 seconds
....*
optimization finished, #iter = 4793
nu = 0.025686
obj = -3763.207897, rho = 0.089777
nSV = 965, nBSV = 37
Total nSV = 965
Accuracy = 99.403% (3663/3685) (classification)
```

Comparison of Linear, RBF with Polynomial Kernel:

What I observed are the following:

C is a trade-off between training error and the flatness of the solution. The larger C is the less the final training error will be. But if you increase C too much you risk losing the generalization properties of the classifier, because it will try to fit as best as possible all the training points (including the possible errors of your dataset). In addition a large C, usually increases the time needed for training.

If C is small, then the classifier is flat (meaning that its derivatives are small - close to zero, at least for the Gaussian RBF kernel this is substantiated theoretically). We have to find a C that keeps the training error small, but also generalizes well (i.e., it doesn't have large fluctuations).

It says how much outliers are taken into account in calculating Support Vectors.

C not too large: The outliners won't affect the decision boundary much.

C large: SVM will try to overfit the decision boundary to the outliners.

If the number of features is large compared to the training sample, we may not need to map data to a higher dimensional space. That is, the nonlinear mapping does not improve the performance. So using the linear kernel is good enough, and hence one only searches for the parameter C.

For our data, we can see that RBF kernel produced better results than Polynomial Kernel. So I would prefer RBF over Polynomial Kernel for this data set.

In conclusion, It seems like squared exponential kernel is generally more flexible than the linear or polynomial kernels so that we can model a whole lot more functions with its function space.

<----------**End**---------->

Thank you so much, Han Wang