CSCE 633: Machine Learning Assignment 2

Sameer Kumar Behera - (UIN: 526004296)

2018/10/11

1 Ridge and Lasso (After Appendix)

2 Decision Trees (After Appendix)

3 Conceptual

The Best Subset, Forward Stepwise, and Backward Stepwise Selection on a single data set was performed. For each approach, we obtain p + 1 models, containing 0; 1; 2; · · · ; p predictors:

(1) The Best Subset Selection has the Smallest Training RSS with k predictors. This is because the approach will be chosen after considering all the possible models with k parameters for best subset. This is not true for either Forward Stepwise or Backward Stepwise Selection. Best Subset Selection exhaustively searches all possible models with k predictors choosing the smallest training RSS while the other two methods heuristically explore a subset of that space, either by starting with the best k-1 model and choosing the best k given a fixed k-1, i.e. in Forward Stepwise Selection or by starting in reverse at the best k+1 and choosing the best single feature to remove and get the best model with that constraint, i.e. the Backward Stepwise Selection.

(2) Out of the three models, it can't be decided which model will have the smallest test RSS with k predictors as it is possible to overfit with any of these models. The Best Subset Selection may have the smallest test RSS as it takes into account more models than the other approaches. However, the other approaches might also pick a model with smaller test RSS by sheer luck.

(3a) TRUE. The (k+1)-variable model identified by Forward Stepwise contains all k predictors chosen in the k-variable model along with the best additional predictor.

(3b) TRUE. The k-variable model identified by Backward Stepwise contains all except one predictor in the (k+1)-variable model which is the only predictor that results in the smallest gain in RSS.

(3c) FALSE. There is no direct link between the models obtained from Forward and Backward Selection as the predictors will be different on both approaches.

(3d) FALSE. There is no direct link between the models obtained from Forward and Backward Selection as the predictors will be different on both approaches.

(3e) FALSE. The (k+1)-variable model is obtained by selecting among all possible models with (k+1) predictors, and thus, does not necessarily contain all the predictors selected for the k-variable model.

4 Classifying Benign vs Malignant Tumors

4.1

Number of Samples Belonging to the Benign Case = 444 Number of Samples Belonging to the Malignant Case = 239 No, the Classes are Not Equally Represented in the Dataset. The data was separated into train data (2/3 of the data) and a test data (1/3 of the data) in such a manner so that both the classes were represented with the same proportion in both the sets.

4.2

The Decision Trees are implemented from scratch and also compared against the DecisionTreeClassifier and the performance is found to be fairly similar for the complete models. Maximum number of nodes is used as the parameterto build the decision tree. A flag is used to to switch from Gini Index to Information Gain based classification.
The Accuracy Plots from the Entropy and Gini Index based Implementations:

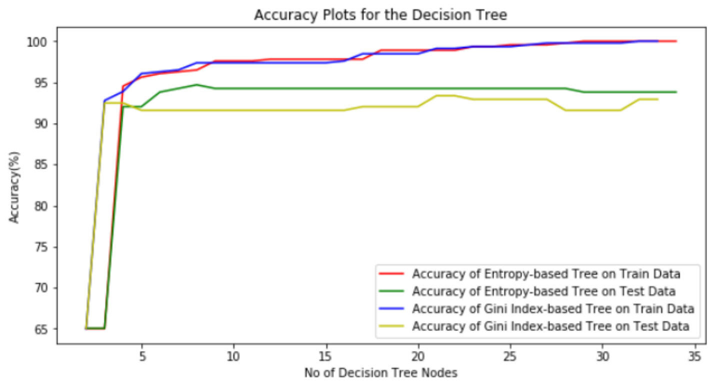


Figure 1: Accuracy Vs Decision Nodes

Clearly on analysis, we can see that the Decision Tree overfits when the number of nodes are increased beyond 10. In other words, the model becomes more biased towards the training data and loses the performance on the test data.

4.3

A 10-fold stratified cross-validation was done for the problem with a random forest model. Also, a secondary 10-fold cross-validation was performed too on the training set in each fold to tune the number of trees given as (10-20) and their maximum depths given as (2-7) in the model. The model got tuned on the inner-validation-fold and later the outer-test-fold was used to acquire the final accuracy.

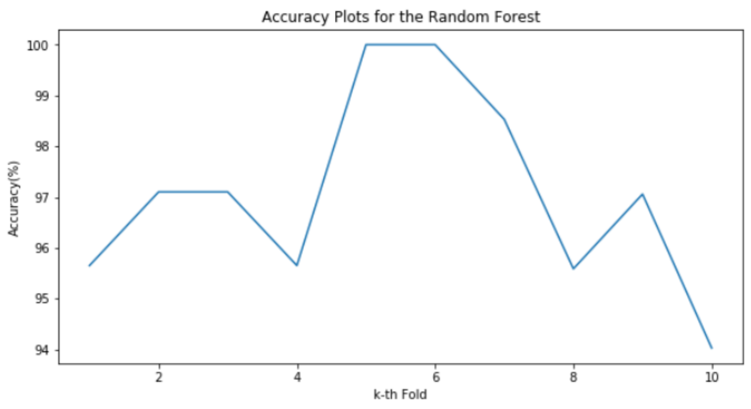
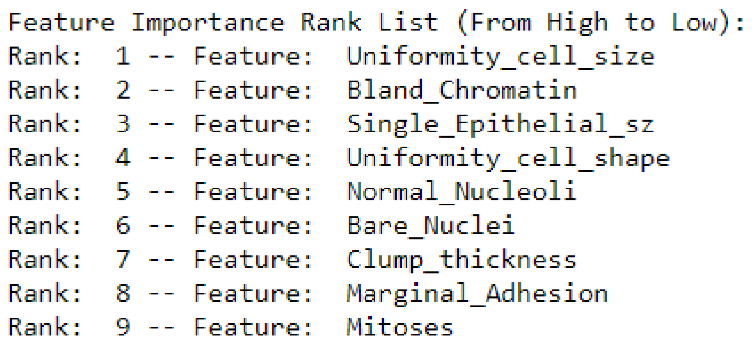


Figure 2: Accuracy across Different Folds



```
Feature Importance Rank List (From High to Low):
Rank:  1 -- Feature:  Uniformity_cell_size
Rank:  2 -- Feature:  Bland_Chromatin
Rank:  3 -- Feature:  Single_Epithelial_sz
Rank:  4 -- Feature:  Uniformity_cell_shape
Rank:  5 -- Feature:  Normal_Nucleoli
Rank:  6 -- Feature:  Bare_Nuclei
Rank:  7 -- Feature:  Clump_thickness
Rank:  8 -- Feature:  Marginal_Adhesion
Rank:  9 -- Feature:  Mitoses
```

Figure 3: Feature Importance Rank List of Model using Random Forest

Even though sklearn.ensemble.RandomForestClassifier uses Gini Index as default criterion to measure the quality of a split, for the list described above, cannot be obtained using Gini Index or Mean Decrease in Accuracy alone. Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does) and so there isn't a single final tree governing the classification results and different trees get activated for different inputs.

The final list of features varies on changing the hyperparameters, i.e. the number of trees and the tree depth, etc. and thus the final model's list of features is quite a relative phenomenon. Also, at times the normal decision tree classification is preferred when better purity is preferred and at times the random forest model is preferred to avoid overfitting.

5 References

https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/
https://medium.com/@rakendd/decision-tree-from-scratch-9e23bcfb4928

6 Appendix

The Python Code for the Benign vs Malignant Tumor Classification Problem is as follows:

```python
# coding: utf-8

# In[56]:


import numpy as np
import pandas as pd
from sklearn import tree
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier


# In[57]:


cols = ['Clump_thickness' , 'Uniformity_cell_size', 'Uniformity_cell_shape',
            'Marginal_Adhesion', 'Single_Epithelial_sz', 'Bare_Nuclei',
            'Bland_Chromatin', 'Normal_Nucleoli', 'Mitoses', 'Class']
df = pd.read_csv(r'E:\Courses\Sem3\Machine Learning\Assignment2\hw2_question1.csv', names = cols)
df.head(10)


# In[58]:


count = df['Class'].value_counts()
print('No of Benign Tumors:', count[2])
print('No of Malignant Tumors:', count[4])


# In[59]:


#Ratio of Class Values
def ratio(my_df):
    count = my_df['Class'].value_counts()
    return count[2]/count[4]


# In[60]:


#Building the Train and Test Sets with Equal Representation
train_factor = 0.67
df = df.sort_values(['Class'])
split1 = (int) (count[2] * train_factor)
split2 = (int) (count[4] * train_factor)

df_train = pd.concat([df.iloc[0:split1,:],df.iloc[count[2]:(count[2]+split2),:]], axis=0)
#print(df_train.shape)
```

```
print("The Ratio in Training Set: ", ratio(df_train))

df_test = pd.concat([df.iloc[split1:count[2],:],df.iloc[(count[2]+split2):,:]], axis=0)
#print(df_test.shape)
print("The Ratio in Testing Set: ", ratio(df_test))


# In[61]:


#Tree Implementation
#Splitting Dataset based on an Attribute and its Value
def test_split(col_idx, threshold, dataset):
    left_group, right_group = list(), list()
    for row in dataset:
        if row[col_idx] < threshold:
            # Left Group Lesser than Threshold
            left_group.append(row)
        else:
            # Right Group Greater than or Equal to Threshold
            right_group.append(row)
    return left_group, right_group

#Gini Index Calculation
def get_gini_index(groups, classes):
    n_instances = float(sum([len(group) for group in groups]))
    gini = 0.0
    for group in groups:
        size = float(len(group))
        if size == 0:
            continue
        score = 0.0
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        gini += (1.0 - score) * (size / n_instances)
    return gini

#Information Gain Calculation
def get_info_gain(groups, classes):
    n_instances = float(sum([len(group) for group in groups]))
    info_gain = 0.0
    for class_val in classes:
        total = 0
        this_class_count = 0
        for group in groups:
            total += len(group)
            this_class_count += [row[-1] for row in group].count(class_val)
        p = this_class_count / total
        if p != 1:
            info_gain += (-1 * p * math.log(1-p,2))

    for group in groups:
        size = float(len(group))
        if size == 0:
            continue
        child_entropy = 0.0
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            if p != 1:
                child_entropy += (-1 * p * math.log(1-p,2))
        info_gain -= (child_entropy) * (size / n_instances)
    return info_gain

#Select Best Split Point
def get_split(dataset, useGini = True):
    class_values = list(set(row[-1] for row in dataset))
    b_score = 999
    b_info_gain = -1
    for index in range(len(dataset[0])-1):
        for row in dataset:
            groups = test_split(index, row[index], dataset)
            if useGini == True:
                gini = get_gini_index(groups, class_values)
                if gini < b_score:
                    b_index, b_value, b_score, b_groups = index, row[index], gini, groups
            else:
                info_gain = get_info_gain(groups, class_values)
                if info_gain > b_info_gain:
                    b_index, b_value, b_info_gain, b_groups = index, row[index], info_gain, groups
    return {'index':b_index, 'value':b_value, 'groups':b_groups}

#Creating a Terminal Node
def to_terminal(group):
    outcomes = [row[-1] for row in group]
    return max(set(outcomes), key=outcomes.count)

#Creating Child Splits at a Node or to Terminate
def split(node, useGini, node_queue):
    left, right = node['groups']
    del(node['groups'])
    if not left or not right :
        node['left'] = node['right'] = to_terminal(left + right)
        return False

    node['left'] = get_split(left)
    node_queue.append(node['left'])
    node['right'] = get_split(right)
    node_queue.append(node['right'])
    return True


#Building Decision Tree
def build_tree(train, max_nodes, useGini):
    node_queue = []
    root = get_split(train, useGini)
    num_nodes = 2
    node_queue.append(root)
    while (len(node_queue) != 0) and (num_nodes < max_nodes):
        this_node = node_queue.pop(0)
        new_node = split(this_node, useGini, node_queue)
        if new_node == True:
            num_nodes += 1


    if len(node_queue) != 0:
        while len(node_queue) != 0:
            this_node = node_queue.pop(0)
            left, right = this_node['groups']
            del(this_node['groups'])
            this_node['left'] = this_node['right'] = to_terminal(left + right)
        return [root, False]
    else:
        return [root, True]

#Make a Prediction with a Decision Tree
def predict(node, row):
    if row[node['index']] < node['value']:
        if isinstance(node['left'], dict):
            return predict(node['left'], row)
        else:
            return node['left']
    else:
        if isinstance(node['right'], dict):
            return predict(node['right'], row)
        else:
            return node['right']

#Calculate Accuracy%
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

#Classification and Regression Tree Algorithm
def decision_tree(train, test, num_nodes, useGini = True):
    [tree, finish] = build_tree(train, num_nodes, useGini)
    train_predictions = list()
    for row in train:
        prediction = predict(tree, row)
        train_predictions.append(prediction)

    test_predictions = list()
    for row in test:
        prediction = predict(tree, row)
        test_predictions.append(prediction)
    return [accuracy_metric(train[:,-1], train_predictions) , accuracy_metric(test[:,-1], test_predi


# In[62]:


# Gini-Index and Entropy based Tree Building
# The Maximum No of Nodes in Tree
max_nodes_entropy = 2
max_nodes_gini = 2
```

```python
entropy_train_acc = []
entropy_test_acc = []
gini_train_acc = []
gini_test_acc = []

while True:
    [train_accuracy, test_accuracy, finish] = decision_tree(df_train.values, df_test.values, max_nod
    gini_train_acc.append(train_accuracy)
    gini_test_acc.append(test_accuracy)
    if finish == True:
        break
    max_nodes_gini += 1

while True:
    [train_accuracy, test_accuracy, finish] = decision_tree(df_train.values, df_test.values, max_nod
    entropy_train_acc.append(train_accuracy)
    entropy_test_acc.append(test_accuracy)
    if finish == True:
        break
    max_nodes_entropy += 1


# In[63]:


plt.figure(figsize=(10,5))
plt.plot(range(2,max_nodes_entropy+1), entropy_train_acc, 'r-', label = "Accuracy of Entropy-based T
plt.plot(range(2,max_nodes_entropy+1), entropy_test_acc, 'g-', label = "Accuracy of Entropy-based Tr
plt.plot(range(2,max_nodes_gini+1), gini_train_acc, 'b-', label = "Accuracy of Gini Index-based Tree
plt.plot(range(2,max_nodes_gini+1), gini_test_acc, 'y-', label = "Accuracy of Gini Index-based Tree
plt.legend(loc='best')
plt.title("Accuracy Plots for the Decision Tree")
plt.xlabel("No of Decision Tree Nodes")
plt.ylabel("Accuracy(%)")
plt.show()


# In[71]:


# K-Fold Primary Stratifified
kfold = StratifiedKFold(n_splits = 10)
test_acc_list = []
feature_imp_list = []

for train_index, test_index in kfold.split(df.iloc[:,:-1], df.iloc[:,-1]):
    df_train, df_test = df.iloc[train_index], df.iloc[test_index]
    #To Select the Best Model on the Secondary K-Fold Data
    best_model = RandomForestClassifier(n_estimators=10, max_depth=2, random_state=0)
    best_acc_validation = 0
    for tree_depth in range(2,7):
        for number_of_trees in range(10,20):
            current_model = RandomForestClassifier(n_estimators=number_of_trees, max_depth=tree_dept
            total_score = 0
            for sec_train_index, sec_test_index in kfold.split(df_train.iloc[:,:-1], df_train.iloc[:
                df_train_inner, df_validate = df_train.iloc[sec_train_index], df_train.iloc[sec_test
                current_model.fit(df_train_inner.iloc[:,:-1], df_train_inner.iloc[:,-1])
                total_score += current_model.score(df_validate.iloc[:,:-1], df_validate.iloc[:,-1])
            if (total_score > best_acc_validation) :
                best_model = current_model
                best_acc_validation = total_score

    test_acc_list.append(best_model.score(df_test.iloc[:,:-1], df_test.iloc[:,-1]))
    feature_imp_list.append(best_model.feature_importances_)


# In[72]:


plt.figure(figsize=(10,5))
plt.plot(range(1, 11), [x*100 for x in test_acc_list])
plt.title("Accuracy Plots for the Random Forest")
plt.xlabel("k-th Fold")
plt.ylabel("Accuracy(%)")
plt.show()


# In[73]:


score = np.zeros((df.shape[1] - 1, 2))
for i in range(df.shape[1] - 1):
    score[i,0] = i
for arr in feature_imp_list:
    for i in range(df.shape[1] - 1):
        score[i,1] += arr[i]
rank = score[score[:,1].argsort()[::-1][:df.shape[1] - 1]]
final_rank = rank[:,0].astype(int)
#print(final_rank)
print("Feature Importance Rank List (From High to Low): ")
for i in range(df.shape[1] - 1):
    print('Rank: ', i + 1, '-- Feature: ', cols[final_rank[i]])
```

Q-2 For the ~~Lasso~~ Ridge Regression Problem, we are given that:

$\hat{B}_0 = 0$, $n=2$, $p=2$, $x_{11} = x_{12}$, $x_{21} = x_{22}$ & $y_1 = -y_2$, $x_{11} = -x_{21}$ & $x_{12} = -x_{22}$

$\boxed{a)}$ A general form of Ridge Regression Optimization Looks Like:

① — To Minimize: $\sum_{i=1}^{n}\left(y_i - \hat{B}_0 - \sum_{j=1}^{p} \hat{B}_j x_j\right)^2 + \lambda \sum_{i=1}^{p} \hat{B}_j^2$

In this case, $\hat{B}_0 = 0$ and $n=p=2$.

Thus, the Optimization Looks Like:

② — To Minimize: $\left(y_1 - \hat{B}_1 x_{11} - \hat{B}_2 x_{12}\right)^2 + \left(y_2 - \hat{B}_1 x_{21} - \hat{B}_2 x_{22}\right)^2 + \lambda\left(\hat{B}_1^2 + \hat{B}_2^2\right)$

$\boxed{b)}$ Taking the derivatives w.r.t. $\hat{B}_1$ & $\hat{B}_2$ of the Equation ② above, we get

~~$\hat{B}_1 = x_1 y_1 + x_2 y_2 - \hat{B}_2 (x_1^2 + x_2^2)$~~

③ — $\lambda(\hat{B}_1) = \left(y_1 - x_{11}(\hat{B}_1 + \hat{B}_2)\right) x_{11} + \left(y_2 - x_{22}(\hat{B}_1 + \hat{B}_2)\right) x_{22}$

&

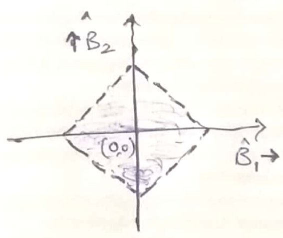
④ — $\lambda(\hat{B}_2) = \left(y_1 - x_{11}(\hat{B}_1 + \hat{B}_2)\right) x_{11} + \left(y_2 - x_{22}(\hat{B}_1 + \hat{B}_2)\right) x_{22}$

On Equating ③ and ④, we get $\hat{B}_1 = \hat{B}_2$

$\boxed{c)}$ The Lasso optimization problem looks like:

$\textcircled{5}$ — To Minimize: $\left(y_1 - \hat{B}_1 x_{11} - \hat{B}_2 x_{12}\right)^2 + \left(y_2 - \hat{B}_1 x_{21} - \hat{B}_2 x_{22}\right)^2$
$+ \lambda\left(|\hat{B}_1| + |\hat{B}_2|\right)$

$\boxed{d)}$ The Lasso Constraint, i.e. $\left(|\hat{B}_1| + |\hat{B}_2|\right)$ takes up the form $|\hat{B}_1| + |\hat{B}_2| < \underset{\text{Fixed Slope}}{S}$, which when plotted takes the shape of a diamond centered at origin $(0,0)$.



Now, Let us consider the Squared Optimization Constraint, i.e. $\left(y_1 - \hat{B}_1 x_{11} - \hat{B}_2 x_{12}\right)^2 + \left(y_2 - \hat{B}_1 x_{21} - \hat{B}_2 x_{22}\right)^2$

Using the given Constraints that:

$$x_{11} = x_{12},\ x_{21} = x_{22},\ x_{11} + x_{21} = 0,$$

$$x_{12} + x_{22} = 0 \ \& \ y_1 + y_2 = 0,$$

the above equation gets simplified to minimize $2\left(y_1 - (\hat{B}_1 + \hat{B}_2)x_{11}\right)^2$

The above optimization problem arrives at a Simple Solution:

$\hat{B}_1 + \hat{B}_2 = \frac{y_1}{x_{11}}$ which is a line parallel to the edge of Lasso-Diamond $\hat{B}_1 + \hat{B}_2 = S$.

Thus, the solutions of the original Lasso optimization problem are contours of the function,

$\left(y_1 - (\hat{B}_1 + \hat{B}_2)x_{11}\right)^2$ that touch the Lasso-Diamond Edge, i.e. $\hat{B}_1 + \hat{B}_2 = S$.

As, $\hat{B}_1$ and $\hat{B}_2$ vary along the line $\hat{B}_1 + \hat{B}_2 = \frac{y_1}{x_{11}}$, these contours touch the Lasso-Diamond Edge $\hat{B}_1 + \hat{B}_2 = S$ at different points. Thus, as a result, the entire edge $\hat{B}_1 + \hat{B}_2 = S$ is a potential solution.

A similar argument can be made for the opposite Lasso-Diamond Edge, i.e. $\hat{B}_1 + \hat{B}_2 = -S$.

Hence, it becomes very clear that the Lasso problem doesn't have a unique solution and rather has many possible solutions

Q-3 Root Entropy is defined as:

$$H = \sum_{i=1}^{\text{No. of Classes}} -P(y=Y_i) \cdot \log P(y=Y_i) \quad \{\text{Log Base} = 2\}$$

As, all the random variables have only 2 discrete values, the Entropy w.r.t. each attribute is defined as:

$$H_{\text{w.r.t. attribute}} = -P(\text{rainy}) \log(P(\text{rainy})) ~~H(P(\text{rainy})~~$$
$$- (1 - P(\text{rainy})) \cdot \log(1 - P(\text{rainy}))$$

$$P(\text{rainy}) \text{ at the Root} = 36/80 = 9/20$$

At Step 1

$$\text{Root Entropy} = \frac{-9}{20} \log\left(\frac{9}{20}\right) + \frac{-11}{20} \log\left(\frac{11}{20}\right) = 0.9927$$

At Step 2 To Check First Split:

Calculating Information Gain from Temperature:

$$H_{\text{Temperature = Hot}} = \frac{-23}{40} \log\left(\frac{23}{40}\right) + \frac{-17}{40} \log\left(\frac{17}{40}\right) = 0.98$$

$$H_{\text{Temperature = Cold}} = \frac{-13}{40} \log\left(\frac{13}{40}\right) + \frac{-27}{40} \log\left(\frac{27}{40}\right) = 0.91$$

$$\text{Information Gain from Temperature} = 0.9927 - \frac{1}{2} \times 0.98 - \frac{1}{2} \times 0.91$$
$$= 0.0477$$

~~Check:~~ Calculating Information Gain from Humidity:

$$H_{Humidity=High} = -\frac{23}{40}\log\left(\frac{23}{40}\right) + -\frac{17}{40}\log\left(\frac{17}{40}\right) = 0.98$$

$$H_{Humidity=Low} = -\frac{13}{40}\log\left(\frac{13}{40}\right) + -\frac{27}{40}\log\left(\frac{27}{40}\right) = 0.91$$

Information Gain from Humidity $= 0.9927 - \frac{1}{2}\times 0.98 - \frac{1}{2} * 0.91$

$= 0.0477$

Calculating Information Gain from Sky:

$$H_{Sky=Cloudy} = -\frac{25}{40}\log\left(\frac{25}{40}\right) + -\frac{15}{40}\log\left(\frac{15}{40}\right) = 0.954$$

$$H_{Sky=Clear} = -\frac{11}{40}\log\left(\frac{11}{40}\right) + -\frac{29}{40}\log\left(\frac{29}{40}\right) = 0.84$$

Information Gain from Sky $= 0.9927 - \frac{1}{2}\times 0.954 - \frac{1}{2}\times 0.84$

$= 0.0957$

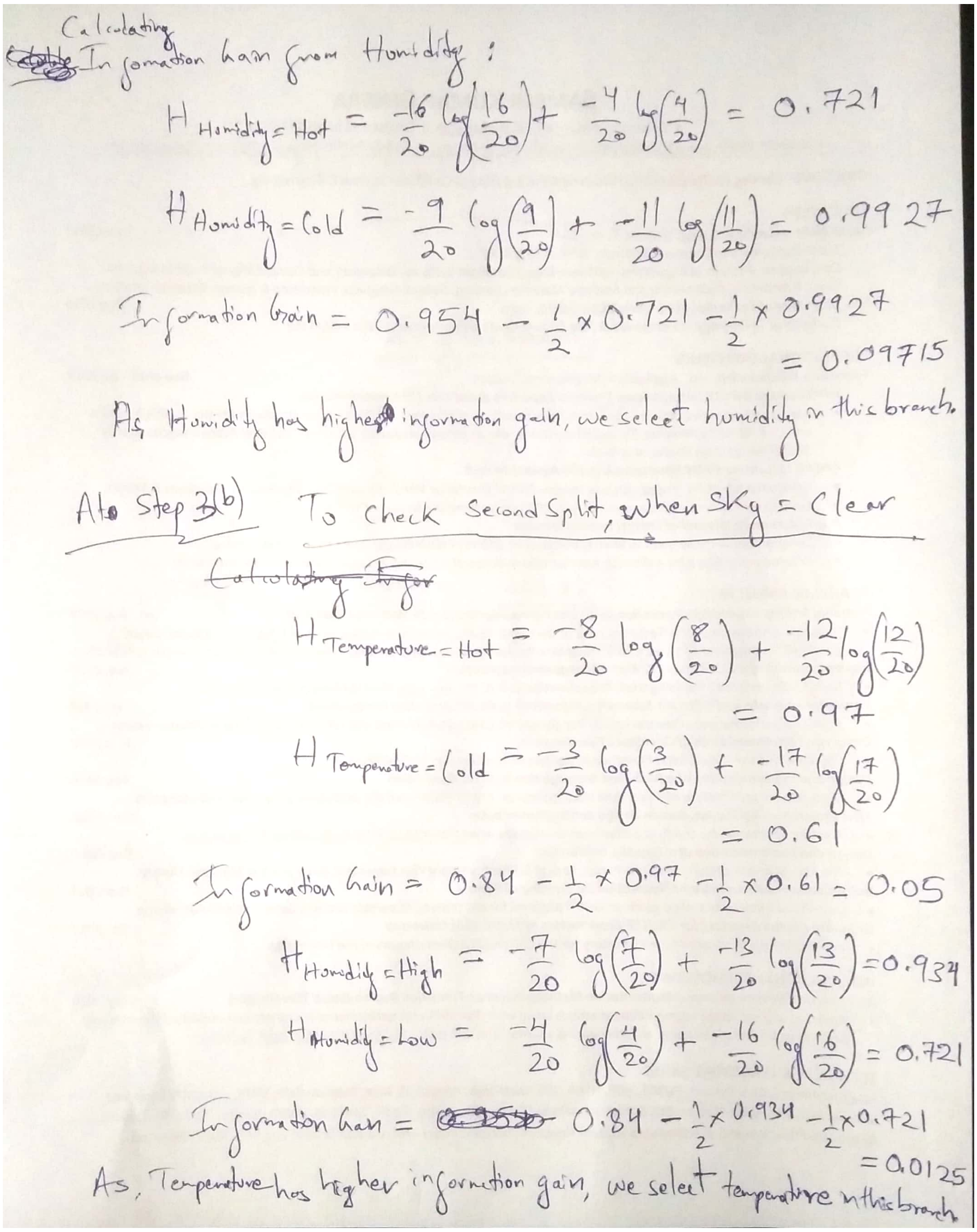
As the Information Gain from Sky is Maximum, we split w.r.t. Sky.

Step-2(a) To Check Second split, ~~[illegible]~~ when $\underbrace{\text{Sky} = \text{Cloudy}}$:

$$H_{Temperature=Hot} = -\frac{15}{20}\log\left(\frac{15}{20}\right) + -\frac{5}{20}\log\left(\frac{5}{20}\right) = 0.811$$

$$H_{Temperature=Cold} = -\frac{10}{20}\log\left(\frac{10}{20}\right) + -\frac{10}{20}\log\left(\frac{10}{20}\right) = 1$$

Information Gain $= 0.954 - \frac{1}{2}\times 0.811 - \frac{1}{2}\times 1 = 0.0485$



Calculating Information Gain from Humidity :

$$H_{Humidity=Hot} = \frac{-16}{20}\log\left(\frac{16}{20}\right) + \frac{-4}{20}\log\left(\frac{4}{20}\right) = 0.721$$

$$H_{Humidity=Cold} = \frac{-9}{20}\log\left(\frac{9}{20}\right) + \frac{-11}{20}\log\left(\frac{11}{20}\right) = 0.9927$$

$$\text{Information Gain} = 0.954 - \frac{1}{2}\times 0.721 - \frac{1}{2}\times 0.9927 = 0.09715$$

As Humidity has highest information gain, we select humidity in this branch.

At Step 3(b) To check Second Split, when Sky = Clear

~~Calculating Infor~~

$$H_{Temperature=Hot} = \frac{-8}{20}\log\left(\frac{8}{20}\right) + \frac{-12}{20}\log\left(\frac{12}{20}\right) = 0.97$$

$$H_{Temperature=Cold} = \frac{-3}{20}\log\left(\frac{3}{20}\right) + \frac{-17}{20}\log\left(\frac{17}{20}\right) = 0.61$$

$$\text{Information Gain} = 0.84 - \frac{1}{2}\times 0.97 - \frac{1}{2}\times 0.61 = 0.05$$

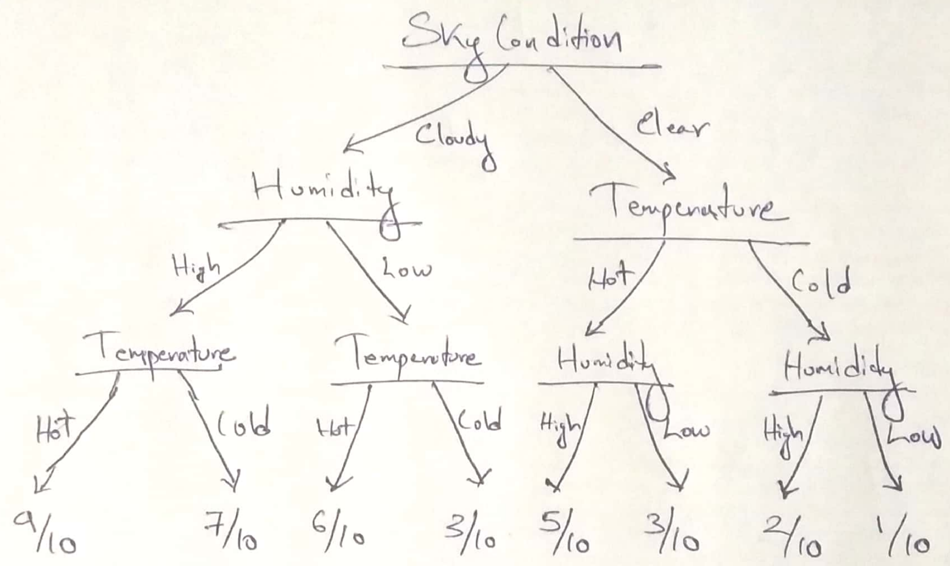
$$H_{Humidity=High} = \frac{-7}{20}\log\left(\frac{7}{20}\right) + \frac{-13}{20}\log\left(\frac{13}{20}\right) = 0.934$$

$$H_{Humidity=Low} = \frac{-4}{20}\log\left(\frac{4}{20}\right) + \frac{-16}{20}\log\left(\frac{16}{20}\right) = 0.721$$

$$\text{Information Gain} = \cancel{0.9550}\; 0.84 - \frac{1}{2}\times 0.934 - \frac{1}{2}\times 0.721 = 0.0125$$

As, Temperature has higher information gain, we select temperature in this branch

The Final Decision Tree:



Sky Condition
Cloudy
Clear
Humidity
Temperature
High
Low
Hot
Cold
Temperature
Temperature
Humidity
Humidity
Hot
Cold
Hot
Cold
High
Low
High
Low
9/10
7/10
6/10
3/10
5/10
3/10
2/10
1/10