UNIVERSITY OF TEHRAN

COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

INTELLIGENT SYSTEM

ASSIGNMENT#2

MOHAMMAD HEYDARI

810197494

UNDER SUPERVISION OF:

DR. RESHAD HOSSINI

ASSISTANT PROFESSOR

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF TEHRAN

Nov. 2021

1 CONTENTS

2 QUESTION #1

2.1 AANALYSIS OF DECISION TREE (THEORETICAL)

In this part we intend to find a decision tree which describes dataset features as well as we supposed, moreover specify whether vascular occlusive disease is likely or not between our subjective.

Note that the final goal of this part is deciding whether a person tend to having vascular occlusive or not, we know that in our investigation there are some considerable factors where abbreviated below:

	Number	blood pressure	Cholesterol levels	smoking	Weight	Arterial occlusion
0	1	Yes	Normal	No	Overweight	Yes
1	2	No	Normal	Yes	Normal	No
2	3	No	Critical	No	Overweight	Yes
3	4	No	High	Yes	Overweight	Yes
4	5	Yes	Critical	Yes	Fat	Yes
5	6	Yes	High	Yes	Normal	Yes
6	7	No	High	No	Fat	No
7	8	Yes	Normal	Yes	Normal	Yes
8	9	Yes	Critical	No	Fat	Yes
9	10	No	Normal	No	Overweight	No
10	11	No	Critical	Yes	Normal	Yes
11	12	Yes	High	No	Overweight	No
12	13	Yes	Normal	Yes	Overweight	Yes
13	14	Yes	High	No	Fat	No

Figure 1: Representation of Dataset

Then, we analyse the dataset and afterward according to above criteria make a decision tree based on **Information Gain** as of our factor to decide why a leaf or node is considered as Top-level of our implementation.

Note that in this part I have used a MATLAB code to calculate entropy and also Information gain.

For more detailed information, please check the related directory where you can easily find my intelligent_system.m & information_Thery.m codes.

Here I have written needed calculations:

First Level:

1) blood pressure:

$$\begin{cases} pos = 9^+, neg = 5^- \\ sub\_pos1 = 6^+, sub\_neg1 = 2^- \\ sub\_pos2 = 3^+, sub\_neg2 = 3^- \end{cases} \xrightarrow{yields} \text{Information Gain=0.0481}$$

2) smoking:

$$\begin{cases} pos = 9^+, neg = 5^- \\ sub\_pos1 = 6^+, sub\_neg1 = 1^- \\ sub\_pos2 = 2^+, sub\_neg2 = 4^- \end{cases} \xrightarrow{yields} \text{Information Gain=0.2509}$$

3) Cholesterol levels:

$$\begin{cases} \text{Cholesterol levels:} \\ Critical \\ High \\ Normal \end{cases}$$

$$\begin{cases} pos = 9^+, neg = 5^- \\ sub\_pos1 = 4^+, sub\_neg1 = 0^- \\ sub\_pos1 = 2^+, sub\_neg1 = 3^- \\ sub\_pos1 = 3^+, sub\_neg1 = 2^- \end{cases} \xrightarrow{yields} \text{Information Gain=0.2467}$$

3) Weight:

$$\begin{cases} \text{Weight:} \\ Fat \\ Overweight \\ Normal \end{cases}$$

$$\begin{cases} pos = 9^+, neg = 5^- \\ sub\_pos1 = 2^+, sub\_neg1 = 2^- \\ sub\_pos1 = 4^+, sub\_neg1 = 2^- \\ sub\_pos1 = 3^+, sub\_neg1 = 1^- \end{cases} \xrightarrow{yields} \text{Information Gain=0.0292}$$

So the top-level is Cholesterol levels.

And it has been divided to three separated part: Critical, High and Normal so moving forward and calculate sub information gain to help us find the next node or leaf.

Second Level:

Cholesterol levels:

Critical:

in This condition the subjective definitely has vascular occlusive so the result is positive!

High:

1) blood pressure:

$$\begin{cases} pos = 2^+, neg = 3^- \\ sub\_pos1 = 1^+, sub\_neg1 = 2^- \\ sub\_pos2 = 1^+, sub\_neg2 = 1^- \end{cases} \xrightarrow{yields}$$ Information Gain=0.02

2) smoking:

$$\begin{cases} pos = 2^+, neg = 3^- \\ sub\_pos1 = 2^+, sub\_neg1 = 0^- \\ sub\_pos2 = 0^+, sub\_neg2 = 3^- \end{cases} \xrightarrow{yields}$$ Information Gain=1

3) Weight:

$$\begin{cases} \text{Weight:} \\ Fat \\ Overweight \\ Normal \end{cases}$$

$$\begin{cases} pos = 2^+, neg = 3^- \\ sub\_pos1 = 0^+, sub\_neg1 = 2^- \\ sub\_pos1 = 1^+, sub\_neg1 = 1^- \\ sub\_pos1 = 1^+, sub\_neg1 = 0^- \end{cases} \xrightarrow{yields}$$ Information Gain=0.78

So the top-level of this part is **Smoking.**

As we can see if the **Cholesterol levels** would be High the **Smoking** is determinative so if the subjective tend to have **Smoking** feature definitely we see that target result is positive and if there is not any **Smoking** feature the target result is negative.

Normal:

1) blood pressure:

$$\begin{cases} pos = 3^+, neg = 2^- \\ sub\_pos1 = 3^+, sub\_neg1 = 0^- \\ sub\_pos2 = 0^+, sub\_neg2 = 2^- \end{cases} \xrightarrow{yields} \text{Information Gain=1}$$

2) smoking:

$$\begin{cases} pos = 3^+, neg = 2^- \\ sub\_pos1 = 2^+, sub\_neg1 = 1^- \\ sub\_pos2 = 1^+, sub\_neg2 = 1^- \end{cases} \xrightarrow{yields} \text{Information Gain=0.02}$$

3) Weight:

$$\begin{cases} \text{Weight:} \\ Fat \\ Overweight \\ Normal \end{cases}$$

$$\begin{cases} pos = 3^+, neg = 2^- \\ sub\_pos1 = 0^+, sub\_neg1 = 0^- \\ sub\_pos1 = 2^+, sub\_neg1 = 1^- \\ sub\_pos1 = 1^+, sub\_neg1 = 1^- \end{cases} \xrightarrow{yields} \text{Information Gain=0.38}$$

So the top-level of this part is **blood pressure.**

As we can see if the **Cholesterol levels** would be Normal the **blood pressure** is determinative so if the subjective tend to have **blood pressure** definitely we see that target result is positive and if there is not any blood pressure the target result is negative.

Here you can see the Whole decision tree which make sense in cases of all features concerned in our dataset and as we can see it's independent of weight attribute based on information Gain factor:

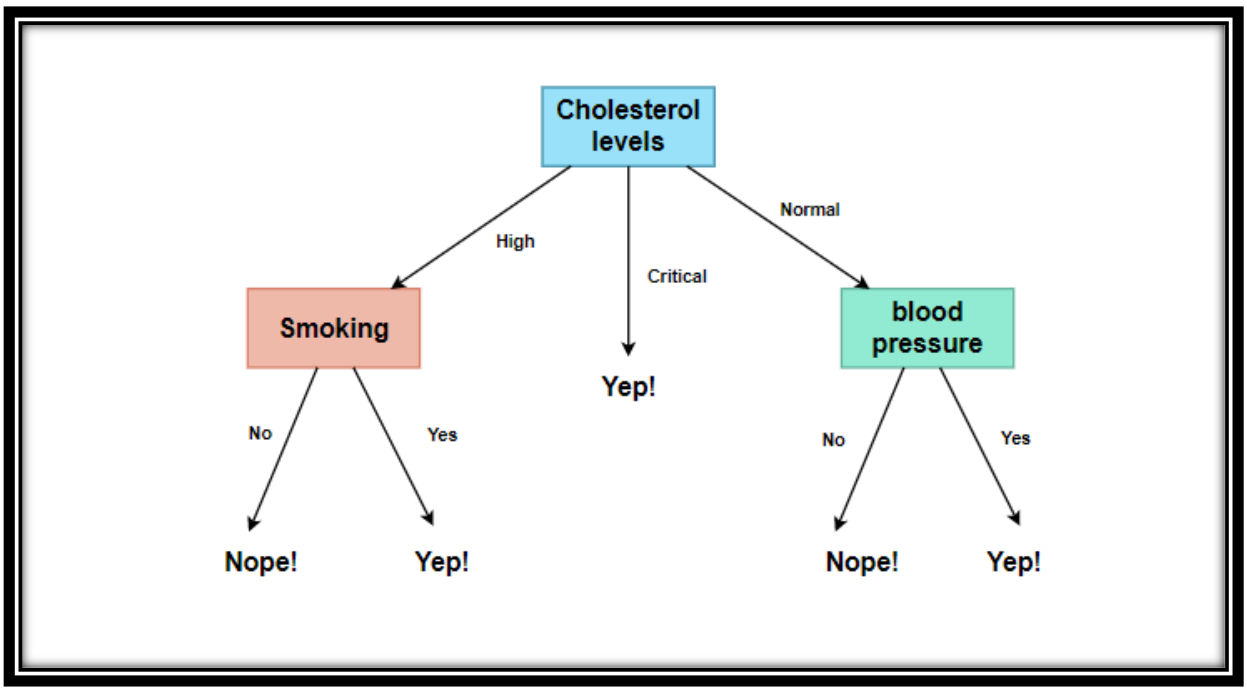


Figure 2: Representation of Decision Tree

For checking this part please have an accurate look on related directory and There you can easily find all of you need included information Gain calculator MATLAB code and also I would appreciate it if you would consider them ☺

2.2 TESTING OUR DECISION TREE

In this part we intend to use Decision tree classification method for predicting the result and investigate whether vascular occlusive disease is likely or not between our test-subjective.

Afterward, we analyse the performance of our model and use confusion matrix to specify our statements as well as it matches with practical cases.

Number	blood pressure	Cholesterol levels	smoking	Weight	Arterial occlusion
15	Yes	Normal	Yes	Fat	Yes
16	Yes	High	Yes	Fat	Yes
17	Yes	High	No	Normal	No
18	Yes	Normal	No	Normal	No
19	No	Normal	Yes	Overweight	Yes

Figure 3: Test Data

Number	blood pressure	Cholesterol levels	smoking	Weight	Arterial occlusion	Predictions
15	Yes	Normal	Yes	Fat	Yes	Yes
16	Yes	High	Yes	Fat	Yes	Yes
17	Yes	High	No	Normal	No	No
18	Yes	Normal	No	Normal	No	Yes
19	No	Normal	Yes	Overweight	Yes	No

Figure 4: Predicted Data

As you can see our model made a mistake when facing with row 18 & 19 as of our test-data but in other cases consist of row 15, 16, and 17 the performance of our tree is nice and reliable.

Now let's determine confusion matrix to have a deeper investigation on the results:

Interpreting our findings:

As we know A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class.

This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by our classifier but more importantly the types of errors that are being made. It is this breakdown that overcomes the limitation of using classification accuracy alone.

"true positive" for correctly predicted event values, and in our case it's **2**

"false positive" for incorrectly predicted event values, and in our case it's **1**

"true negative" for correctly predicted no-event values, and in our case it's **1**

"false negative" for incorrectly predicted no-event values, and in our case it's **1**

Confusion matrix abbreviated below:

$$\begin{bmatrix} TP = 2 & TN = 1 \\ FP = 1 & FN = 1 \end{bmatrix}$$

And at the end, Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} = \frac{2+1}{2+1+1+1} = \frac{3}{5} = 0.6 = 60\%$$

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

2.3 WAYS TO IMPROVE DECISION TREE PERFORMANCE

in this part we express some ways and ideas to help decision tree acts as much as better and improve performance in ways that it can be dominate overfitting problem and eliminate such these stuffs.

Decision Trees are a non-parametric supervised machine learning approach for classification and regression tasks. Overfitting is a common problem, a data scientist needs to handle while training decision tree models. Comparing to other machine learning algorithms, decision trees can easily overfit.

Overfitting refers to the condition when the model completely fits the training data but fails to generalize the testing unseen data. Overfit condition arises when the model memorizes the noise of the training data and fails to capture important patterns. A perfectly fit decision tree performs well for training data but performs poorly for unseen test data.

If the decision tree is allowed to train to its full strength, the model will overfit the training data. There are **various techniques to prevent the decision tree model from overfitting**. Here, we will discuss **3 such techniques.**

1) Pruning

By default, the decision tree model is allowed to grow to its full depth. Pruning refers to a technique to remove the parts of the decision tree to prevent growing to its full depth. By tuning the hyperparameters of the decision tree model one can prune the trees and prevent them from overfitting.

There are two types of pruning Pre-pruning and Post-pruning. Now let's discuss the in-depth understanding.

Pre-Pruning:

The pre-pruning technique refers to the early stopping of the growth of the decision tree. The pre-pruning technique involves tuning the hyperparameters of the decision tree model prior to the training pipeline. The hyperparameters of the decision tree including max-depth, min-samples-leaf, min-samples-split can be tuned to early stop the growth of the tree and prevent the model from overfitting.

Post-Pruning:

The Post-pruning technique allows the decision tree model to grow to its full depth, then removes the tree branches to prevent the model from overfitting. Cost complexity pruning (ccp) is one type of post-pruning technique. In case of cost complexity pruning, the ccp_alpha can be tuned to get the best fit model.

3) Ensemble: Random Forest

Random Forest is an ensemble technique for classification and regression by bootstrapping multiple decision trees. Random Forest follows bootstrap sampling and aggregation techniques to prevent overfitting.

Random Forest can be implemented using the Scikit-Learn library. we can use the hyperparameters of the Random Forest algorithm to improve the performance of the model. n_estimator parameter can be tuned to reduce the overfitting of the model.

3 QUESTION #2

3.1 IMPLEMENTING DECISION TREE (SIMULATION)

In this part we intend to simulate decision tree based on ID3 algorithm with result class of "Return to Prison numeric" parameter.

We consider max depth=3 and also use the 80% data in train cases and rest of that to use in test purposes.

As the question has pointed to a dynamic max-depth parameter in further sections, so we choose max-depth as one of our input argument.

Algorithm:

It is a classification algorithm that follows a greedy approach by selecting a best attribute that yields maximum Information Gain(IG) or minimum Entropy(H).

In ID3, entropy is calculated for each remaining attribute. The attribute with the smallest entropy is used to split the set S on that particular iteration.

Entropy = 0 implies it is of pure class, that means all are of same category.

	Fiscal Year Released	Recidivism Reporting Year	Race - Ethnicity	Age At Release	Convicting Offense Classification	Convicting Offense Type	Convicting Offense Subtype	Main Supervising District	Release Type	Part of Target Population	Recidivism - Return to Prison numeric
0	2010	2013	White	<45	D Felony	Violent	Other	3JD	Parole	Yes	1
1	2010	2013	White	>45	D Felony	Other	Other	3JD	Parole	Yes	1
2	2010	2013	White	<45	D Felony	Other	Other	5JD	Parole	Yes	1
3	2010	2013	White	>45	Other Felony	Drug	Trafficking	3JD	Parole	Yes	1
4	2010	2013	Black	<45	D Felony	Drug	Trafficking	3JD	Parole	Yes	1
...	...	...	...	...	...	...	...	...	...	...	...
15419	2015	2018	White	<45	Other Felony	Violent	Other	3JD	Discharged End of Sentence	Yes	0
15420	2015	2018	White	<45	D Felony	Other	Other	5JD	Discharged End of Sentence	No	0
15421	2015	2018	Black	<45	Other Felony	Violent	Other	3JD	Discharged End of Sentence	Yes	0
15422	2015	2018	White	<45	D Felony	Drug	Other	5JD	Parole	No	0
15423	2015	2018	White	<45	Other Felony	Violent	Other	3JD	Parole	No	0

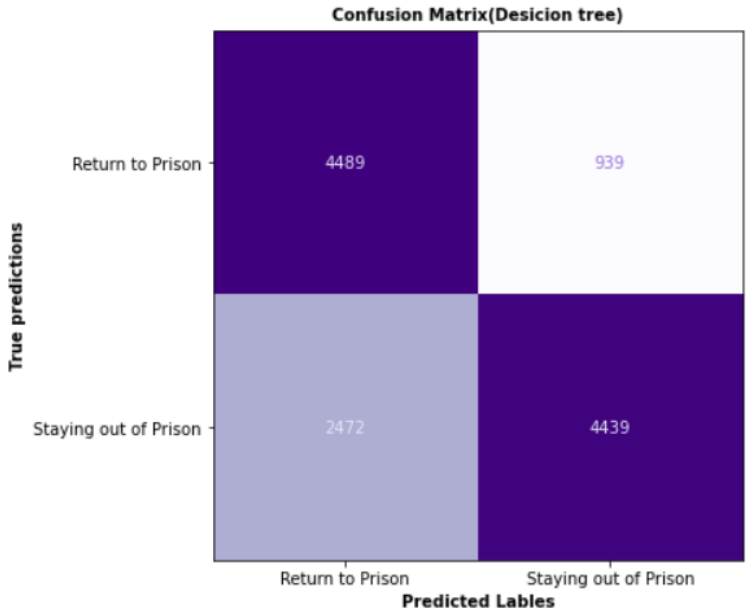
15424 rows × 11 columns

Figure 5: Dataset

Results:

```
Accuracy of Decision Tree is: % 72.36
```

Confusion Matrix:



<u>For running this part please have an accurate look on related directory and There you can easily find all of you need and also I would appreciate it if you would consider them</u> ☺

Here you can find all of my code implementation using Python if there is any issue with that please let me know to provide more detailed information. (be in touch **<u>here!</u>**)

Import libaraies

```
import pandas as pd
import random
import numpy as np
import itertools
import matplotlib.pyplot as plt
import matplotlib
import pandas as pd
import numpy as np
import math
import random
import matplotlib.pyplot as plt
import os
from numpy import linalg

from sklearn import preprocessing
from sklearn import tree
from sklearn.metrics import plot_confusion_matrix as plotConfustionMatrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
```

Required Functions

```
class Node:
    def __init__(self, child = np.array([]), data = None, label = None, attributes = None):
        self.child = child
        self.data = data
        self.label = label
        self.attributes = attributes
```

```
def grow_node(root, attribute, threshold):
  if(len(threshold[attribute]) == 2):
    root.child = np.append(root.child, [Node(data = np.array([data for data in root.data if data[attribute] ==  threshold[attribute][0]]))], axis = 0)
    root.child = np.append(root.child, [Node(data = np.array([data for data in root.data if data[attribute] == threshold[attribute][1]]))], axis = 0)

  else:
    root.child = np.append(root.child, [Node(data = np.array([data for data in root.data if data[attribute] ==  threshold[attribute][0]]))], axis = 0)
    root.child = np.append(root.child, [Node(data = np.array([data for data in root.data if data[attribute] ==  threshold[attribute][1]]))], axis = 0)
    root.child = np.append(root.child, [Node(data = np.array([data for data in root.data if data[attribute] ==  threshold[attribute][2]]))], axis = 0)
```

```
def encode_data(data, flag = 'library'):
  encoded_data = []
  for i in range(FEATURE):
    if flag == 'library':
      label_encoder = preprocessing.LabelEncoder()
      label_encoder.fit(train_data.T[i])
      encoded_data.append(label_encoder.transform(train_data.T[i]))
    else:
      encoded_data.append([dict([(y, x) for x, y in enumerate(sorted(set(train_data.T[i])))])[x] for x in train_data.T[i]])

  return encoded_data
```

```
def split_data(data,size_percent,label_column):
  train_data_size = int(np.floor(data.shape[0]*size_percent))
  indexes = random.sample(range(0, data.shape[0]), train_data_size)
  indexes.sort()

  train = data[indexes]
  train_data, train_label = train[:, :], train[:,label_column]
  listed_data = data.tolist()
  for i in range(train_data_size): listed_data.pop(indexes[i] - i)
  test = np.array(listed_data)
  test_data, test_label = test[:, :], test[:,label_column]
  return train_data, test_data, train_label, test_label
```

```
def entropy(target_node):
    elements, counts = np.unique(target_node, return_counts = True)
    entropy = np.sum([(-counts[i]/np.sum(counts)) * np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])
    return entropy
```

```
def information_gain(root):
  total_entropy = entropy(root.data.T[-1])
  reduction = np.sum([(child.data.shape[0] / root.data.shape[0]) * entropy(child.data.T[-1]) for child in root.child if not child.data.shape[0] == 0], axis = 0)
  gain = total_entropy - reduction
  return gain
```

```
def find_best_attribute(root, threshold):
  information = np.array([])
  for attr in range(len(root.data[0]) - 1):
      temp_node = Node(data = root.data, label = root.label, attributes = root.attributes)
      grow_node(temp_node, attr, threshold)
      information = np.append(information, [information_gain(temp_node)], axis=0)
      best_attibute = np.argmax(information)
```

```
def move_along_tree(test_data, tree, threshold):
  if tree.label != None : return tree.label
  else:
    if len(threshold[tree.attributes]) == 2 :
      return move_along_tree(test_data, tree.child[0], threshold) if test_data[tree.attributes] == threshold[tree.attributes][0] else move_along_tree(test_data, tree.child[1], threshold)
    else:
        if test_data[tree.attributes] == threshold[tree.attributes][0]: return move_along_tree(test_data, tree.child[0], threshold)
        elif test_data[tree.attributes] == threshold[tree.attributes][1]: return move_along_tree(test_data, tree.child[1], threshold)
        else: return move_along_tree(test_data, tree.child[2], threshold)
```

```
def test_trained_tree(train_data, test_data, final_tree):
  return [move_along_tree(test_data[i], final_tree, threshold) for i in range(len(test_data))]
```

```
def getAccuracy(predicts_label,test_data):
    return (predicts_label == test_data[:, -1].astype(int)).mean()*100
```

Reading Data & Shuffling

```
data = pd.read_csv('prison_dataset.csv')
data = data.to_numpy()
np.random.shuffle(data)
train_data, test_data, train_label, test_label = split_data(data,0.8,10)
encoded_train_data = np.array(encode_data(train_data)).T
encoded_test_data = np.array(encode_data(test_data)).T
FEATURE = 11
K = 3
SIZE = 11/K
```

```
threshold = calculate_threshold(encoded_train_data)
max_depth = 3
final_tree = ID3(encoded_train_data, max_depth, threshold)
```

```
predicted_labels = test_trained_tree(encoded_train_data, encoded_test_data, final_tree)

print('Accuracy of Decision Tree is: %% %1.2f' %(getAccuracy(predicted_labels, encoded_test_data)))
```

```
Accuracy of Decision Tree is: % 72.36
```

```
  return best_attibute
```

```
def calculate_threshold(train_data):
  threshold = []
  for i in range(train_data.shape[1]):
      unique_attribute = np.unique(train_data[:, i])
      threshold.append(unique_attribute)
  return threshold
```

```
def confusion_matrix(actual, predicted):
    classes = np.unique(actual)
    confmat = np.zeros((len(classes), len(classes)))
    for i in range(len(classes)):
        for j in range(len(classes)):
            confmat[i, j] = np.sum((actual == classes[i]) & (predicted == classes[j]))
    return confmat
```

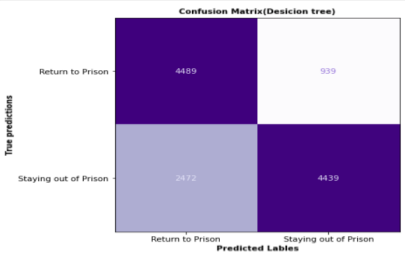
Implementing ID3 algorithm

```
def ID3(train_data, max_depth, threshold):
        root = Node(data = train_data)
        if (root.data.T[-1].all() == True):
            root.label = 1
            return root
        if (root.data.T[-1].any() == False):
            root.label = 0
            return root
        if(max_depth == 0):
            root.label = 1*((root.data.T[-1] == 1).sum() > (root.data.T[-1] == 0).sum())
            return root
        attribute = find_best_attribute(root, threshold)
        root.attributes = attribute
        if(len(threshold[attribute]) == 2):
            root.child = np.append(root.child, [ID3(np.array([data for data in train_data if data[attribute] == threshold[attribute][0]]), max_depth-1, threshold)], axis = 0)
            root.child = np.append(root.child, [ID3(np.array([data for data in train_data if data[attribute] == threshold[attribute][1]]), max_depth-1, threshold)], axis = 0)
        else:
            root.child = np.append(root.child, [ID3(np.array([data for data in train_data if data[attribute] == threshold[attribute][0]]), max_depth-1, threshold)], axis = 0)
            root.child = np.append(root.child, [ID3(np.array([data for data in train_data if data[attribute] == threshold[attribute][1]]), max_depth-1, threshold)], axis = 0)
            root.child = np.append(root.child, [ID3(np.array([data for data in train_data if data[attribute] == threshold[attribute][2]]), max_depth-1, threshold)], axis = 0)
        return root
```

```
def make_confusion_matrix(test_labels, predicted_labels):
  class_labels = np.unique(test_labels)
  return np.array([[np.sum((test_labels == class_labels[i]) & (predicted_labels == class_labels[j])) for j in range(len(class_labels))] for i in  range(len(class_labels))])
```

```
def plot_confusion_matrix(matrix, classes, normalize=False, title='Confusion matrix', cmap = plt.cm.Purples):
    plt.figure(figsize = (6,6))
    plt.imshow(matrix, interpolation = 'nearest', cmap = cmap)
    plt.title(title, fontsize = 10, fontweight = 'bold')
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)
    thresh = matrix.max() / 2.
    for i, j in itertools.product(range(matrix.shape[0]), range(matrix.shape[1])):
        plt.text(j, i, matrix[i, j], ha='center', va='center', color='lavender' if matrix[i,j]>thresh else 'mediumpurple')
    plt.ylabel('True predictions', fontsize = 10, fontweight = 'bold')
    plt.xlabel('Predicted Lables', fontsize = 10, fontweight = 'bold')
    plt.show()
```

```
confusion_matrix = make_confusion_matrix(encoded_test_data[:,-1], predicted_labels)
plot_confusion_matrix(confusion_matrix, ['Return to Prison', 'Staying out of Prison'], title = 'Confusion Matrix(Desicion tree)')
```



3.2 USING RANDOM FOREST (NO PYTHON LIBRARIES)

In this part we intend to simulate decision tree based on Random forest algorithm with result class of "Return to Prison numeric" parameter.

We consider max depth=3 and also use the 80% data in train cases and rest of that to use in test purposes.

Description:

As the question has pointed to a dynamic max-depth parameter in further sections, so we choose max-depth as one of our input argument.

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

Below are some points that explain why we should use the Random Forest algorithm:

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

Algorithm:

The Working process can be explained in the below steps:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

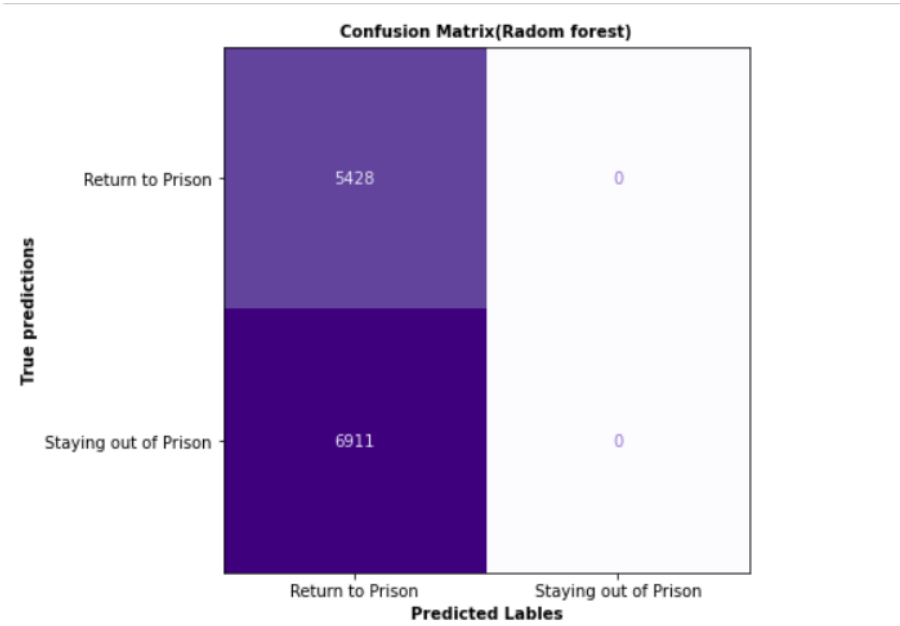
Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Results:

```
Accuracy of Random Forest with k =  3 is: % 43.99
```

Confusion Matrix:



We expected that the accuracy would be better but as you can see the accuracy has been decreased and it may refer to indexes which passed to input as of our random vectors and also it may refer to some key factors which doesn't concerned in my implementation!

However, I think it provides reliable and reasonable result in our approach.

Here you can find all of my code implementation using Python if there is any issue with that please let me know to provide more detailed information. (be in touch **here!**)

Part 2: Random Forest without libararies

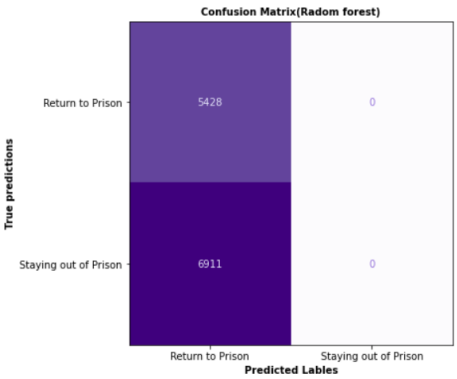
```
def ID3_random_forest(train_data, max_depth, threshold, k):

    root = Node(data = train_data)
    if not len(np.array(root.data)) == 0:
      if (root.data.T[-1].all() == True):
        root.label = 1
        return root
      if (root.data[-1].any() == False):
        root.label = 0
        return root
      if(max_depth == 0):
        root.label = 1*((root.data[-1] == 1).sum() > (root.data.T[-1] == 0).sum())
        return root
    else : return root
    attribute = find_best_attribute(root, threshold)
    new_attribute = attribute + int(k*SIZE)
    root.attributes = new_attribute
    if(len(threshold[new_attribute]) == 2):
      root.child = np.append(root.child, [ID3_random_forest(np.array([data for data in train_data if data[attribute] == threshold[new_attribute][0]]), max_depth-1, threshold, k)], axis = 0)
      root.child = np.append(root.child, [ID3_random_forest(np.array([data for data in train_data if data[attribute] == threshold[new_attribute][1]]), max_depth-1, threshold, k)], axis = 0)
    else:
      root.child = np.append(root.child, [ID3_random_forest(np.array([data for data in train_data if data[attribute] == threshold[new_attribute][0]]), max_depth-1, threshold, k)], axis = 0)
      root.child = np.append(root.child, [ID3_random_forest(np.array([data for data in train_data if data[attribute] == threshold[new_attribute][1]]), max_depth-1, threshold, k)], axis = 0)
      root.child = np.append(root.child, [ID3_random_forest(np.array([data for data in train_data if data[attribute] == threshold[new_attribute][2]]), max_depth-1, threshold, k)], axis = 0)
    return root
```

```
K = 3
SIZE = 11/K
final_tree = [ID3_random_forest(np.column_stack((encoded_train_data[:, int(K*SIZE):int((K+1)*SIZE)], encoded_train_data[:,-1])), 3, threshold, k) for k in range(K)]
final_predictions = []
for en in range(len(encoded_test_data)):
  predictions = [(move_along_tree(encoded_train_data[en], final_tree[k], threshold)) for k in range(K)]
  final_predictions.append(1*((np.array(predictions) == 1).sum() > (np.array(predictions) == 0).sum()))
```

```
print('Accuracy of Random Forest with k = ', K , 'is: %% %1.2f' %(getAccuracy(final_predictions, encoded_test_data)))
```

```
Accuracy of Random Forest with k =  3 is: % 43.99
```

```
confusion_matrix = make_confusion_matrix(encoded_test_data[:,-1], final_predictions)
plot_confusion_matrix(confusion_matrix, ['Return to Prison', 'Staying out of Prison'], title = 'Confusion Matrix(Radom forest)')
```



Part 3: Random Forest with libararies

```
clf = tree.DecisionTreeClassifier(criterion = "entropy", random_state = 0, max_depth = 3)
clf = clf.fit(encoded_train_data[:,:-1], encoded_test_data[:,-1])
tree.plot_tree(clf);
```

For running this part please have an accurate look on related directory and There you can easily find all of you need and also I would appreciate it if you would consider them ☺

3.3 USING RANDOM FOREST (WITH PYTHON LIBRARIES)

Implementation in Python:

In this section we intend to implement random forest using python libraries.

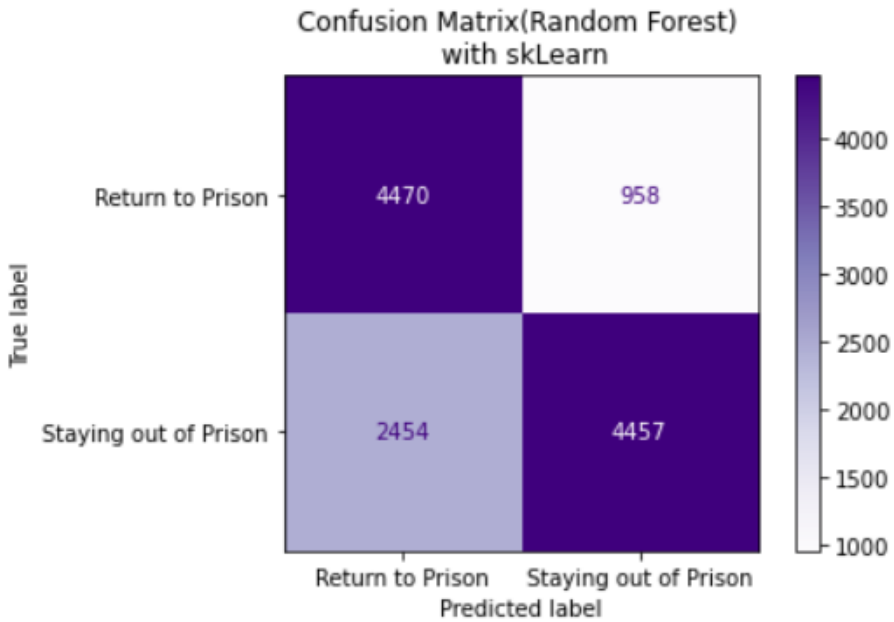
results have been provided below:

As you can see the results match as well as we expected with our previous analyses.

Results:

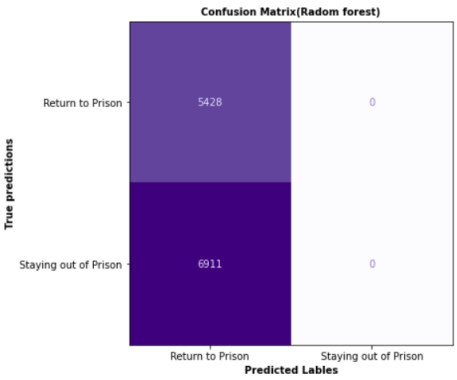
```
Accuracy of Decision Tree using sklearn is: % 72.34
```

Confusion Matrix:



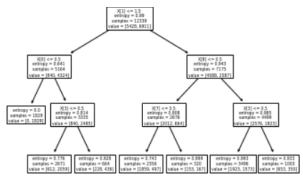
Here you can find all of my code implementation using Python if there is any issue with that please let me know to provide more detailed information. (be in touch **here!**)

```
confusion_matrix = make_confusion_matrix(encoded_test_data[:,-1], final_predictions)
plot_confusion_matrix(confusion_matrix, ['Return to Prison', 'Staying out of Prison'], title = 'Confusion Matrix(Radom forest)')
```



Part 3: Random Forest with libaraies

```
clf = tree.DecisionTreeClassifier(criterion = "entropy", random_state = 0, max_depth = 3)
clf = clf.fit(encoded_train_data[:,:-1], encoded_test_data[:,-1])
tree.plot_tree(clf);
```

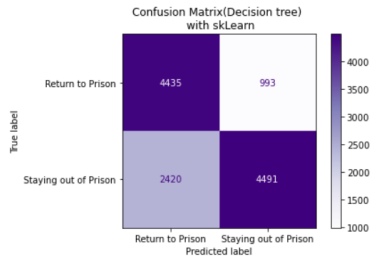


```
y_pred = clf.predict(encoded_test_data[:,:-1])
accuracy = (y_pred == encoded_test_data[:, -1].astype(int)).mean()
print('Accuracy of Decision Tree using sklearn is: %% %1.2f' %(accuracy*100))
```

```
Accuracy of Decision Tree using sklearn is: % 72.34
```

```
def plot_confusion_matrix_sk(clf, test_data, test_label, matrix_labels, matrix_title):
  disp = plotConfustionMatrix(clf, test_data, test_label, display_labels = matrix_labels, cmap=plt.cm.Purples)
  disp.ax_.set_title(matrix_title)
  plt.show()
```

```
plot_confusion_matrix_sk(clf, encoded_train_data[:,:-1], encoded_test_data[:,-1], ['Return to Prison', 'Staying out of Prison'],  'Confusion Matrix(Decision tree) \n with skLearn')
```



4 QUESTION #3

4.1 IMPLEMENTING KNN

In this part we tend to implement K-nearest neighbours classifier without using any libaries in python.

Note that in this stage we have used 80% of data for training purpose and rest of that has been used for testing the performance of our model.

After implementing our needed function, we obtain confusion matrix and also accuracy of model to found that the effects of increasing k-value in model performance.

Please also Note that cause of project description we implement KNN with an extra input argument has been setted to repeat results more than once and with different K parameter.

The best choice of k depends upon the data.

Generally, a small value of k means that noise will have a higher influence on the result; although larger values of k reduces effect of the noise on the classification, but also make it computationally expensive and make boundaries between classes less distinct.

In addition, a small value of k could lead to overfitting as well as a big value of k can lead to underfitting. Overfitting imply that the model is well on the training data but has poor performance when new data is coming. Underfitting refers to a model that is not good on the training data and also cannot be generalized to predict new data.

further you can find all of my code refer to part 4.1(implementing without libraries).

Import libaraies

```
import pandas as pd
import numpy as np
import math
import random
import os
from numpy import linalg
from metric_learn import LMNN
from metric_learn import NCA
```

Import data

```
z=pd.read_csv('wine.csv').to_numpy()
```

Required Functions

```
def getAccuracy(predicts_label,train_label):
    correct = 0
    for x in range(len(predicts_label)):
        if predicts_label[x] == train_label[x]:
            correct += 1
    return (correct/float(len(predicts_label))) * 100.0
def calculate_accuracy(predict, test):
    return ((predict == test).mean())*100
```

```
def confusion_matrix(actual, predicted):

    # extract the different classes
    classes = np.unique(actual)

    # initialize the confusion matrix
    confmat = np.zeros((len(classes), len(classes)))

    # loop across the different combinations of actual / predicted classes
    for i in range(len(classes)):
        for j in range(len(classes)):

            # count the number of instances in each combination of actual / predicted classes
            confmat[i, j] = np.sum((actual == classes[i]) & (predicted == classes[j]))

    return confmat
```

```
def Euclidian_Distance(instance1, instance2):
    distance = 0
    distance += pow((instance1 - instance2), 2)
    return math.sqrt(distance)
```

```
def KNN(train_data, train_label, test_data, k):

    train_data, test_data = train_data.astype(float), test_data.astype(float)

    predicts = []
    length = len(test_data)-1
    for i in range(len(test_data)):
        distances = np.linalg.norm(test_data[i] - train_data, axis = 1)
        neighbors_index = np.argsort(distances)[0:k]
        predicts.append(np.bincount(train_label[neighbors_index].flatten().astype(int)).argmax())
    return predicts
```

Reading Data & Shuffle

```
random.shuffle(z)
training_data=z[0:142,1:]       ##142
training_Label=z[0:142,0]       #142
testing_data=z[142:178,1:]      ##36
testing_Label=z[142:178,0]      ##36
k = [3, 5,9,11,13,15]
```

Testing Performance K=3

```
#k = 3
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[0])

Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_KNN=confusion_matrix(predictions_KNN, testing_Label)


print(Accuracy_KNN)
print(confusion_matrix_KNN)
```

```
80.55555555555556
[[13.  2.]
 [ 1. 16.]]
```

Testing Performance K=5

```
#k = 5
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[1])


Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_KNN=confusion_matrix(predictions_KNN, testing_Label)


print(Accuracy_KNN)
print(confusion_matrix_KNN)
```

```
77.77777777777779
[[12.  2.]
 [ 2. 16.]]
```

Testing Performance K=9

```
#k = 9
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[2])

Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_KNN=confusion_matrix(predictions_KNN, testing_Label)


print(Accuracy_KNN)
print(confusion_matrix_KNN)
```

```
77.77777777777779
[[14.  4.]
 [ 0. 14.]]
```

Testing Performance K=11

```
#k = 11
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[3])


Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_KNN=confusion_matrix(predictions_KNN, testing_Label)


print(Accuracy_KNN)
print(confusion_matrix_KNN)
```

```
77.77777777777779
[[14.  4.]
 [ 0. 14.]]
```

Testing Performance K=13

```
#k = 13
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[4])


Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_KNN=confusion_matrix(predictions_KNN, testing_Label)


print(Accuracy_KNN)
print(confusion_matrix_KNN)
```

```
75.0
[[14.  5.]
 [ 0. 13.]]
```

Testing Performance K=15

```
#k = 15
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[5])

Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_KNN=confusion_matrix(predictions_KNN, testing_Label)


print(Accuracy_KNN)
print(confusion_matrix_KNN)
```

```
75.0
[[14.  5.]
 [ 0. 13.]]
```

K=5

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} \mathbf{14} & \mathbf{5} \\ \mathbf{0} & \mathbf{13} \end{bmatrix}$$

Accuracy of KNN for K=5 Is: % 72.22

As you can see in results I have repeated experiments for different values of k and as you can see it hasn't any regular trend and as we mentioned above we can say that:

Generally, a small value of k means that noise will have a higher influence on the result; although larger values of k reduces effect of the noise on the classification, but also make it computationally expensive and make boundaries between classes less distinct.

In addition, a small value of k could lead to overfitting as well as a big value of k can lead to underfitting. Overfitting imply that the model is well on the training data but has poor performance when new data is coming. Underfitting refers to a model that is not good on the training data and also cannot be generalized to predict new data.

4.2 IMPLEMENTING METRIC LEARNING (LMNN & NCA)

In this part we implement LMNN & NCA algorithm for improving KNN performance as per usual we have used Sklearn python libarary to implement LMNN & NCA

Implementing has been done in python and results have provided below:

K=3

LMNN:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} 9 & 0 & 0 \\ 0 & 13 & 0 \\ 0 & 1 & 13 \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of LMNN for K=3 Is: % 97.22

NCA:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} 7 & 0 & 1 \\ 0 & 11 & 4 \\ 2 & 3 & 8 \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of NCA for K=3 Is: % 72.22

K=5

LMNN:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} \mathbf{9} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{13} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{13} \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of LMNN for K=5 Is: % 97.22

NCA:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} \mathbf{7} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{11} & \mathbf{2} \\ \mathbf{2} & \mathbf{3} & \mathbf{11} \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of NCA for K=5 Is: % 80.56

K=9

LMNN:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} \mathbf{8} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{13} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{13} \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of LMNN for K=9 Is: % 94.44

NCA:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} \mathbf{7} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{11} & \mathbf{4} \\ \mathbf{2} & \mathbf{2} & \mathbf{8} \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of NCA for K=9 Is: % 72.22

K=11

LMNN:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} 8 & 0 & 0 \\ 1 & 13 & 0 \\ 0 & 1 & 13 \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of LMNN for K=11 Is: % 94.44

NCA:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} 6 & 0 & 1 \\ 0 & 10 & 5 \\ 3 & 4 & 7 \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of NCA for K=11 Is: % 63.89

K=13

LMNN:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} 8 & 0 & 0 \\ 1 & 13 & 0 \\ 0 & 1 & 13 \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of LMNN for K=13 Is: % 94.44

NCA:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} 7 & 0 & 0 \\ 0 & 10 & 6 \\ 2 & 4 & 7 \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of NCA for K=13 Is: % 66.67

K=15

LMNN:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} 8 & 0 & 0 \\ 1 & 13 & 1 \\ 0 & 1 & 12 \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of LMNN for K=15 Is: % 91.67

NCA:

Confusion Matrix: (Obtained by Python)

$$\begin{bmatrix} 6 & 0 & 0 \\ 0 & 11 & 3 \\ 3 & 3 & 10 \end{bmatrix}$$

Accuracy: (Obtained by Python)

Accuracy of NCA for K=15 Is: % 75

As you can see the accuracy has been increased when we used LMNN & NCA algorithm in touch with normal KNN and we have provided different results with different K parameter as you can see in my codes and in all of that we observed that accuracy increased when we used a LMNN or NCA algorithm.

ALGORITHMS:

1) Large Margin Nearest Neighbor Metric Learning:

LMNN learns a Mahalanobis distance metric in the kNN classification setting. The learned metric attempts to keep close k-nearest neighbors from the same class, while keeping examples from different classes separated by a large margin. This algorithm makes no assumptions about the distribution of the data.

The distance is learned by solving the following optimization problem:

$$\min_{\mathbf{L}} \sum_{i,j} \eta_{ij} ||\mathbf{L}(\mathbf{x_i} - \mathbf{x_j})||^2 + c \sum_{i,j,l} \eta_{ij}(1 - y_{ij})[1 + ||\mathbf{L}(\mathbf{x_i} - \mathbf{x_j})||^2 - ||\mathbf{L}(\mathbf{x_i} - \mathbf{x_l})||^2]_+)$$

where x_i is a data point, x_j is one of its k-nearest neighbors sharing the same label, and x_l are all the other instances within that region with different labels, $\eta_{ij}, y_{ij} \in \{0, 1\}$ are both the indicators, η_{ij} represents x_{ij} is the k-nearest neighbors (with same labels) of x_i, y_{ij}=0 indicates x_i, x_j belong to different classes, $[\cdot]_+$=max(0,·) is the Hinge loss.

2) Neighborhood Components Analysis:

NCA is a distance metric learning algorithm which aims to improve the accuracy of nearest neighbors classification compared to the standard Euclidean distance. The algorithm directly maximizes a stochastic variant of the leave-one-out k-nearest neighbors (KNN) score on the training set. It can also learn a low-dimensional linear transformation of data that can be used for data visualization and fast classification.

They use the decomposition $M = L^T L$ and define the probability p_{ij} that x_i is the neighbor of x_j by calculating the softmax likelihood of the Mahalanobis distance:

$$p_{ij} = \frac{\exp(-||\mathbf{L}x_i - \mathbf{L}x_j||_2^2)}{\sum_{l \neq i} \exp(-||\mathbf{L}x_i - \mathbf{L}x_l||_2^2)}, \qquad p_{ii} = 0$$

Then the probability that x_i will be correctly classified by the stochastic nearest neighbors rule is:

$$p_i = \sum_{j: j \neq i, y_j = y_i} p_{ij}$$

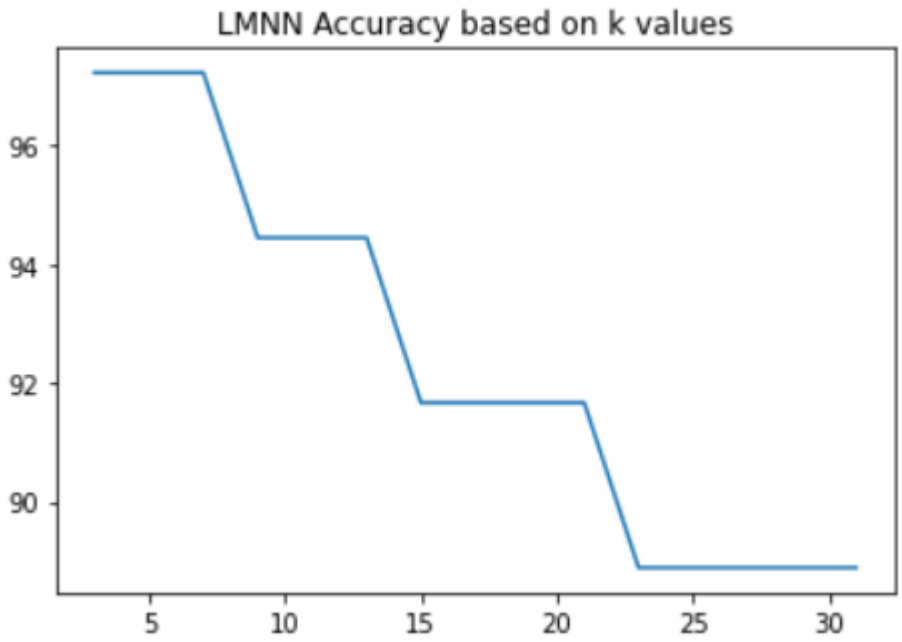
The optimization problem is to find matrix $\mathbf{L}$ that maximizes the sum of probability of being correctly classified:

$$\mathbf{L} = \text{argmax} \sum_i p_i$$

Finding the best K value using graphing the LMNN & NCA accuracy:

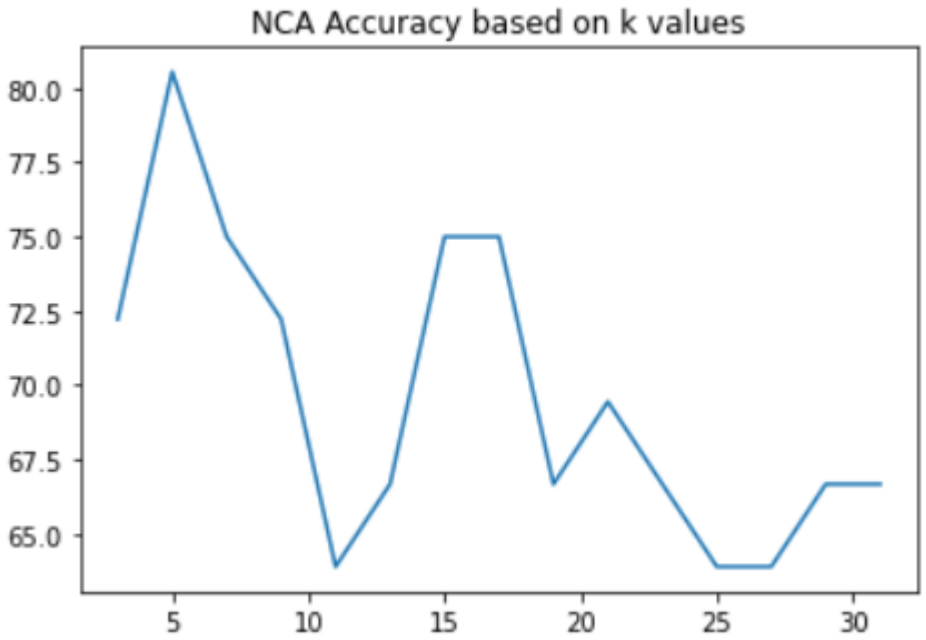
In this part we used a k vector and using 'for' command in python obtained all of accuracy values based on our Metric-learn function for both LMNN & NCA algorithm and as you can see in below figure for LMNN algorithm the optimal K values could be **k=5** as of our analyses.

k-values = [3,5,7,9,11,13,15,17,19,21,23,25,27,29,31]



And further we have used a k vector and using 'for' command in python obtained all of accuracy values based on our Metric-learn function for both LMNN & NCA algorithm and as you can see in below figure for LMNN algorithm the optimal K values could be **k=5** as of our analyses.

k-values = [3,5,7,9,11,13,15,17,19,21,23,25,27,29,31]



Finding The best K value

```
k_values = [3,5,7,9,11,13,15,17,19,21,23,25,27,29,31]
k_dimansion=np.size(k_values)
```

```
vector_accuracy_LMNN=np.zeros(k_dimansion)
vector_accuracy_NCA =np.zeros(k_dimansion)
```

```
for i in range(k_dimansion):
    predictions_LMNN=  Metric_learn(training_data,training_Label,testing_data,k_values[i],'LMNN')
    predictions_NCA =  Metric_learn(training_data,training_Label,testing_data,k_values[i],'NCA')
    vector_accuracy_LMNN[i]=getAccuracy(predictions_LMNN, testing_Label)
    vector_accuracy_NCA[i] =getAccuracy(predictions_NCA, testing_Label)
```

```
plt.plot(k_values,vector_accuracy_LMNN)
plt.title("LMNN Accuracy based on k values")
```

```
plt.plot(k_values,vector_accuracy_NCA)
plt.title("NCA Accuracy based on k values")
```

Here you can find all of my code implementation using Python if there is any issue with that please let me know to provide more detailed information. (be in touch **here!**)

Here you can find all of my code refer to part 4.2(implementing Metric Learning).

Import libaraies

```
import pandas as pd
import numpy as np
import math
import random
import os
from numpy import linalg
from metric_learn import LMNN
from metric_learn import NCA
```

Import data

```
z=pd.read_csv('wine.csv').to_numpy()
```

Required Functions

```
def getAccuracy(predicts_label,train_label):
    correct = 0
    for x in range(len(predicts_label)):
        if predicts_label[x] == train_label[x]:
            correct += 1
    return (correct/float(len(predicts_label))) * 100.0
def calculate_accuracy(predict, test):
    return ((predict == test).mean())*100
```

```
def confusion_matrix(actual, predicted):

    # extract the different classes
    classes = np.unique(actual)

    # initialize the confusion matrix
    confmat = np.zeros((len(classes), len(classes)))

    # loop across the different combinations of actual / predicted classes
    for i in range(len(classes)):
        for j in range(len(classes)):

           # count the number of instances in each combination of actual / predicted classes
           confmat[i, j] = np.sum((actual == classes[i]) & (predicted == classes[j]))

    return confmat
```

```
def Euclidian_Distance(instance1, instance2):
    distance = 0
    distance += pow((instance1 - instance2), 2)
    return math.sqrt(distance)
```

```
def split_data(data):
  train_data_size = int(np.floor(data.shape[0]*0.8))
  indexes = random.sample(range(0, data.shape[0]), train_data_size)
  indexes.sort()

  train = data[indexes]
  train_data, train_label = train[:, :], train[:, 0]
  listed_data = data.tolist()
  for i in range(train_data_size): listed_data.pop(indexes[i] - i)
  test = np.array(listed_data)
  test_data, test_label = test[:, :], test[:, 0]
  return train_data, test_data, train_label, test_label
```

```
def KNN(train_data, train_label, test_data, k):

    train_data, test_data = train_data.astype(float), test_data.astype(float)

    predicts = []
    length = len(test_data)-1
    for i in range(len(test_data)):
        distances = np.linalg.norm(test_data[i] - train_data, axis = 1)
        neighbors_index = np.argsort(distances)[0:k]
        predicts.append(np.bincount(train_label[neighbors_index].flatten().astype(int)).argmax())
    return predicts
```

```
def Metric_learn(train_data,train_label,test_data,k0,algorithm_type):

    train_data, test_data = train_data.astype(float), test_data.astype(float)
    lmnn = LMNN(k=k0, learn_rate=1e-7)
    nca  = NCA(max_iter=1000)
    lmnn.fit(train_data,train_label)
    nca.fit(train_data,train_label)

    if algorithm_type == 'LMNN':
        train_data_new_space =lmnn.transform(train_data)
        test_data_new_space=lmnn.transform(test_data)
    elif algorithm_type == 'NCA':
        train_data_new_space =nca.transform(train_data)
        test_data_new_space=nca.transform(test_data)
```

```
    predicts = []
    length = len(test_data_new_space)-1
    for i in range(len(test_data_new_space)):
        distances = np.linalg.norm(test_data_new_space[i] - train_data_new_space, axis = 1)
      ##  distances = Euclidian_Distance(test_data_new_space[i],train_data_new_space)
        neighbors_index = np.argsort(distances)[0:k0]
        predicts.append(np.bincount(train_label[neighbors_index].flatten().astype(int)).argmax())
    return predicts
```

Reading Data & Shuffle

```
k = [3, 5,9,11,13,15]


training_data,testing_data,training_Label,testing_Label=split_data(z)
```

Testing Performance K=3

```
#k = 3
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[0])
predictions_LMNN=  Metric_learn(training_data,training_Label,testing_data,k[0],'LMNN')
predictions_NCA =  Metric_learn(training_data,training_Label,testing_data,k[0],'NCA')


Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_KNN, testing_Label)


Accuracy_LMNN = getAccuracy(predictions_LMNN,testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_LMNN, testing_Label)


Accuracy_NCA  = getAccuracy(predictions_NCA, testing_Label)
confusion_matrix_NCA=confusion_matrix(predictions_NCA, testing_Label)

print('Accuracy of KNN is: %% %1.2f' %(Accuracy_KNN))
print(confusion_matrix_KNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_LMNN))
print(confusion_matrix_LMNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_NCA))
print(confusion_matrix_NCA)
```

```
print('Accuracy of KNN is: %% %1.2f' %(Accuracy_NCA))
print(confusion_matrix_NCA)
```

```
Accuracy of KNN is: % 69.44
[[14.  5.]
 [ 0. 13.]]
Accuracy of KNN is: % 97.22
[[ 9.  0.  0.]
 [ 0. 13.  0.]
 [ 0.  1. 13.]]
Accuracy of KNN is: % 72.22
[[ 7.  0.  1.]
 [ 0. 11.  4.]
 [ 2.  3.  8.]]
```

Testing Performance K=5

```
#k = 5
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[1])
predictions_LMNN=  Metric_learn(training_data,training_Label,testing_data,k[1],'LMNN')
predictions_NCA =  Metric_learn(training_data,training_Label,testing_data,k[1],'NCA')


Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_KNN, testing_Label)


Accuracy_LMNN = getAccuracy(predictions_LMNN,testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_LMNN, testing_Label)


Accuracy_NCA  = getAccuracy(predictions_NCA, testing_Label)
confusion_matrix_NCA=confusion_matrix(predictions_NCA, testing_Label)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_KNN))
print(confusion_matrix_KNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_LMNN))
print(confusion_matrix_LMNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_NCA))
print(confusion_matrix_NCA)
```

```
Accuracy of KNN is: % 72.22
[[14.  5.]
 [ 0. 13.]]
Accuracy of KNN is: % 97.22
[[ 9.  0.  0.]
 [ 0. 13.  0.]
 [ 0.  1. 13.]]
Accuracy of KNN is: % 80.56
[[ 7.  0.  0.]
 [ 0. 11.  2.]
 [ 2.  3. 11.]]
```

Testing Performance K=9

```
#k = 9
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[2])
predictions_LMNN=  Metric_learn(training_data,training_Label,testing_data,k[2],'LMNN')
predictions_NCA =  Metric_learn(training_data,training_Label,testing_data,k[2],'NCA')


Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_KNN, testing_Label)


Accuracy_LMNN = getAccuracy(predictions_LMNN,testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_LMNN, testing_Label)


Accuracy_NCA  = getAccuracy(predictions_NCA, testing_Label)
confusion_matrix_NCA=confusion_matrix(predictions_NCA, testing_Label)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_KNN))
print(confusion_matrix_KNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_LMNN))
print(confusion_matrix_LMNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_NCA))
print(confusion_matrix_NCA)
```

```
Accuracy of KNN is: % 69.44
[[14.  5.]
 [ 0. 13.]]
Accuracy of KNN is: % 94.44
[[ 8.  0.  0.]
 [ 1. 13.  0.]
 [ 0.  1. 13.]]
Accuracy of KNN is: % 72.22
[[ 7.  1.  1.]
 [ 0. 11.  4.]
 [ 2.  2.  8.]]
```

Testing Performance K=11

```
#k = 11
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[3])
predictions_LMNN=  Metric_learn(training_data,training_Label,testing_data,k[3],'LMNN')
predictions_NCA =  Metric_learn(training_data,training_Label,testing_data,k[3],'NCA')


Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_KNN, testing_Label)


Accuracy_LMNN = getAccuracy(predictions_LMNN,testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_LMNN, testing_Label)


Accuracy_NCA  = getAccuracy(predictions_NCA, testing_Label)
confusion_matrix_NCA=confusion_matrix(predictions_NCA, testing_Label)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_KNN))
print(confusion_matrix_KNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_LMNN))
print(confusion_matrix_LMNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_NCA))
print(confusion_matrix_NCA)
```

```
Accuracy of KNN is: % 66.67
[[14.  5.]
 [ 0. 13.]]
Accuracy of KNN is: % 94.44
[[ 8.  0.  0.]
 [ 1. 13.  0.]
 [ 0.  1. 13.]]
Accuracy of KNN is: % 63.89
[[ 6.  0.  1.]
 [ 0. 10.  5.]
 [ 3.  4.  7.]]
```

Testing Performance K=13

```
#k = 13
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[4])
predictions_LMNN=  Metric_learn(training_data,training_Label,testing_data,k[4],'LMNN')
predictions_NCA =  Metric_learn(training_data,training_Label,testing_data,k[4],'NCA')


Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_KNN, testing_Label)


Accuracy_LMNN = getAccuracy(predictions_LMNN,testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_LMNN, testing_Label)
```

```
Accuracy_NCA  = getAccuracy(predictions_NCA, testing_Label)
confusion_matrix_NCA=confusion_matrix(predictions_NCA, testing_Label)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_KNN))
print(confusion_matrix_KNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_LMNN))
print(confusion_matrix_LMNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_NCA))
print(confusion_matrix_NCA)
```

```
Accuracy of KNN is: % 69.44
[[14.  5.]
 [ 0. 13.]]
Accuracy of KNN is: % 94.44
[[ 8.  0.  0.]
 [ 1. 13.  0.]
 [ 0.  1. 13.]]
Accuracy of KNN is: % 66.67
[[ 7.  0.  0.]
 [ 0. 10.  6.]
 [ 2.  4.  7.]]
```

Testing Performance K=15

```
#k = 15
predictions_KNN =  KNN(training_data,training_Label,testing_data,k[5])
predictions_LMNN=  Metric_learn(training_data,training_Label,testing_data,k[5],'LMNN')
predictions_NCA =  Metric_learn(training_data,training_Label,testing_data,k[5],'NCA')


Accuracy_KNN  = getAccuracy(predictions_KNN, testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_KNN, testing_Label)


Accuracy_LMNN = getAccuracy(predictions_LMNN,testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_LMNN, testing_Label)
```

```
Accuracy_LMNN = getAccuracy(predictions_LMNN,testing_Label)
confusion_matrix_LMNN=confusion_matrix(predictions_LMNN, testing_Label)


Accuracy_NCA  = getAccuracy(predictions_NCA, testing_Label)
confusion_matrix_NCA=confusion_matrix(predictions_NCA, testing_Label)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_KNN))
print(confusion_matrix_KNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_LMNN))
print(confusion_matrix_LMNN)


print('Accuracy of KNN is: %% %1.2f' %(Accuracy_NCA))
print(confusion_matrix_NCA)
```

```
Accuracy of KNN is: % 72.22
[[14.  5.]
 [ 0. 13.]]
Accuracy of KNN is: % 91.67
[[ 8.  0.  0.]
 [ 1. 13.  1.]
 [ 0.  1. 12.]]
Accuracy of KNN is: % 75.00
[[ 6.  0.  0.]
 [ 0. 11.  3.]
 [ 3.  3. 10.]]
```

For running this part please have an accurate look on related directory and There you can easily find all of you need and also I would appreciate it if you would consider them ☺

Conclusion:

We found that despite the second implementation without using libraries has lower accuracy but this implementation helped us to have a dipper investigation on discussed concepts and materials.

And as we expect the results of the sklearn implementation are better, but our result is still fine for a really simple model!

Final Note:

in some part of this project specially in part 1 I implement functions to process theoretical results in MATLAB which provided useful knowledge about numerical process which has been hidden behind the problems!

You can find all of my additional code in related directory, Thank you so much for your consideration ;)

5 ACKNOWLEDGEMENT

I am really grateful for Mr MohammadReza Tavakoli (810197477) because in some part of this project we had a nice and effective discussion which highly helped us to have a better analyse and also have more adequate results! (Note that we only talked about ideas behind different problems.)

Afterwards, I am thankful to all of course teaching assistants: Omid vaheb(ovaheb@gmail.com) and Ali saeizadeh (Alisaei90@gmail.com) and Reza talakoob (rezatalakoob@yahoo.com) who designed this project with high quality.

6 REFERENCES

[1] https://towardsdatascience.com/3-techniques-to-avoid-overfitting-of-decision-trees-1e7d3d985a09
[2] http://contrib.scikit-learn.org/metric-learn/supervised.html
[3] https://stackoverflow.com/questions/57876956/converting-pandas-string-data-to-numeric-for-decision-tree
[4] https://datascience.stackexchange.com/questions/5226/strings-as-features-in-decision-tree-random-forest
[5] https://pydml.readthedocs.io/en/latest/dml.lmnn.html
[6] https://en.wikipedia.org/wiki/Large_margin_nearest_neighbor
[7] https://datascience.stackexchange.com/questions/31729/random-forest-vs-rainforest
[8] https://link.springer.com/article/10.1023/A:1009839829793
[9] https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/
[10] https://machinelearningmastery.com/implement-random-forest-scratch-python/
[11] https://towardsdatascience.com/id3-decision-tree-classifier-from-scratch-in-python-b38ef145fd90
[12] https://guillermoarriadevoe.com/blog/building-a-id3-decision-tree-classifier-with-python/
[13] https://python-course.eu/Decision_Trees.php