Project Report

Title:

Tag-to-Tag Routing Algorithm Implementation using Python and Machine Learning

Prepared By:
Sourabh Tantuway
MSCS at The University of Texas at Dallas
Spring 2023 semester

Instructed By:

Dr. Zygmunt J. Haas
Professor at The University of Texas at Dallas

Overview:

This project is based on the research paper by Dr. Zygmunt J. Haas, which deals with the idea of passive tags communicating with each other in a large-scale network. While the research paper provides an extensive study about how T2T communication results in significantly larger capacity than centralized RFID reader-based system, this project is just an effort to practically implement the basic routing algorithm mentioned as "*Algorithm 1: Routing Protocol for Scenario A*" in the paper. Kindly go through the paper to better understand the project.

Research paper link: (https://ieeexplore.ieee.org/document/9146888)

Tools and Technologies Used:

In this project, I used *Python* language and some *Machine Learning libraries* for the implementation of the algorithm. Entire project can be run using the python notebook file (i.e. *Project.ipynb*). In my case I used *Jupyter Notebook* server locally and ran the *.ipynb* file on that.

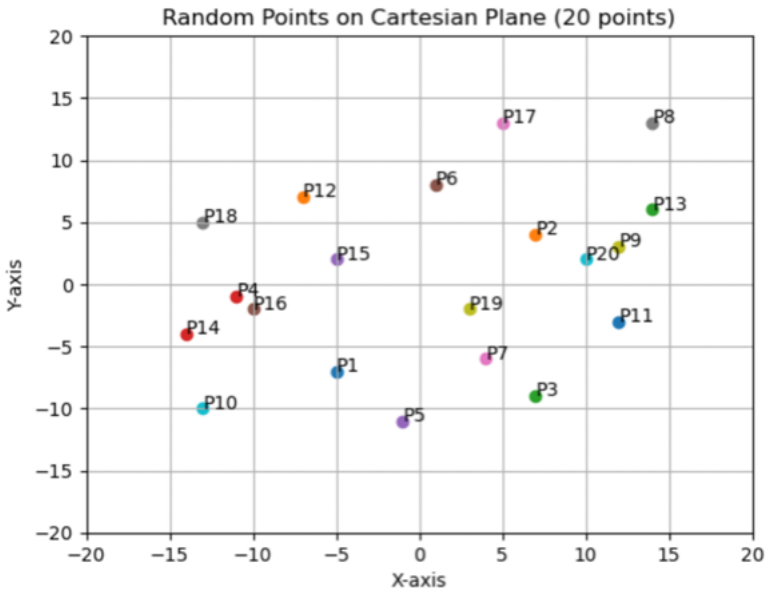
Assumptions:

- For making things simpler, in this project we assumed that each tag cannot have more than one message to transmit.
- Weights of all messages will be same.

Implementation:

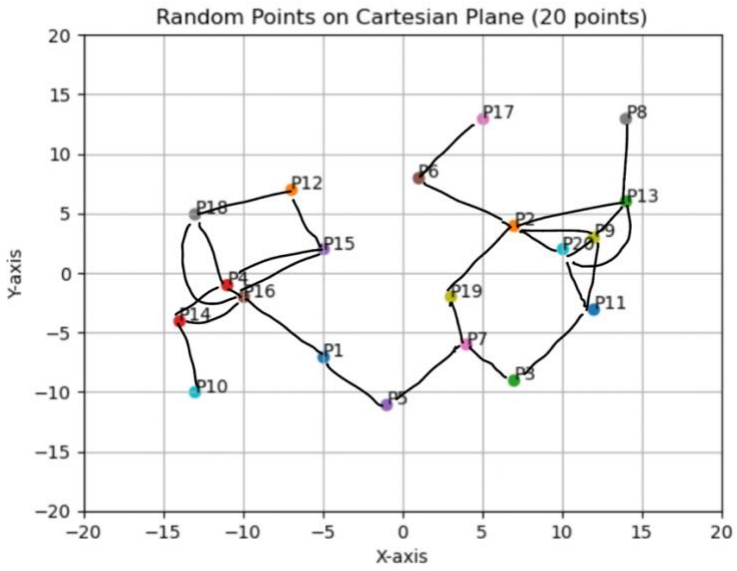
The project consists of *Project.ipynb* file, which has the main code and some additional *csv* files having some data.

Python code generates the initial arrangement of RF tags in a 2D plane. Specifically, **20** random points are selected inside a **15x15** grid. Number of points and grid dimensions can be set in the code. Arrangement used for this project is shown below.



After positions of tags are decided, distance between each pair of tags is calculated. These distances will be stored in a matrix. This *distance matrix* can be stored locally so that this arrangement of tags can be reused later to perform analysis. In our case, I stored one specific arrangement in a *csv* file (*dist_table.csv*) and entire project analysis is done over that specific arrangement only.

Now the *Radius* of transmission is selected, which is same for all tags. *Radius* selection is subject to *distance matrix*, such that each tag is having at least one other tag inside it's radius. More specifically, If the distance between a pair of tags is less than or equal to the *radius*, there is an edge between them. So, select a radius such that connected graph is formed where tags are the nodes. In my case, I selected the radius value *8.00* . Graph of tags is shown below.



Now the idea is that if some number of tags want to transmit their messages to their respective destinations. The *Algorithm1*, tells which tags will transmit finally, such that there will be minimum interference between tags.

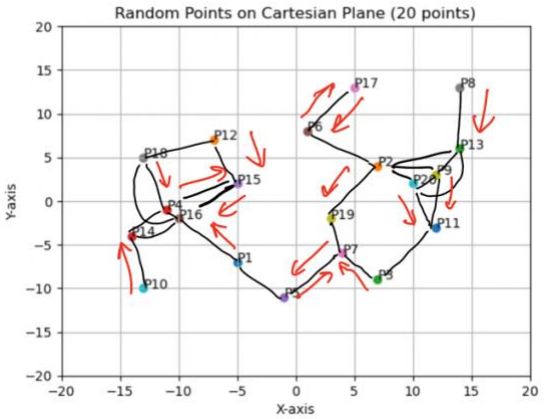
Algorithm1 uses constraints to decide which tags can't transmit while some other tag is transmitting. That is avoiding the interference between tags.

Python code generate an "*Instance*" of tags. By instance we mean number of candidate tags which want to transmit. And corresponding "*Solution*" of the instance, which is final tags selected for transmission. *Instance* and *Solution* are vectors of size equal to number of nodes, i.e. 20.

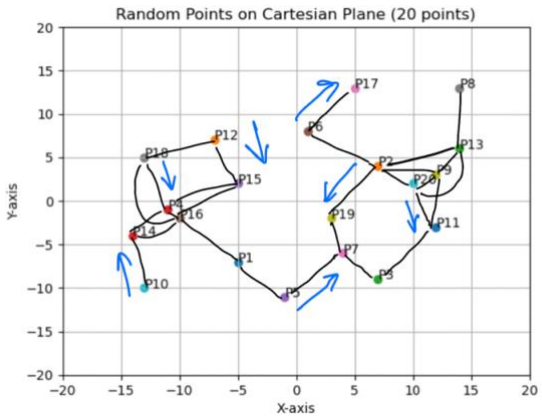
In *Instance* vector value *a* at index *i*, means *node i* wants to transmit to destination *node a*.

In *Solution* vector value *0/1* at index *i*, means node *i* is finally selected for transmission.

Examples of *Instance*(left) and *Solution*(right) shown below.



[16,19,7,15,7,17,5,13,11,14,0,15,0,0,16,0,6,4,0,11]



[0,1,0,0,1,1,0,0,0,1,0,1,0,0,0,0,0,1,0,1]

With python code we can generate as many rows as we want, of the *instances* and corresponding *solutions* given by the *Algorithm1*. We use *generate_data ()* function for generating the dataset. It is the main function which uses all other functions used in the *Algorithm1*. This function takes single parameter which is the total number of rows we want to generate.

In my specific arrangement of tags, I generated *90k* rows using this function and saved the dataset locally. And used 80% of this dataset to train a machine learning model.

This model can predict the solution for any given instance with a certain amount of accuracy. In this project I was able to achieve nearly *70%* accuracy for the ML model.

Steps to generate the dataset and train the model is given in 'Running Instruction' section.

Running Instruction:

Python notebook file should be executed in sequence, from starting to end.

Some dependencies might be required to run the code successfully. They can be installed as you run the code.

Before executing the python file make sure if the csv files are in place:

- *dist_table.csv* should be in the same folder as *Project.ipynb*
- folder named '*dataset'* should also be in the same folder as *Project.ipynb* file. This folder contains *dataset.csv* used for training and testing the ML model.

dist_table.csv contains specific arrangement of tags for which ML model will be trained.

dataset.csv contains the 90k rows of instances and solutions. These rows correspond to the tag arrangement stored in *dist_table.csv*.

Now change the path of *dataset.csv* in the code and execute all the cells. The model will be trained, and the accuracy of model will come nearly *70%*.

New arrangement of tags can be generated and stored in *dist_table.csv* and new dataset can be generated using *generate_data(num_of_rows)* function and saved in *dataset.csv*. If done so, ML model will be generated for this new data and accuracy will be according to the amount of data generated.

Challenges:

I faced some main challenges in preparing the data for the classifier. We are predicting a solution for an instance which is a vector representing the number of nodes. This type of problem comes under multilabel classification problem. So, preparation of data took some time, which involved making 20 separate features for our 20 nodes arrangement. Also, there are 20 labels corresponding to our 20 nodes. If *label_1* has value 1, it means node 1 is selected for transmission, otherwise value will be 0. After preparation of data applying the model was relatively easier.

Results:

I used *DecisionTreeClassifier* from *sklearn* library. *MultiOutputClassifier* class in *sklearn* helped dealing with multi label classification and supported vector type prediction in our case. I divided all *90k* data rows in *training* and *testing* set *80%* and *20%* respectively. Trained the classifier over *training* set and tested over *testing* set to achieve ***70%* accuracy**.

Future Work and Improvement:

There is a scope of improvement in the current project as mentioned below:

- Other ML classifiers can be explored which will support our type of dataset.
- Hyperparameter tuning can be done over current model to see if higher accuracy can be achieved.
- Optimization can be done in current code.
- Accommodating code to allow more than one message for a single tag.
- Allowing weights/priorities of messages.
- Implementing *Algorithm2* and *Algorithm3* mentioned in the paper.

Iteration 2:

Understanding the accuracy:

If any solution has more than one incorrectly selected node, the whole solution is discarded.
Only the ones with all nodes selected correctly are contributing to the accuracy.

Overfitting and hyperparameter tuning:

It turned out that our model was overfitting because training accuracy was 100%. So, we performed hyperparameter tuning over our model.

First we analyzed the decision tree being formed to check it's max-depth, min-samples-leaf, min-samples-split. These are the hyperparameters of the decision tree.

Now using sklearn's GridSearchCV, we figured out best hyperparameters as shown below.

```
]: best_params  #printing best hyperparameters

]: {'estimator__criterion': 'gini',
    'estimator__max_depth': 16,
    'estimator__min_samples_leaf': 4,
    'estimator__min_samples_split': 16}
```

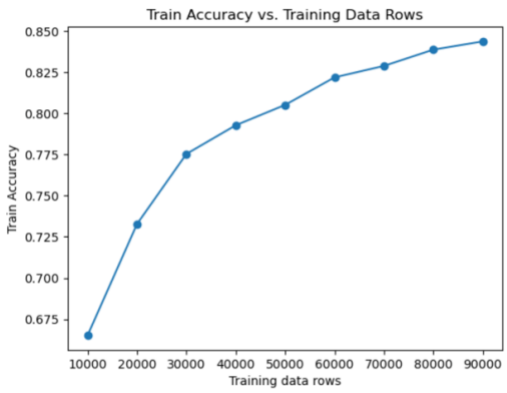
Using these parameters in our model on the same dataset, we got higher test accuracy of 75%. Training accuracy also got reduced to 83%. It means model is not overfitting.

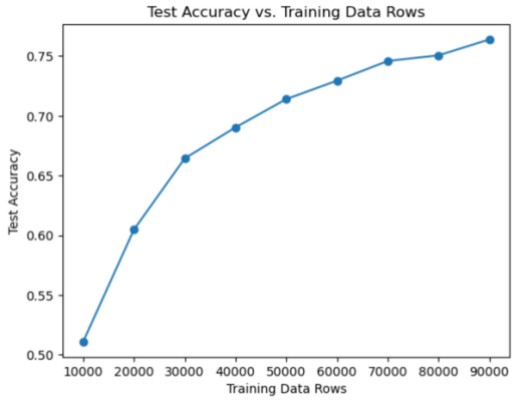
```
score = clf.score(X_train, y_train)
print('Accuracy:', score)
score = clf.score(X_test, y_test)
print('Accuracy:', score)
```

```
Accuracy: 0.8306994993911514
Accuracy: 0.7499447548694637
```

Relation of accuracy with amount of training data:

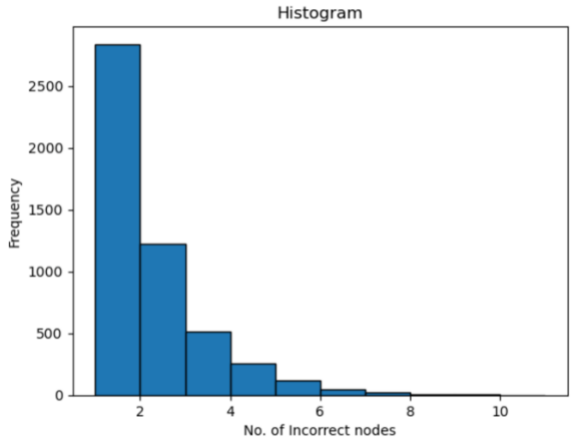
As figured, increasing the amount of training data also increases the accuracy. We calculated train and test accuracy of model by providing different amount of training data and plotted a graph as shown below:





Distribution of incorrect nodes in all incorrect predictions:

We want to see in all the wrong predictions made by model, how many incorrect nodes are there. Some predictions can have 1 incorrect nodes , 2 incorrect nodes and so on. This distribution can be represented by the graph shown below:



Test data rows : 21118
Incorrect predictions : 5032
Around 2800 predictions has only 1 incorrectly selected node

Optimization in generating Datasets:

Previously it was taking way too longer to generate the dataset for some specific arrangement of tags. For example, for generating 20k rows it was taking whole day. Now the code has been changed. There were some unnecessary conditions which are removed now. Also, loop was running unnecessarily which is corrected and break statements are added.

Following is the optimized code for solve_instance() function which mainly decided the running time:

In [21]:
```
import sys
import itertools

def solve_instance(d, c):          #updated the parameters

    indexes = [i for i, num in enumerate(d) if num > 0]
    combinations = []
    result = None
    unique_combinations = []


    for size in range(len(indexes), 0, -1):
        #genetating all combination such that any 1 node finally transmits , or 2 nodes and so on.
        #and these combinations will be checked if they satisfy the constraints
        combinations = itertools.combinations(indexes, size)


        for combination in combinations:
            valid_comb = True
            for elem in combination:          # EACH element of combination should satisfy the consrtaints
                intersection = set(c[elem]).intersection(set(combination))
                if(len(intersection) > 0):
                    valid_comb = False
                    break


            if(valid_comb):                              #for stopping the loop when valid combination found
                result = combination
                break


        if(result != None and len(result) > 0):    #for stopping the loop when valid combination found
            break


    return list(result)
```

Hoeffding Tree for Incremental Learning:

As we want to achieve dynamic incremental learning for our case, we are updating our classifier to incremental classifier which preserves old information as well as keeps on learning from new incoming data from stream or batch. One such classifier is HoeffdingAdaptiveDecisionTree from river library.

- classifier works well with dictionary datatypes.

Handling case when arrangement of tags changes:

If tags are moved. In this case new data generated will be different from the previous arrangement. So our classifier should detect the change in data and train itself according to the new data. So drift detection mechanism is to be used.

```
ht_model = multioutput.ClassifierChain(
    model=tree.HoeffdingAdaptiveTreeClassifier( grace_period=100,
                                                drift_detector=drift.binary.DDM(),
                                                #max_depth = 16,
                                                #split_criterion = 'gini',
                                                #nominal_attributes=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
                                                nominal_attributes=['Att1', 'Att2', 'Att3', 'Att4', 'Att5', 'A
))

n_samples = 0
correct_cnt = 0
```

Aug 21, 2023 & Aug 22, 2023:

Task it to check the effect on the accuracy of the model when it is trained by one type of data and later incrementally trained with slightly different type of data. In our case training the classifier for first arrangement of tags and later for second arrangement of tags and check the accuracies.

For this, I generated 5 *training datasets* for arrangement_1, each having 20k rows:

1. X_set1_1, y_set1_1
2. X_set2_1, y_set2_1
3. X_set3_1, y_set3_1
4. X_set4_1, y_set4_1
5. X_set5_1, y_set5_1

```
In [155]: X_set1_1, y_set1_1 = generate_data(20000)
          X_set2_1, y_set2_1 = generate_data(20000)
          X_set3_1, y_set3_1 = generate_data(20000)
          X_set4_1, y_set4_1 = generate_data(20000)
          X_set5_1, y_set5_1 = generate_data(20000)

          9293 19294 19295 19296 19297 19298 19299 19300 19301 19302 19303 19304 19305 19306 19307 19308 19309 19310 19
          311 19312 19313 19314 19315 19316 19317 19318 19319 19320 19321 19322 19323 19324 19325 19326 19327 19328 193
          29 19330 19331 19332 19333 19334 19335 19336 19337 19338 19339 19340 19341 19342 19343 19344 19345 19346 1934
          7 19348 19349 19350 19351 19352 19353 19354 19355 19356 19357 19358 19359 19360 19361 19362 19363 19364 19365
          19366 19367 19368 19369 19370 19371 19372 19373 19374 19375 19376 19377 19378 19379 19380 19381 19382 19383 1
```

And 5 datasets for arrangement_2 :

1. X_set1_2, y_set1_2
2. X_set2_2, y_set2_2
3. X_set3_2, y_set3_2
4. X_set4_2, y_set4_2
5. X_set5_2, y_set5_2

```
In [64]: X_set1_2, y_set1_2 = generate_data(20000)
         X_set2_2, y_set2_2 = generate_data(20000)
         X_set3_2, y_set3_2 = generate_data(20000)
         X_set4_2, y_set4_2 = generate_data(20000)
         X_set5_2, y_set5_2 = generate_data(20000)

         2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087
         2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109
         2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131
         2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153
```

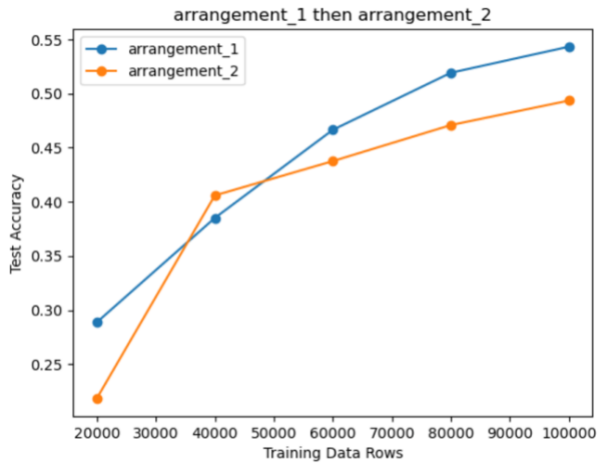
Along with above , 1 *test dataset* for each arrangement, having 50k rows:

```
In [29]: X_test_1, y_test_1 = generate_data(50000)
         X_test_2, y_test_2 = generate_data(50000)
```

Now trained the classifier with arrangement_1(incrementally with all 5 training sets) and then arrangement_2. And plotted the graph of accuracy:

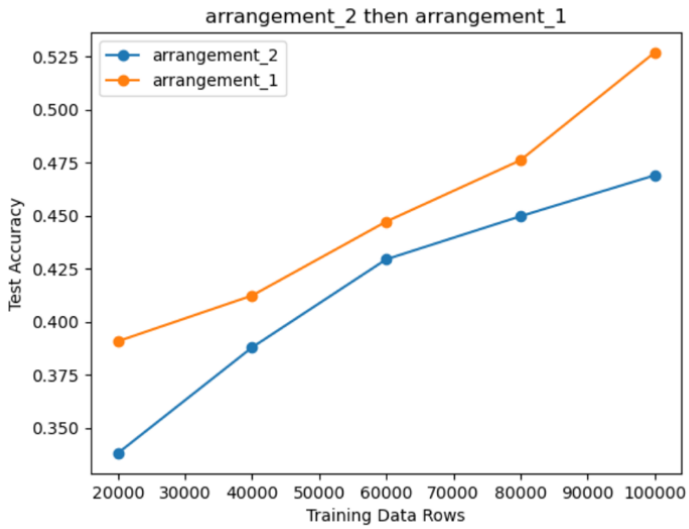
```
# accuracy_arr1 =  [0.2887, 0.38502, 0.46644, 0.51914, 0.54324]
# accuracy_arr2 =  [0.21838, 0.4058, 0.43754, 0.47082, 0.49346]


# accuracy_arr2 =  [0.33812, 0.38794, 0.42944, 0.44972, 0.469]
# accuracy_arr1 = [0.39084, 0.41236, 0.44722, 0.4761, 0.52682]


# accuracy_arr1 =  [0.32092, 0.42144, 0.51282, 0.51024, 0.53542]
# accuracy_arr2 =  [0.34202, 0.41472, 0.43708, 0.46606, 0.47888]


# accuracy_arr2 =  [0.32962, 0.41138, 0.43526, 0.46308, 0.49114]
# accuracy_arr1 =  [0.33266, 0.41292, 0.47982, 0.51108, 0.54498]


import matplotlib.pyplot as plt

input_rows = [20000, 40000, 60000, 80000, 100000]

# plt.subplot(3, 1, 1)

plt.plot(input_rows, accuracy_arr1, input_rows, accuracy_arr2, marker='o')

# plt.title('arrangment_1')

# plt.subplot(3, 1, 2)
# plt.plot(input_rows, , marker='x')
# plt.title('arrangment_2')

# Adding labels and title
plt.xlabel('Training Data Rows')
plt.ylabel('Test Accuracy')
plt.legend(['arrangement_1', 'arrangement_2'])
plt.title('arrangement_1 then arrangement_2')

# Display the plot
# plt.tight_layout()
plt.show()
```

1.



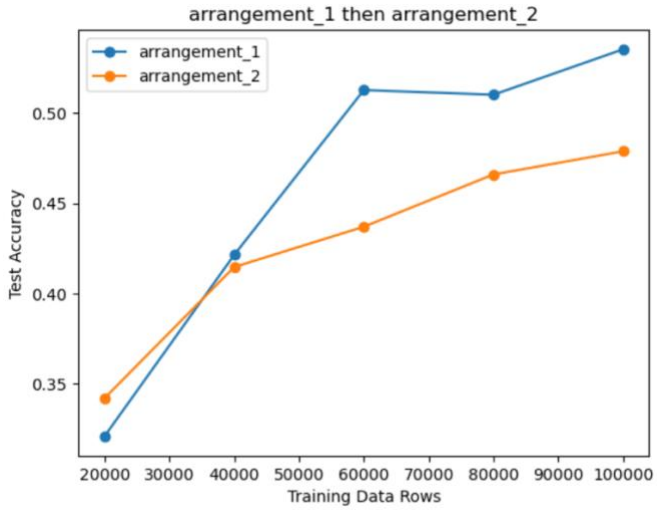
Now, resetting the classifier and reversed the order of training.
Now trained the classifier with arrangement_2 and then arrangement_1. And plotted the graph of accuracy:

2.

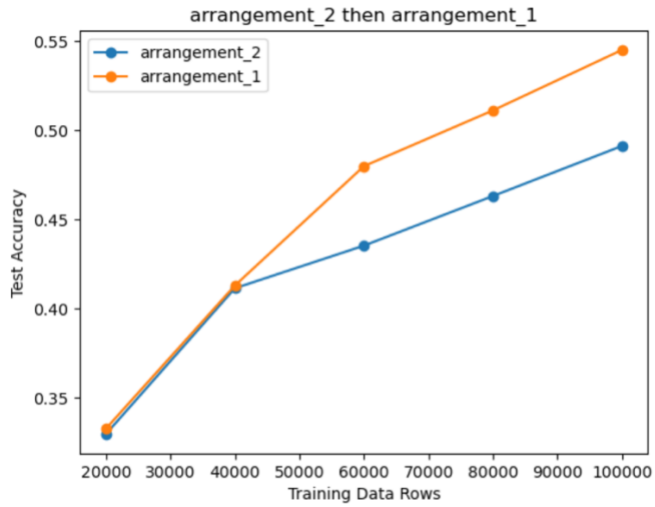


Repeated above two steps again:

3.



4.



Observation: In plot 1 and 3, it appears that the classifier is not able to persist the previous learning, as accuracy is declining

In plot lot 2 and 4 accuracy later, accuracy is higher. Thus, ambiguity is there.

Aug 23, 2023:

More research about how to persist previous learning of classifier in case the dataset changes.

https://machinelearningmastery.com/transfer-learning-for-deep-learning/

https://builtin.com/data-science/transfer-learning

https://neptune.ai/blog/concept-drift-best-practices

https://en.wikipedia.org/wiki/Catastrophic_interference#:~:text=When%20such%20deep%20generative%20models,than%20at%20the%20input%20level.

Aug 24, 2023:

Analyzed how accuracy varies:

Results: if dataset changes, classifier forgets the older learning and gradually learns the new dataset.

If I train the classifier with first dataset and reaches maximum accuracy of 53 %, and then train with a part of second dataset accuracy goes down 19% for older examples but for new examples it goes up from 0% to 34% and it gradually increases till max accuracy of 47% if we keep on training with new dataset(second one)

X_set_1 => [.53, .34(.19), .39, .41, .43, .47]

Similarly, if we reverse the dataset and train it with second dataset first.

X_set_2 => [.48, .39(.25), .44, .49, .51, .54]

Aug 25, 2023:

Research about Evolving classifiers

Reference : https://arxiv.org/pdf/0709.3965.pdf

Aug 28, 2023 & Aug 29, 2023:

Ran demo tutorial and research about Continual Learning

References :

- https://github.com/ContinualAI/colab
- https://www.continualai.org
- https://wiki.continualai.org/the-continualai-wiki/introduction-to-continual-learning

Scope of Neural network in previous memory utilization:

Reference:

- https://www.coursera.org/learn/advanced-learning-algorithms/
- https://medium.com/@satnalikamayank12/on-learning-embeddings-for-categorical-data-using-keras-165ff2773fc9
- https://www.simplilearn.com/tutorials/deep-learning-tutorial/neural-network

Algorithm for managing memory outside classifiers:

1. Train the classifier with, say, 100,000 samples of scenario 1.
2. Obtain the solution as the vector ***x1***
3. Reset the classifier
4. Train the classifier with, say, 100,000 samples of scenario 2.
5. Obtain the solution as the vector ***x2***
6. Calculate: ***x3***$=\alpha \cdot$***x1+(1-α)·x2,*** starting with ***α****=0.1*
7. Obtain the accuracy of the ***x3*** solution for scenario 2
8. Plot on the graph the accuracy as a function of ***α***
9. Repeat steps 1) through 8) for ***α*** changing from 0 to 1, in steps of 0.1 . You will end up with 11 points on the graph.

Aug 30, 2023:

We have 2 training datasets for tag **scenario_1**[X_set_1(instances), y_set_1(solutions)]and **scenario_2** (X_set_2, y_set_2). Which we will use for the training of classifier sequentially.

Training dataset looks like following:

```
: display(X_set_1)
array([[16, 13,  0, 18,  0, 17, 19, 13,  0,  0,  0, 15,  0, 10, 16,  0,
        6, 12,  0,  0],
       [16,  0,  0,  0,  0,  0,  5, 13,  0,  0,  0, 15,  0, 16,  0, 15,
        6,  0,  0, 11],
       [ 0,  0, 11,  0,  7, 17,  3,  0,  0, 14,  3, 18,  0, 10, 12,  1,
        0,  0,  7,  2],
       [ 0,  6,  0, 18,  1, 17,  5,  0,  0, 14,  3, 15,  8,  4, 16,  1,
```

(nodes want to transfer to the destination)

```
display(y_set_1)
array([[1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0],
       [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
```

(solution)

We are considering a **test** dataset(**X_test_2**, **y_test_2**) which is generated for scenario 2. Over this we will be testing our combined prediction of both classifiers. This dataset has 50k rows.

Training the classifier for scenario 1:

```
: # building the classifier.

from sklearn.tree import DecisionTreeClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import train_test_split

# Define the classifier
clf = MultiOutputClassifier(DecisionTreeClassifier(random_state=42, max_depth = 16, min_samples_split = 16,
                                                   min_samples_leaf=4))


# Train the classifier
clf.fit(X_set_1, y_set_1)
```

Finding accuracy for this classifier for some unseen test dataset of scenario 1:

```
score = clf.score(X_test_1, y_test_1)
print('Test Accuracy:', score)
```

```
Test Accuracy: 0.74364
```

Storing solution for scenario_1 in **y_pred_1** (x1). Having 50k rows. **y_pred_1** stores predicted solution by classifier 1 for our **test** dataset(X_test_2, y_test_2).

```
# Test the classifier
y_pred_1 = clf.predict(X_test_2)
y_pred_1
```

```
array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0],
       [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0],
       [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0],
       [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0],
       [0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]
```

Resetting the classifier:

```
clf = None
```

training again for scenario 2.

```
# building the classifier.

from sklearn.tree import DecisionTreeClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import train_test_split

# Define the classifier
clf = MultiOutputClassifier(DecisionTreeClassifier(random_state=42, max_depth = 14, min_samples_split = 16,
                                                   min_samples_leaf=4))

# Train the classifier
clf.fit(X_set_2, y_set_2)
```

Accuracy for unseen data of scenario 2:

```
score = clf.score(X_test_2, y_test_2)
print('Test Accuracy:', score)
```

```
Test Accuracy: 0.56388
```

Note: Accuracy is lesser for scenario 2 but it is different issue which I am looking concurrently. But for now considering 56% is the highest accuracy for next arrangement.

Storing solution for scenario_2 in **y_pred_2** (x2). Having **50k** rows:

```
y_pred_2 = clf.predict(X_test_2)
y_pred_2
```

```
array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
       [1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1],
       [1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0],
       [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0],
       [1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0],
       [1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
       [0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0]])
```

Now, calculating **x3** for **alpha** values [0.1 – 0.9] , storing nine x3 sets for each alpha namely [**y_pred_3_1** to **y_pred_3_9**]

Displaying **y_pred_3_1 , y_pred_3_5 and y_pred_3_9:**

```
y_pred_3_1
```

```
array([[1. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 1. , 1. , 1. , 0. ,
        0. , 0. , 0. , 1. , 0. , 0. , 0. ],
       [0. , 0. , 0. , 1. , 1. , 1. , 0. , 0.1, 0. , 0. , 1. , 0. , 0. ,
        0. , 1. , 0. , 0. , 0. , 0. , 0. ],
       [0. , 0.9, 0. , 0. , 0. , 0.1, 0. , 0.1, 0.9, 1. , 1. , 1. , 0. ,
        0. , 1. , 0. , 0. , 0. , 0.1, 0. ],
       [1. , 0.9, 1. , 0. , 0. , 0. , 0. , 0.9, 0. , 1. , 0.1, 0. , 0. ,
        0. , 0. , 0. , 0. , 0.9, 0. , 0.9],
       [1. , 1. , 1. , 0. , 0. , 1. , 0. , 0. , 0. , 1. , 0. , 0. , 0. ,
        0. , 0. , 0.9, 0. , 0. , 0. , 0.9],
       [0. , 0. , 0. , 0. , 1. , 1. , 0. , 0. , 0. , 1. , 1. , 0. , 1. ,
        0. , 0.1, 0.9, 0. , 0.9, 0. , 0. ],
       [0. , 0. , 1. , 0.1, 1. , 1. , 0. , 0. , 0. , 0. , 0. , 0. , 0.9,
        0.9, 0. , 0. , 0. , 0. , 0. , 0. ],
       [1. , 0. , 0. , 0. , 0. , 1. , 0.9, 0. , 0.1, 0. , 0.9, 1. , 0. ,
        0. , 0. , 0. , 0. , 0. , 0.9, 0. ],
       [0. , 0. , 1. , 0. , 0. , 1. , 1. , 0.1, 0. , 1. , 0. , 1. , 0. ,
        0. , 1. , 0. , 0. , 0. , 0. , 0. ],
       [1. , 0. , 1. , 0.9, 0. , 1. , 0. , 0.1, 0. , 0. , 0.1, 0. , 0. ,
        0. , 0. , 0. , 0.9, 0. , 0. , 0. ],
       [1. , 1. , 1. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 1. , 0. ,
```

```
y_pred_3_5
```

```
array([[1. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 1. , 1. , 1. , 0. ,
        0. , 0. , 0. , 1. , 0. , 0. , 0. ],
       [0. , 0. , 0. , 1. , 1. , 1. , 0. , 0.5, 0. , 0. , 1. , 0. , 0. ,
        0. , 1. , 0. , 0. , 0. , 0. , 0. ],
       [0. , 0.5, 0. , 0. , 0. , 0.5, 0. , 0.5, 0.5, 1. , 1. , 1. , 0. ,
        0. , 1. , 0. , 0. , 0. , 0.5, 0. ],
       [1. , 0.5, 1. , 0. , 0. , 0. , 0. , 0.5, 0. , 1. , 0.5, 0. , 0. ,
        0. , 0. , 0. , 0. , 0.5, 0. , 0.5],
       [1. , 1. , 1. , 0. , 0. , 1. , 0. , 0. , 0. , 1. , 0. , 0. , 0. ,
        0. , 0. , 0.5, 0. , 0. , 0. , 0.5],
       [0. , 0. , 0. , 0. , 1. , 1. , 0. , 0. , 0. , 1. , 1. , 0. , 1. ,
        0. , 0.5, 0.5, 0. , 0.5, 0. , 0. ],
       [0. , 0. , 1. , 0.5, 1. , 1. , 0. , 0. , 0. , 0. , 0. , 0. , 0.5,
        0.5, 0. , 0. , 0. , 0. , 0. , 0. ],
       [1. , 0. , 0. , 0. , 0. , 1. , 0.5, 0. , 0.5, 0. , 0.5, 1. , 0. ,
        0. , 0. , 0. , 0. , 0. , 0.5, 0. ],
       [0. , 0. , 1. , 0. , 0. , 1. , 1. , 0.5, 0. , 1. , 0. , 1. , 0. ,
        0. , 1. , 0. , 0. , 0. , 0. , 0. ],
       [1. , 0. , 1. , 0.5, 0. , 1. , 0. , 0.5, 0. , 0. , 0.5, 0. , 0. ,
        0. , 0. , 0. , 0.5, 0. , 0. , 0. ],
       [1. , 1. , 1. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 1. , 0. ,
```

```
y_pred_3_9
```

```
array([[1. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 1. , 1. , 1. , 0. ,
        0. , 0. , 0. , 1. , 0. , 0. , 0. ],
       [0. , 0. , 0. , 1. , 1. , 1. , 0. , 0.9, 0. , 0. , 1. , 0. , 0. ,
        0. , 1. , 0. , 0. , 0. , 0. , 0. ],
       [0. , 0.1, 0. , 0. , 0. , 0.9, 0. , 0.9, 0.1, 1. , 1. , 1. , 0. ,
        0. , 1. , 0. , 0. , 0. , 0.9, 0. ],
       [1. , 0.1, 1. , 0. , 0. , 0. , 0. , 0.1, 0. , 1. , 0.9, 0. , 0. ,
        0. , 0. , 0. , 0. , 0.1, 0. , 0.1],
       [1. , 1. , 1. , 0. , 0. , 1. , 0. , 0. , 0. , 1. , 0. , 0. , 0. ,
        0. , 0. , 0.1, 0. , 0. , 0. , 0.1],
       [0. , 0. , 0. , 0. , 1. , 1. , 0. , 0. , 0. , 1. , 1. , 0. , 1. ,
        0. , 0.9, 0.1, 0. , 0.1, 0. , 0. ],
       [0. , 0. , 1. , 0.9, 1. , 1. , 0. , 0. , 0. , 0. , 0. , 0. , 0.1,
        0.1, 0. , 0. , 0. , 0. , 0. , 0. ],
       [1. , 0. , 0. , 0. , 0. , 1. , 0.1, 0. , 0.9, 0. , 0.1, 1. , 0. ,
        0. , 0. , 0. , 0. , 0. , 0.1, 0. ],
       [0. , 0. , 1. , 0. , 0. , 0. , 1. , 1. , 0.9, 0. , 1. , 0. , 1. , 0. ,
        0. , 1. , 0. , 0. , 0. , 0. , 0. ],
       [1. , 0. , 1. , 0.1, 0. , 1. , 0. , 0.9, 0. , 0. , 0.9, 0. , 0. ,
        0. , 0. , 0. , 0.1, 0. , 0. , 0. ],
       [1. , 1. , 1. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 1. , 0. ,
```

Storing accuracy calculation for each x3 in a vector **accuracy_vec** comparing with actual solution y_test_2.

In [33]:
```
# checking if value is greater than threshold 0.5
y_pred_3_1 = np.where(y_pred_3_1 > 0.5, 1, 0)
y_pred_3_2 = np.where(y_pred_3_2 > 0.5, 1, 0)
y_pred_3_3 = np.where(y_pred_3_3 > 0.5, 1, 0)
y_pred_3_4 = np.where(y_pred_3_4 > 0.5, 1, 0)
y_pred_3_5 = np.where(y_pred_3_5 > 0.5, 1, 0)
y_pred_3_6 = np.where(y_pred_3_6 > 0.5, 1, 0)
y_pred_3_7 = np.where(y_pred_3_7 > 0.5, 1, 0)
y_pred_3_8 = np.where(y_pred_3_8 > 0.5, 1, 0)
y_pred_3_9 = np.where(y_pred_3_9 > 0.5, 1, 0)
```

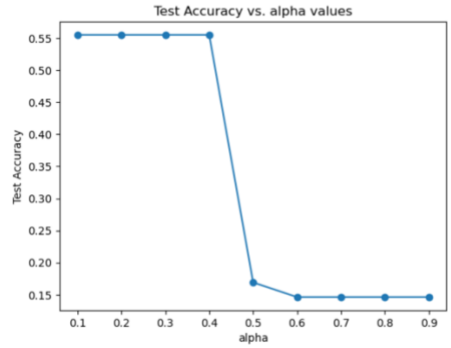
In [34]:
```
count_1 =  np.sum(np.all(y_pred_3_1 == y_test_2, axis=1))
count_2 =  np.sum(np.all(y_pred_3_2 == y_test_2, axis=1))
count_3 =  np.sum(np.all(y_pred_3_3 == y_test_2, axis=1))
count_4 =  np.sum(np.all(y_pred_3_4 == y_test_2, axis=1))
count_5 =  np.sum(np.all(y_pred_3_5 == y_test_2, axis=1))
count_6 =  np.sum(np.all(y_pred_3_6 == y_test_2, axis=1))
count_7 =  np.sum(np.all(y_pred_3_7 == y_test_2, axis=1))
count_8 =  np.sum(np.all(y_pred_3_8 == y_test_2, axis=1))
count_9 =  np.sum(np.all(y_pred_3_9 == y_test_2, axis=1))
```

In [35]:
```
total = y_test_2.shape[0]

accuracy_vec = []
accuracy_vec.append( count_1 / total )
accuracy_vec.append( count_2 / total )
accuracy_vec.append( count_3 / total )
accuracy_vec.append( count_4 / total )
accuracy_vec.append( count_5 / total )
accuracy_vec.append( count_6 / total )
accuracy_vec.append( count_7 / total )
accuracy_vec.append( count_8 / total )
accuracy_vec.append( count_9 / total )
accuracy_vec
```

Out[35]: `[0.5551, 0.5551, 0.5551, 0.5551, 0.16904, 0.14624, 0.14624, 0.14624, 0.14624]`

Plotting graph of accuracy against **alpha** values:

```
import matplotlib.pyplot as plt

alpha_vec = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,  0.9]

plt.plot(alpha_vec, accuracy_vec, marker='o')

# Adding labels and title
plt.xlabel('alpha')
plt.ylabel('Test Accuracy')
plt.title('Test Accuracy vs. alpha values')

# Display the plot
plt.show()
```



Aug 31, 2023 & Sep 01, 2023

Revisiting the accuracy calculation:
Unlike previous accuracy calculations, we are checking fraction of nodes classifier is predicting correctly for each solution. Then taking average of all calculated fractions for the prediction set.
Definition of the function **accuracy_function()** responsible of calculating accuracy is fillowing.

```
def accuracy_function(pred_solution, test_solution):
    # Compare each pair of rows
    elementwise_comparison = (pred_solution == test_solution)

    # Count the number of equal elements in each pair
    equal_counts = np.sum(elementwise_comparison, axis=1)

#     print("Number of equal elements in each pair of rows:")
#     print(equal_counts)

    average = np.sum(equal_counts, axis = 0) / (pred_solution.shape[0] * pred_solution.shape[1])

    return average
```

Example:

Classifier training over some training data X_set_3 , y_set_3

```
# building the classifier.

from sklearn.tree import DecisionTreeClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import train_test_split

# Define the classifier
clf = MultiOutputClassifier(DecisionTreeClassifier(random_state=42, max_depth = 16, min_samples_split = 16,
                                                   min_samples_leaf=4))


# Train the classifier
clf.fit(X_set_3, y_set_3)
```

Older accuracy calculation:

Accuracy over unseen data **X_test_3**(instances) **, y_test_3**(solutions)

```
score = clf.score(X_test_3, y_test_3)
print('Test Accuracy:', score)
```

```
Test Accuracy: 0.7185333333333334
```

New accuracy calculation:

First taking out predictions of the classifier over unseen instance set **X_test_3.**

```
# predictions using classifier
y_pred_1 = clf.predict(X_test_3)
y_pred_1
```

```
array([[0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0],
       [1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0],
       [1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0],
       [1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
       [1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
```

Calculating accuracy using our function, with prediction and actual solution set.

```
In [113]: print(accuracy_function(y_pred_1, y_test_3))
```

```
array([[20, 20, 19, 20, 20, 20, 20, 20, 19, 20, 20, 20, 20, 15, 16, 20, 20, 20, 20, 20, 20, 19, 19],
       [20, 20, 16, 18, 20, 20, 20, 20, 20, 18, 20, 20, 20, 19, 20, 19, 20, 20, 19, 20, 20, 20, 20, 20],
       [20, 20, 20, 20, 20, 20, 20, 19, 20, 16, 19, 20, 20, 20, 16, 20, 20, 20, 20, 20, 20, 20, 20, 20],
       [20, 20, 20, 20, 20, 20, 20, 16, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 19, 15, 20, 19, 19, 20],
       [20, 20, 20, 20, 19, 20, 20, 20, 18, 20, 19, 19, 20, 17, 20, 19, 20, 20, 20, 20, 20, 16, 20, 20],
       [20, 20, 19, 19, 20, 20, 17, 20, 20, 19, 20, 20, 20, 17, 20, 19, 18, 18, 20, 20, 20, 17, 20, 20],
       [20, 20, 20, 18, 19, 18, 20, 20, 19, 20, 18, 20, 20, 20, 20, 18, 20, 15, 20, 20, 20, 20, 20, 19, 20],
       [18, 20, 19, 20, 20, 20, 17, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 18, 20, 20, 20, 19],
       [18, 20, 19, 18, 18, 20, 19, 20, 20, 19, 19, 20, 17, 20, 20, 20, 18, 20, 20, 20, 20, 20, 20, 20],
       [20, 20, 20, 20, 20, 20, 20, 19, 20, 20, 20, 18, 20, 20, 20, 20, 20, 19, 17, 20, 20, 20, 19, 20],
       [20, 20, 20, 20, 19, 20, 20, 19, 19, 20, 20, 20, 20, 18, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20],
       [20, 18, 20, 20, 20, 20, 19, 16, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 19],
       [20, 20, 20, 20, 19, 20, 17, 20, 20, 20, 19, 20, 20, 18, 18, 17, 20, 20, 20, 20, 19, 17, 20, 19],
       [18, 20, 20, 19, 20, 20, 20, 20, 20, 20, 20, 20, 17, 20, 20, 20, 20, 20, 20, 18, 20, 18, 20, 20],
       [19, 20, 20, 18, 20, 19, 19, 20, 20, 20, 20, 20, 20, 20, 17, 16, 20, 19, 19, 20, 17, 19, 19, 20],
       [20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20],
       [17, 19, 20, 19, 20, 19, 20, 20, 20, 20, 20, 20, 18, 20, 20, 20, 20, 20, 20, 20, 20, 20, 19, 20],
       [20, 19, 20, 20, 20, 20, 20, 20, 20, 20, 18, 18, 20, 19, 20, 20, 20, 20, 14, 20, 20, 20, 20, 20],
       [20, 20, 20, 20, 20, 20, 20, 19, 20, 20, 20, 19, 19, 18, 20, 20, 18, 19, 18, 20, 20, 20, 19, 20]])
0.9747833333333333
```

Here, accuracy value is much higher than before, because before even 1 bit in the solution was wrong whole solution was considered wrong. Now, instead we are considering number of bits correctly predicted.

Analyzing accuracy behavior when tags move:

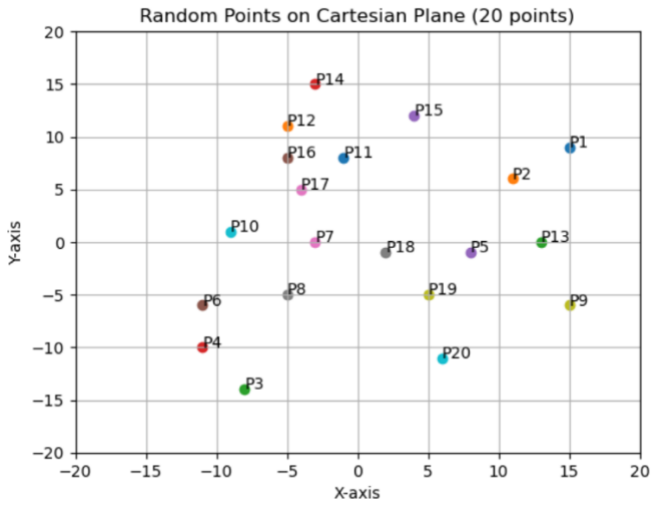
I generated other scenarios from scratch and moved some nodes and for both scenarios' accuracies are nearly same.

Maybe the inconsistency in accuracies were for one particular combination of arrangement, which I am looking further into.

Sep 05, 2023

There was a mistake in the last observation. After analyzing some more, using datasets for new tag arrangements, it is observed that accuracy is going down when some nodes are moved. Further explanation is as follow:

Let's start following tag arrangement:



(set 5)

Transmission radius is same(i.e. 8):

Training dataset (instances(**X_set_1**) and solutions(**y_set_1**) for this scenario_1 is shown below:

```
display(X_set_1)
```

```
array([[ 0,  1,  4,  0,  0,  4, 17,  0, 13,  0,  0,  0,  0,  0, 11, 12,
        16,  0,  0, 19],
       [ 2,  1,  4,  0,  2,  0,  8, 10,  0,  0, 12,  0,  5,  0, 11,  0,
         7,  0, 20, 19],
       [ 0,  0,  4,  6, 13,  8, 18,  4,  0,  0, 15,  0,  9,  0,  0, 11,
         0,  0,  0,  0],
       [ 2,  1,  0,  0,  0,  0,  0,  0, 13,  8,  0,  0,  0, 15,  0, 17,
         0,  0,  0,  0],
       [ 2,  0,  4,  8,  2,  8,  0,  6, 13,  8, 16, 14,  0, 16, 14, 12,
         0,  5,  0,  0],
       [ 2,  0,  0,  8, 13,  4,  0, 10,  0,  0, 15,  0,  5, 12, 14,  0,
         0,  0,  0,  0],
       [ 2,  5,  4,  3,  0, 10,  0,  4, 13,  0, 14, 11,  2, 11,  0,  0,
        12,  7,  0,  0],
       [ 0,  1,  0,  0, 13,  8, 10,  0,  0,  0,  0,  0,  5, 12, 11,  0,
         7,  0,  0, 19],
       [ 0,  0,  4,  3, 18,  4,  8,  4, 13, 17, 15, 16,  0, 16, 14, 14,
        12,  0, 18, 19],
       [ 2, 13,  4,  6,  2,  8,  0,  6, 13,  7,  0, 17,  0,  0, 14, 17,
         0,  7, 18, 19]
```

```
display(y_set_1)
```

```
array([[0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0],
       [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
```

Test dataset (instances(**X_test_1**) and solutions(**y_test_1**) for this scenario 1 is shown below:

```
display(X_test_1)
```

```
array([[ 0,  5,  0,  0,  0,  0, 17,  4, 13,  6,  0, 16,  2, 16, 11,  0,
         0, 19, 20, 19],
       [ 2,  0,  4,  8, 13,  0, 17,  7, 13,  7, 14,  0,  2, 12,  0,  0,
        12,  7, 18,  0],
       [ 0,  0,  4,  0,  2,  0,  0,  0, 13, 17,  0, 11,  0,  0, 11,  0,
        12,  5,  5,  0],
       [ 2,  1,  4,  0, 19,  0, 18,  7, 13,  8,  0,  0,  2,  0, 11, 14,
        12,  5,  5, 19],
       [ 2,  0,  0,  3, 18,  0, 17,  4, 13,  0,  0,  0,  5, 11,  0,  0,
        11,  7,  0, 19],
       [ 2,  5,  4,  8,  0,  4, 18,  7, 13,  6,  0,  0,  0, 16, 14, 14,
         7,  7,  5, 19],
       [ 2,  0,  0,  0,  0,  0, 10,  0, 13,  0, 14, 16,  9,  0,  0,  0,
         0,  0, 18,  0],
       [ 2,  0,  0,  3,  0,  0,  0,  0, 13,  7, 16,  0,  9, 15,  0,  0,
         0,  5, 18,  0],
       [ 2, 13,  4,  0, 13,  0, 10,  6, 13,  0, 15,  0,  0,  0, 11, 14,
         7, 19,  0, 19],
       [ 2,  1,  4,  0,  0,  8,  0,  4,  0,  7,  0, 16,  0, 12, 14,  0,
         7,  5, 20, 19],
       [ 0,  5,  4,  8, 18,  0,  0,  0, 13,  8,  0, 16,  0,  0, 11, 11,
```

```
display(y_test_1)
```

```
array([[0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
       [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
       [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
```

Training the classifier with training dataset:

```
]: # building the classifier.

   from sklearn.tree import DecisionTreeClassifier
   from sklearn.multioutput import MultiOutputClassifier
   from sklearn.model_selection import train_test_split

   # Define the classifier
   clf = MultiOutputClassifier(DecisionTreeClassifier(random_state=42, max_depth = 16, min_samples_split = 16,
                                                      min_samples_leaf=4))
   # Train the classifier
   clf.fit(X_set_1, y_set_1)
```

Accuracy using default function:

```
score = clf.score(X_set_1, y_set_1)
print('Train Accuracy:', score)

score = clf.score(X_test_1, y_test_1)
print('Test Accuracy:', score)
```

```
Train Accuracy: 0.88045
Test Accuracy: 0.82336
```

Accuracy using our custom function:

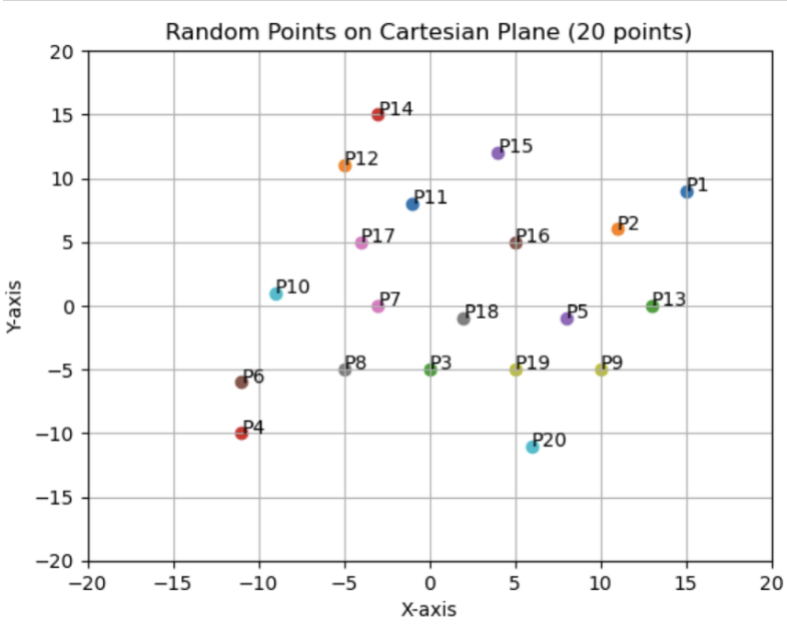
```
# predictions using classifier
y_pred_1 = clf.predict(X_test_1)
y_pred_1
```

```
array([[0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1
        [1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0
        [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0
        [1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1
        [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1
        [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0
        [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0
        [1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0
        [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0
        [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0
        [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0
        [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
        [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1
        [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0
        [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0
        [1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1
        [1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1
        [0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1
        [0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0
```

```
print(accuracy_function(y_pred_1, y_test_1))
```

```
0.984423
```

Now suppose three nodes change their position, respectively node 3, 9 and 16 (**scenario_2**):



Training Datasets for scenario 2 (**X_set_2**, **y_set_2**):

```
display(X_set_2)
```

```
array([[ 0,  0,  0,  0,  0,  8, 10,  0, 19,  7,  0,  0,  0, 11,  0,  0,
         7,  0,  5, 19],
       [ 2,  0, 18,  8,  2, 10,  0,  0,  5,  0, 15, 11,  2, 15, 14, 15,
        11, 16,  0, 19],
       [ 2,  5, 18,  0, 13,  4,  0,  0, 13, 17,  0, 17,  9, 11, 16, 15,
        12, 16,  3,  9],
       [ 0, 13, 19,  0,  0,  0,  8,  6, 13,  6, 17, 11,  0, 12,  0,  5,
        10,  3,  9, 19],
       [ 2,  0,  7,  0, 13,  0,  0,  3,  5,  8,  0, 14,  5, 11,  0, 11,
        10,  0,  9,  9],
       [ 2,  5,  8,  6,  0, 10,  3,  3, 19, 17, 12,  0,  2,  0,  0,  5,
        11,  0,  0, 19],
       [ 2, 13,  8,  0, 16,  8,  3,  7,  0, 17, 16,  0,  0,  0, 16,  0,
         0,  3, 20,  9],
       [ 0,  5,  0,  6,  0,  0,  0, 10,  5,  6, 17, 14,  0, 11, 11,  2,
        12,  0,  0,  0],
       [ 2,  0,  0,  0,  0,  4,  0,  0,  0,  0, 12, 17,  9, 11, 11, 11,
         0,  7,  0,  0],
       [ 2, 13,  0,  8,  2,  4,  3,  0,  0,  8,  0, 17,  5,  0, 11,  0,
        12,  7,  3,  9],
       [ 2, 16,  0,  8, 19,  8,  0,  0,  5,  0, 15,  0,  2, 11, 11,  0
```

```
display(y_set_2)
```

```
array([[0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
       [1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
```

Test Dataset for scenario 2:

```
display(X_test_2)
```

```
array([[ 0,  5, 19,  8, 13,  0,  0,  0,  0,  8,  0,  0,  0, 15, 14,  0,
        11, 19,  0, 19],
       [ 2, 16,  8,  8,  0,  0,  8,  0, 13, 17,  0, 17,  5, 15, 11, 15,
         0,  5,  9,  9],
       [ 0,  0,  0,  0, 18, 10,  0,  7,  0,  6, 15, 14,  9,  0,  0, 18,
         7,  3,  5,  9],
       [ 2,  0,  0,  8,  0, 10, 10,  6, 19,  0,  0, 14,  0, 11,  0, 18,
        10,  0, 18, 19],
       [ 0,  1,  0,  0, 13,  8, 10,  7, 20,  7, 12, 11,  0, 12, 16,  5,
        11,  5,  3,  9],
       [ 2,  5,  0,  0, 18,  8, 18,  3,  5,  0, 16, 17,  2,  0, 11,  2,
         0,  5, 20, 19],
       [ 2,  1,  7,  6, 18,  0,  0,  7, 20, 17,  0, 14,  0,  0, 16,  0,
        12,  5,  5,  0],
       [ 2,  0,  7,  0, 18,  8, 17,  0,  0,  8, 16, 17,  9,  0, 14,  0,
        10,  5,  0,  9],
       [ 2, 16, 19,  8, 13, 10,  8,  6,  0,  7, 16,  0,  5,  0, 14, 15,
        10,  0, 20,  9],
       [ 0, 13, 19,  8,  0, 10,  0, 10, 20,  7,  0,  0,  0,  0, 11,  0,
        11,  5,  0,  0],
       [ 2,  0,  7,  0, 19,  4,  8,  0,  0,  0, 12, 17,  5, 11,  0, 11
```

```
display(y_test_2)
```

```
array([[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
```

Now resetting the classifier and training the classifier with this dataset and checking the accuracy:

```
clf = None
```

```
# building the classifier.

from sklearn.tree import DecisionTreeClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import train_test_split

# Define the classifier
clf = MultiOutputClassifier(DecisionTreeClassifier(random_state=42, max_depth = 16, min_samples_split = 16,
                                                   min_samples_leaf=4))

# Train the classifier
clf.fit(X_set_2, y_set_2)
```

Accuracy using default function:

```
score = clf.score(X_set_2, y_set_2)
print('Test Accuracy:', score)
score = clf.score(X_test_2, y_test_2)
print('Test Accuracy:', score)
```

```
Test Accuracy: 0.57865
Test Accuracy: 0.3972
```

Accuracy using our custom function:

```
y_pred_2 = clf.predict(X_test_2)
y_pred_2
```

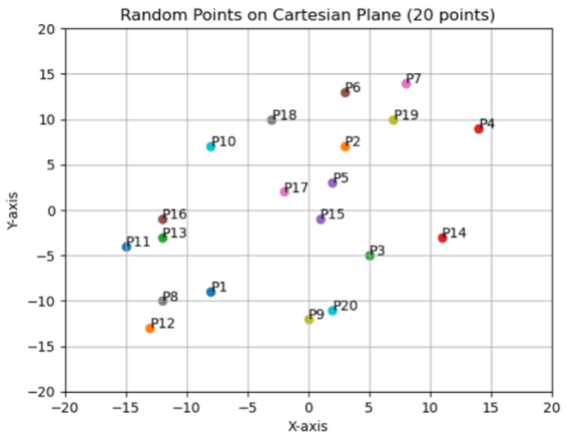
```
array([[1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
       [0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
```

```
print(accuracy_function(y_pred_2, y_test_2))
```

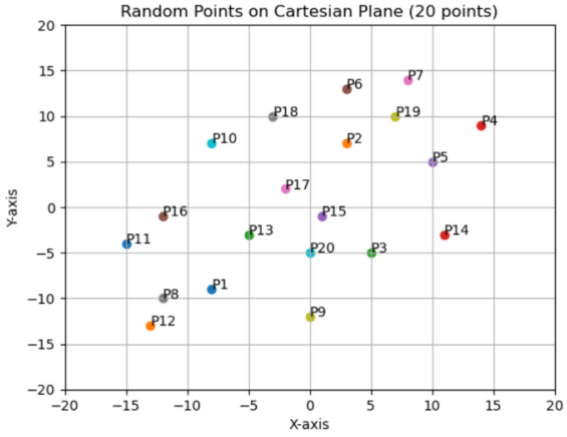
```
0.934743
```

Here, we can see accuracy went down:

Now, repeating the same with another arrangement of tags:



(Scenario 1)



(Scenario 2: nodes 5, 13 & 20 moved)

Datasets for scenario 1:

Training set

```
display(X_set_1)
```

```
array([[ 8,  5,  0,  0,  0,  2,  4, 13, 20,  0,  0,  0,  1,  3,  0, 11,
         2,  6,  0,  9],
       [ 8,  0, 14, 19, 15,  7, 19,  0, 20,  0,  0,  0,  0,  0,  0,  0,
         5,  0,  0,  3],
       [ 8, 17,  0, 19,  2,  0,  4,  0, 20, 18, 16,  1, 16,  3,  0,  0,
         0,  0,  0,  9],
       [12,  6, 14,  0,  0,  0,  4, 11, 20, 18,  0,  8,  1,  3,  0, 13,
        15,  2,  2,  3],
       [ 0,  0,  0,  0, 17,  0,  0, 11, 20,  0,  0,  0,  0,  3,  5,  0,
         0,  0,  7,  0],
       [ 8,  0, 14, 19,  2, 18,  6,  0,  0, 17,  8,  1,  0,  3, 17, 11,
         2, 10,  7,  3],
       [ 8,  0,  0,  7,  2, 19, 19,  1, 20,  0,  0,  1,  0,  0,  5,  0,
         2,  0,  7,  0],
       [ 0,  0,  0,  7, 15,  7,  0, 11, 20, 18, 13,  8,  8,  0,  3, 13,
         0,  0,  6,  9],
       [ 0,  0, 20,  7, 15, 18, 19, 12,  0, 18,  0,  1, 16,  0,  3, 13,
        10, 10,  4,  0],
       [ 0,  6, 20,  0,  2,  0,  0, 13,  0, 17, 13,  8,  8,  3,  0,  0,
        10,  0,  0,  0],
       [ 8,  6, 15, 19, 15, 18,  6, 11, 20, 17,  0,  0,  0,  0, 17, 13,
```

```
display(y_set_1)
```

```
array([[1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0],
       [1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0],
       [0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
       [0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0],
```

Testing set

```
display(X_test_1)
```

```
array([[13, 19, 20,  0,  0, 19,  4, 11,  0,  0,  0,  1,  0,  3,  0, 13,
         0,  2,  2,  0],
       [ 0,  0,  0,  0, 17,  0,  4, 12,  0, 18, 13,  1,  8,  3,  0, 13,
         5,  6,  7,  0],
       [ 8,  0, 20, 19,  0,  2,  0, 12,  0, 17, 13,  8,  1,  0,  5, 11,
         5,  6,  0,  9],
       [12,  6, 15,  0,  0, 19,  6, 12,  0, 17, 16,  1,  0,  0, 17,  0,
         2, 10,  0,  9],
       [ 8,  0, 14,  0, 17,  0,  4, 11,  0,  0, 13,  0,  0,  3,  3, 13,
         5,  0,  4,  0],
       [ 0,  0, 14,  7,  0,  7,  0,  0, 20,  0, 13,  8,  0,  0,  3, 13,
         0,  2,  0,  0],
       [12,  0, 15, 19,  2,  0, 19, 11,  0, 17, 16,  0,  0,  3,  5, 11,
         0,  0,  6,  0],
       [13, 17, 15, 19,  2, 19, 19,  0,  0,  0,  0,  0, 11,  3,  5, 13,
         0,  6,  0,  0],
       [13,  0, 14,  7,  0,  2,  6, 13,  0,  0,  8,  8,  1,  3, 17, 11,
         0,  0,  2,  3],
       [ 8, 19, 14, 19,  2, 19,  4,  0,  0,  0,  0,  1, 16,  0,  0,  0,
         0, 10,  6,  9],
       [ 0,  0, 15,  7, 17,  0, 19, 11, 20, 17,  8,  8,  8,  0,  3,  0,
```

```
display(y_test_1)
```

```
array([[0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0],
       [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1],
       [0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0],
       [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1],
       [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0],
```

Datasets for scenario 2 :

```
display(X_set_2)
```

```
array([[ 0,  6, 15,  0,  2,  0,  6,  0, 20, 17,  0,  0,  1,  3, 13,  0,
         0, 10,  0, 13],
       [ 0, 18, 15,  0,  2, 18, 19,  0, 20, 17,  8,  8, 16,  3, 17, 13,
         0,  0,  7,  3],
       [ 0,  0, 20, 19, 19,  7,  0, 11, 20, 17, 16,  0,  0,  0,  0, 11,
        15,  2,  4,  3],
       [13,  0, 14,  7,  4,  0,  0, 12, 20, 18,  0,  0,  0,  3,  3, 11,
        13,  6,  6, 17],
       [ 0,  0, 14, 19,  4,  7,  0,  0,  0,  0,  8,  0, 17,  0, 13,  0,
        10,  2,  7,  0],
       [ 8, 18, 14,  0,  0, 18,  0,  1,  0, 18,  0,  1, 15,  3, 17, 11,
         0,  0,  7,  0],
       [ 8,  5, 14,  5,  0, 19, 19,  0, 20,  0,  0,  1, 17,  3,  3, 13,
         2,  2,  4, 17],
       [ 8,  0,  0,  0, 19,  0,  0, 11,  0,  0,  8,  8,  0,  0, 20, 11,
        13, 10,  2,  3],
       [ 8,  0, 20,  7, 19, 18,  0, 12,  0,  0, 16,  0, 15,  3,  0, 13,
         0, 10,  0,  0],
       [13,  5,  0,  7, 19,  7, 19,  0,  0, 18, 16,  8, 20,  3,  3, 13,
        10,  0,  2, 13],
       [ 0, 17, 15,  5,  4,  7,  6, 12, 20,  0,  0,  0, 15,  3,  0,  0
```

```
display(y_set_2)
```

```
array([[0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0],
       [0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0],
       [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0],
       [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0],
       [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
```

```
display(X_test_2)
```

```
array([[ 8, 19, 20,  5,  0,  2,  0,  1,  0,  0, 16,  0,  0,  3, 17, 13,
         0,  0,  0,  0],
       [ 0,  0, 20,  7,  2, 18,  0,  0, 20, 17,  8,  8,  1,  3,  0, 13,
        15,  2,  0, 13],
       [13,  5, 14,  5, 19, 18,  6,  1,  0, 18,  0,  8,  0,  0,  0, 13,
         2,  0,  2,  9],
       [ 0,  0,  0,  7,  0,  0,  6,  1, 20,  0,  8,  0,  1,  3,  0,  0,
         0,  0,  0,  0],
       [ 0,  0,  0,  5,  4,  2,  4,  0, 20, 17, 16,  0,  0,  3, 13, 11,
        10,  0,  6,  0],
       [13,  6, 15,  7, 19,  0,  0,  0,  0, 17, 16,  8, 17,  3, 17, 11,
         0,  0,  0,  0],
       [ 0,  5, 14, 19,  4,  0,  4, 11, 20,  0,  8,  8, 20,  3,  0, 11,
         0,  2,  0,  0],
       [ 8, 18,  0,  7, 19,  0,  0,  0, 20,  0,  0,  8,  1,  3, 20,  0,
         0,  6,  0,  9],
       [ 0, 19, 20,  7,  0,  0,  6, 12, 20,  0,  0,  8, 16,  0, 17,  0,
         0,  6,  0,  0],
       [ 0, 19, 15,  5, 19, 18,  0,  0,  0,  0, 16,  0,  0,  3, 13,  0,
         2,  0,  2,  0],
       [ 0,  6,  0,  5,  0, 19,  0,  0,  0, 18,  0,  0,  1,  3,  0, 13
```

```
display(y_test_2)
```

```
array([[0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
       [0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0],
       [1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1],
       [0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
```

Training:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import train_test_split

# Define the classifier
clf = MultiOutputClassifier(DecisionTreeClassifier(random_state=42, max_depth = 16, min_samples_split = 16,
                                                   min_samples_leaf=4))
# Train the classifier
clf.fit(X_set_1, y_set_1)


score = clf.score(X_set_1, y_set_1)
```

Accuracy using default function:

```
score = clf.score(X_set_1, y_set_1)
print('Train Accuracy:', score)

score = clf.score(X_test_1, y_test_1)
print('Test Accuracy:', score)
```

```
Train Accuracy: 0.81097
Test Accuracy: 0.71964
```

Accuracy using custom function:

```
# predictions using classifier
y_pred_1 = clf.predict(X_test_1)
y_pred_1
```

```
array([[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1],
       [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1],
       [1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1],
       [0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1],
       [0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1],
       [0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1],
       [1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1],
       [0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1],
```

```
print(accuracy_function(y_pred_1, y_test_1))
```

```
0.974761
```

Resetting the classifier and training for scenario 2 :

```
# building the classifier.

from sklearn.tree import DecisionTreeClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import train_test_split

# Define the classifier
clf = MultiOutputClassifier(DecisionTreeClassifier(random_state=42, max_depth = 16, min_samples_split = 16,
                                                   min_samples_leaf=4))

# Train the classifier
clf.fit(X_set_2, y_set_2)
```

Accuracy using default function:

```
score = clf.score(X_set_2, y_set_2)
print('Test Accuracy:', score)
score = clf.score(X_test_2, y_test_2)
print('Test Accuracy:', score)
```

```
Test Accuracy: 0.78076
Test Accuracy: 0.67616
```

Accuracy using custom function:

```
y_pred_2 = clf.predict(X_test_2)
y_pred_2
```

```
array([[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0],
       [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0],
       [0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0],
       [1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
       [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0],
       [1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
```

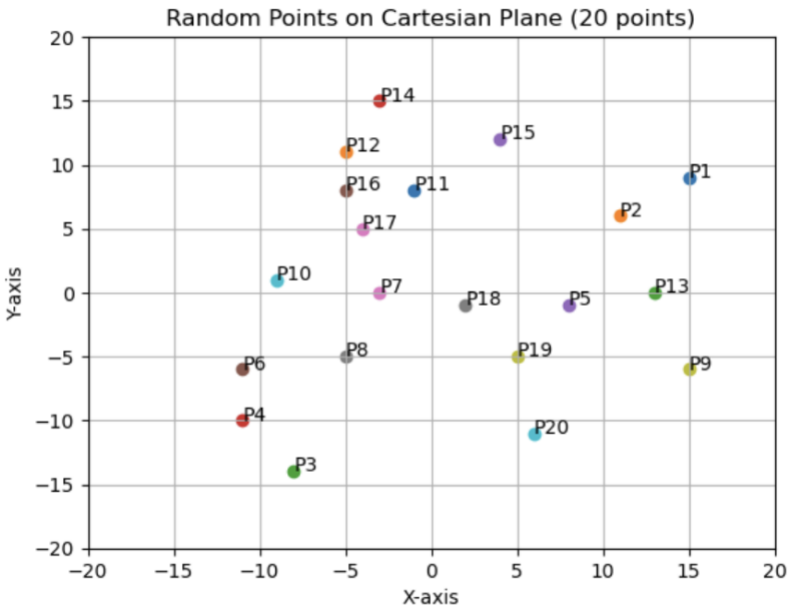
```
print(accuracy_function(y_pred_2, y_test_2))
```

```
0.966314
```

Here accuracy gone down as well.

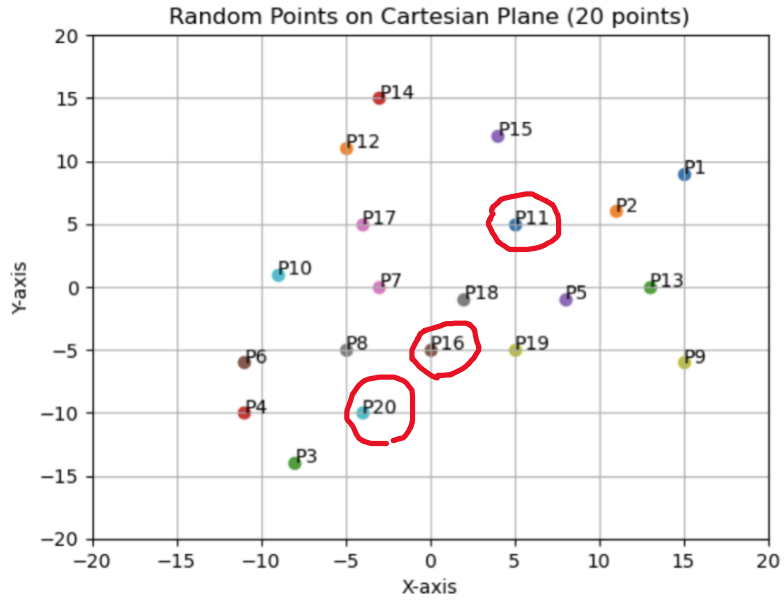
Sep 08, 2023:

Suppose we are starting from the following arrangement. This is our **scenario 1.**



Case A: For scenario 2 , let's first move Node11, Node16 and Node20 **:**

Resulting arrangement is below:



Scenario 1 accuracy:

```
# predictions using classifier
y_pred_1 = clf.predict(X_test_1)
#y_pred_1
print(accuracy_function(y_pred_1, y_test_1))
```

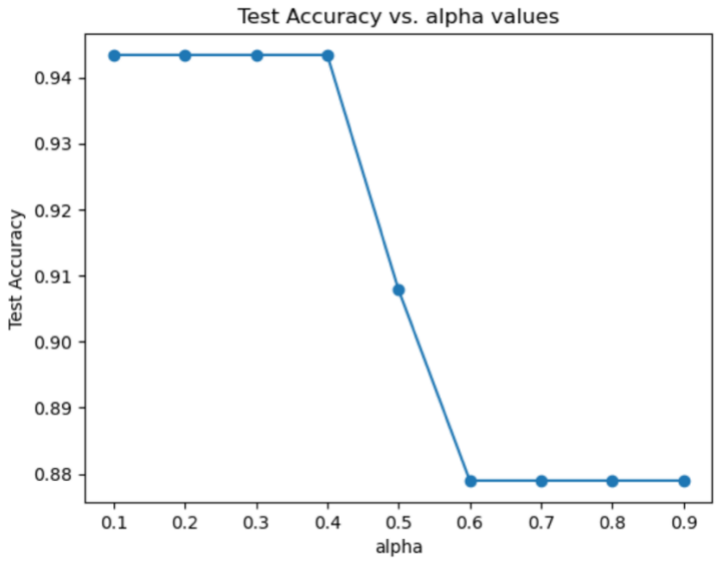
```
0.984209
```

Scenario 2 accuracy:

```
y_pred_2 = clf.predict(X_test_2A)
print(accuracy_function(y_pred_2, y_test_2A))
```

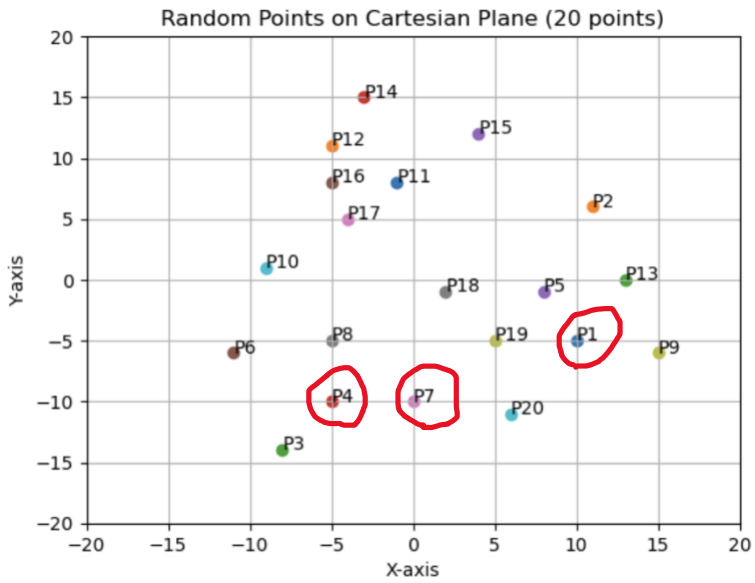
```
0.943396
```

Alpha vs Accuracy graph for this case ;



CASE B:

Case B: For scenario 2 , let's first move Node1, Node4 and Node7 **:**



Scenario 1 accuracy (same as before):

```
# predictions using classifier
y_pred_1 = clf.predict(X_test_1)
#y_pred_1
print(accuracy_function(y_pred_1, y_test_1))
```

```
0.984209
```

Scenario 2 accuracy:

```
y_pred_2 = clf.predict(X_test_2B)
print(accuracy_function(y_pred_2, y_test_2B))
```

```
0.973377
```

Alpha vs Accuracy graph for this case ;