Summary of Object-Oriented Toolbox Design

Class Hierarchy

(1) DataSet: It is a Base Class to construct. It will provide some basic read and load functions but will be overwritten by subclasses. Six subclass will be inherited, which are TimeSeriesDataSet, TextDataSet, QuantDataSet, QualDataSet, TransactionDataSet and HeterogeneousDataSet.

(1.1)TimeSeriesDataSet: It is a subclass that only deals with the TimeSeries dataset and it was inherited from the base class DataSet. It provided a clean function for cleaning the TimeSeries, an explore function for generating two graphs of the TimeSeries Dataset, a function that calculates the median of a list of floats, and a medium filter as a helper function in the clean function.

(1.2)TextDataSet: It is a subclass that only deal with Text dataset and it is inherited from the base class DataSet. It provided a clean function for cleaning the Text dataset, an explore function for generating two graphs of the Text dataset, and the all\_words function will be a helper function for cleaning the dataset.

(1.3)QuantDataSet: It is a subclass that only deals with a Quantitative dataset and it is inherited from the base class DataSet. It provides a new readCSV function, a new clean function, a new explore function for generating two graphs about the quantitative dataset, self\_mean function will be a helper function for the clean function, it will calculate the mean of the list of floats.

(1.4)QualDataSet: It is a subclass that only deals with a Qualitative dataset and it is inherited from the base class DataSet. It provides a new readCSV function, a new clean function, a new explore function for generating two graphs about the qualitative dataset, and a mode function will be a helper function for the clean function, it will calculate the mode of the list of strings.

(1.5)TransactionDataSet: It is a subclass only deals with the Transaction dataset, it will have a new readCSV function, a new clean function, a transaction\_matrix function to count the item vs people, an item\_list function that counts all items that appear in this dataset, and itemset function which select all combinations that support > 0.25, a support function, a confident function, a lift function to calculate corresponding support, confident and lift, and a explore function will show all the relationships that support > 0.25.

(1.6)HeterogeneousDataSet: It is a subclass that deals with several different types of datasets. A new readCSV function that read several different types of datasets, a new clean function and a new explore function that calls the datasets corresponding clean and explore function, and a select function that only select a specific dataset.

(2)ClassifierAlgorithm: It is a base class that will be inherited by simpleKNNClassifier and kdTreeKNNclassifier. It provides a training method for getting the training data and true label and a test method to train the data.

(2.1)simplekNNClassifier: It is a subclass that is inherited from the ClassifierAlgorithm, the training method will get the training data and true labels, and a new test method that gives k = int, it will get the nearest k neighbors to predict the true labels.

(2.2)kdTreeKNNclassifier: It is a subclass inherited from the ClassifierAlgorithm, the training method will get the training data and build a tree structure and a new test method will use the k = int, get the nearest k neighbors as a tree structure to predict the true labels.

(3)Experiment: It is a base class that has no subclass. It will use the different classifiers to present CrossValidation, accuracy, confusion matrix, and a ROC graph.

(4)Rule: A rule class that was used to express the support, confidence, and lift in an organized way. It will work with the Transaction DataSet as a support class.

(5)Tree: A Tree class that was used to express the kdTree structure.

Simple KNN classifier

Summary, discussion and explanation:

In the train method, we get the training data and actual label. In the test method, for each data(each row), I will calculate the Euclidean distance, for other rows with this row. For example, Euclidean distance in row 1 and row2, row 1 and row 3… Finally, we will get a list of Euclidean distances sorted from smallest to largest. Since k is the input we have, we can get the smallest k Euclidean distances. Then, I check all these k labels, get the mode of the labels and it will be the predicted label.

Code:

class simplekNNClassifier(ClassifierAlgorithm):  
 *"""It is a subclass called simpleKNNClassifier  
 and is inherited from the base class ClassifierAlgorithm.  
  
 It will be dealing with simple KNN specifically.  
 """* def \_\_init\_\_(self):  
 *"""This is the constructor for simplekNNClassifier,  
 it is inherited from the base class ClassifierAlgorithm.  
 """* super().\_\_init\_\_()  
 # print("The simple kNN Classifier Constructor is inherit.")  
  
 def train(self, trainingData, true\_label):  
 *"""This is a train function for simple KNN.  
 It will override the train function from the base class ClassifierAlgorithm.  
 trainingData = DataFrame  
 true\_label = List  
  
 return: None  
 """* # print("The simple kNN Classifier Train is build")  
 self.trainingData = trainingData  
 self.true\_label = true\_label  
  
 def test(self, testData, k=5):  
 *"""This is a test function for simple KNN.  
 It will override the test function from the base class ClassifierAlgorithm.  
 It will do the simple KNN classifier for each row of testData, and get the  
 predicted label to return  
 testData: DataFrame  
 k: num  
  
 return: List[str]  
 T(n): O(n^2)  
 S(n): O(n)  
 T(n) is for the data set, I will calculate the Euclidian Distance for each row of train set, compare the  
 results for test data sets rows, one n for test length, one n for train set length  
 S(n) is for the data set, I will save the labels, it will be O(n)  
 """* # print("The simple kNN Classifier Test is build") # step:1 space:1  
 self.testData = testData # step:1 space:n  
 self.k = k # step:1 space:1  
 predicted\_test\_label = [] # step:1 space:1  
 # do the Euclidean Distance for each row of testData  
 for i in range(len(testData)): # step:n space:n  
 ED = [] # step:n space:n  
 labels = [] # step:n space:n  
 # predicted the labels based on all trainingData  
 for j in range(len(self.trainingData)): # step:n^2 space:n  
 total\_distance = (  
 (self.trainingData.iloc[j][0] - testData.iloc[i][0]) \*\* 2  
 + (self.trainingData.iloc[j][1] - testData.iloc[i][1]) \*\* 2  
 + (self.trainingData.iloc[j][2] - testData.iloc[i][2]) \*\* 2  
 + (self.trainingData.iloc[j][3] - testData.iloc[i][3]) \*\* 2  
 ) # step:12n^2 space:1  
 total\_distance = math.sqrt(total\_distance) # step:n^2 space:1  
 ED.append(total\_distance) # step:n^2 space:0  
 # get the mode of the closest k labels  
 for l in range(k): # step:n^2 space:n  
 labels.append(  
 self.true\_label.iloc[ED.index(min(ED))]  
 ) # step:4n^2 space:0  
 ED.pop(ED.index(min(ED))) # step:2n^2 space:-1  
 # save all the test data labels  
 predicted\_test\_label.append(mode(labels)) # step:n space:n  
 return predicted\_test\_label # step:1 space:0

Time complex analysis:

T(n) = O(n^2) and S(n) = O(n)

The more specific time complex analysis is beside each line of code.

ROC method:

Summary, discussion and explanation:

ROC method contains two parts, one is getting the data and calculating each point in the ROC line, which is in the Test – ExperimentTests(), and the other part is drawing the graph, using the data we get from previously, in the Experiment - ROC method. For the first, part, getting the data for graphing, getting the accuracy from the confusion matrix method in the experiment. I have k = 10 for 10 points in the ROC graph. It can be changed. For each point, three kinds of penguins, each has tpr and fpr, I have six groups of values, which will be mapped to three lines in the ROC graph. Each graph will have k( = 10) points.

Code:

In Test – ExperimentTests():

tpr\_a = [] # step:1 space:n  
fpr\_a = [] # step:1 space:n  
tpr\_c = [] # step:1 space:n  
fpr\_c = [] # step:1 space:n  
tpr\_g = [] # step:1 space:n  
fpr\_g = [] # step:1 space:n  
# calculate each of these points  
for k in range(1, 10):  
 experiment.confusionMatrix(k) # step:O(n^3) \* n space:O(n^2) \* n from Experiment.confusion\_matrix  
 tpr\_a.append(experiment.tpr\_a) # step: 21n space: 3n  
 fpr\_a.append(experiment.fpr\_a) # step: 33n space: 3n  
 tpr\_c.append(experiment.tpr\_c) # step: 33n space: 3n  
 fpr\_c.append(experiment.fpr\_c) # step: 33n space: 3n  
 tpr\_g.append(experiment.tpr\_g) # step: 33n space: 3n  
 fpr\_g.append(experiment.fpr\_g) # step: 33n space: 3n  
# saving all points into a list  
tpr = [tpr\_a, tpr\_c, tpr\_g] # step: 3 space: 0  
fpr = [fpr\_a, fpr\_c, fpr\_g] # step: 3 space: 0  
# get the start points and end points  
for i in tpr: # step: n space: n  
 i.append(0) # step: n space: n  
 i.append(1) # step: n space: n  
 i.sort() # step: nlogn space: 0  
for j in fpr: # step: n space: n  
 j.append(0) # step: n space: n  
 j.append(1) # step: n space: n  
 j.sort() # step: nlogn space: 0  
# plot the curve  
experiment.ROC(tpr, fpr)  
print("\n\n\n")

In Experiment – ROC:

def ROC(self, tpr, fpr):  
 *"""Plot the ROC graph based on true positive rate and false positive rate  
  
 tpr: List[List[float]]  
 fpr: List[List[float]]  
 return: None  
  
 step:55 + O(n^4) (from test) + 6n + 21(from ROC)  
 space:6 (from confusion matrix) + O(n^3) (from test) + n^2 (from ROC)  
 to conclude: T(n): O(n^4) S(n): O(n^3)  
 """* plt.figure() # step:1 space:n^n  
 plt.plot(fpr[0], tpr[0], color="b", label="Adelie Penguin") # step:2n space:0  
 plt.plot(fpr[1], tpr[1], color="g", label="Gentoo Penguin") # step:2n space:0  
 plt.plot(fpr[2], tpr[2], color="r", label="Chiinstrap Penguin") # step:2n space:0  
 plt.plot([0, 1], [0, 1], color="navy", linestyle="--", label="ROC 0.5 line") # step:7 space:0  
 plt.xlim([0.0, 1.0]) # step:1 space:0  
 plt.ylim([0.0, 1.0]) # step:7 space:0  
 plt.xlabel("False Positive Rate") # step:1 space:0  
 plt.ylabel("True Positive Rate") # step:1 space:0  
 plt.title("ROC graph") # step:1 space:0  
 plt.legend(loc="lower right") # step:1 space:0  
 plt.show() # step:1 space:0

Time complex analysis:

T(n) = 55 + O(n^4) (from test) + 6n + 21(from ROC)

S(n) = 6(from confusion matrix) + O(n^3) (from test) + n^2 (from ROC)

To conclude: T(n) = O(n^4) S(n) = O(n^3)

More specific is beside each line of codes.

Apriori Algorithm (Transaction Data):

Summary, discussion and explanation:

readFromCSV method is read csv file.

transaction\_matrix method transfers the original CSV into a count vectorizer data frame. Each column is a product, if 1 appears, the customer gets the item, 0 for otherwise.

Item\_list method is the main part of the apriori algorithm, firstly, it will have all the items in a candidate set C1, check the support value, and we will get an itemset L1. Then, give the itemset L1. We will use conditional joining to form C2. And check for the minimum support value to get an L2. Continue doing this, until all itemset fails in minimum support. Connect the L1, L2, and L3 to form an itemset that will be used in the next function.

Itemset method is using the previous function item\_lists, to circular calculate the FP-growth algorithm, it will have all possible categories that have high support and might fit the constrain supportThreshold > 0.25.

clean method is called the transaction\_matrix method and itemset method.

Support, confident, lift methods are calculate corresponding support, confident, and lift

Explore method is using the Rule Class, given the supportThreshold, to make the transaction items' support, confidence and lift, in this format: ['A'] ==> ['B'] | Sup =

| Conf = | Lift = . Make it in better look

Code:

class TransactionDataSet(DataSet):  
 *"""It is a subclass called TransactionData  
 and is inherited from the base class DataSet.  
  
 It will be dealing with Transaction data set specifically.  
 """* def \_\_init\_\_(self, filename):  
 *"""This is the constructor for Transaction Dataset,  
 it is inherited from the base class DataSet.  
  
 original\_item\_set: List  
 final\_item\_set: List  
 """* super().\_\_init\_\_(filename)  
 # saving the first item possible items  
 self.original\_item\_set = []  
 # saving all possible items  
 self.final\_item\_set = list()  
 print("In Transaction DataSet, the constructor is inherit.")  
  
 def readFromCSV(self, filename):  
 *"""The private readFromCSV function, it will read the CSV to be further process  
  
 filename: string  
 return: None  
 """* print("In Transaction DataSet, the readFromCSV is rebuild.")  
 # nrows can be changed, the data is too large to run. If use all data,  
 # the support will be very low  
 self.data = pd.read\_csv(filename, nrows=100)  
  
 def transaction\_matrix(self):  
 *"""Transfer the original csv into a countvectorizer dataframe by own written code.  
 Each column is a product, if 1 appears, the customer get the item, 0 for otherwise  
  
 return: None  
 """* # count how many different items appear in the dataframe  
 item\_set = set()  
 for index, row in self.data.iterrows():  
 item\_set.update(row)  
 item\_set = {x for x in item\_set if pd.notna(x)}  
 new\_columns = len(item\_set)  
 new\_rows = self.data.shape[0]  
  
 # build a countvectorizer dataframe by own code, 1 for appear, 0 for not  
 self.transaction\_matrix = pd.DataFrame(  
 np.zeros([new\_rows, new\_columns]), columns=item\_set  
 )  
 for index, row in self.data.iterrows():  
 for item in row:  
 if item == np.nan:  
 continue  
 self.transaction\_matrix.loc[index][item] = 1  
  
 def item\_lists(self, prev\_item\_set):  
 *"""The apriori method is implemented, prev\_item\_set is the previous item set, if previous set has  
 one item per list, it will return two item per list for next level  
  
 prev\_item\_set: List[List]  
 return: List[List]  
 T(n) = 3 + 3n + 11n^2 + 2n^2\*logn + 7n^3 = O(n^3)  
 S(n) = 7n + 2n^2 = O(n^2)  
 """* # saving new produced items  
 new\_item\_set = list() # step:1 space:n  
 # two situation appears, one if the prev\_item\_set is List[string](first circulation),  
 # the other situation is prev\_item\_set is List[List] (two or more circulation)  
 for i in prev\_item\_set: # step:n space:n  
 if isinstance(i, str): # step:n space:3n  
 # built the new item set  
 rows = self.transaction\_matrix[self.transaction\_matrix[i] == 1] # step:n\* (n^2) space:n^2  
 rows = rows.sum().sort\_values(ascending=False).reset\_index() # step:n^2\*logn space:0  
 rows.rename(  
 columns={rows.columns[0]: "item\_name", rows.columns[1]: "count"}, # step:n\*n space:0  
 inplace=True,  
 )  
 rows = rows[rows["count"] >= 5] # step:n\*(n^2) space:0  
 rows = rows[rows["item\_name"] != i] # step:n\*(n^2) space:0  
 items = rows["item\_name"].tolist() # step:n\*(n^2) space:n  
 for j in items: # step:n\*n space:0  
 if [j, i] in new\_item\_set: # step:n\*n space:0  
 continue  
 new\_item\_set.append([i, j]) # step:n\*n space:0  
 else:  
 for k in i: # step:n space:n  
 # check each item in the matrix to be 1 and built the new item set  
 rows = self.transaction\_matrix[self.transaction\_matrix[k] == 1] # step:n\*(n^2) space:n^2  
 rows = rows.sum().sort\_values(ascending=False).reset\_index() # step:n^2logn space:0  
 rows.rename(  
 columns={rows.columns[0]: "item\_name", rows.columns[1]: "count"}, # step:n\*n space:0  
 inplace=True,  
 )  
 rows = rows[rows["count"] >= 5] # step:n\*(n^2) space:0  
 items = rows["item\_name"].tolist() # step:n\*(n^2) space:n  
 for j in items: # step:n\*n space:0  
 if (j in i) or [j, i] in new\_item\_set: # step:2n\*n space:0  
 continue  
 lst = copy.deepcopy(i) # step:n\*n space:n  
 lst.append(j) # step:n\*n space:0  
 new\_item\_set.append(lst) # step:n\*n space:0  
 return new\_item\_set # step:2 space:0  
  
 def itemset(self):  
 *"""Using the previous function item\_lists, to circular calculate the FP-growth algorithm  
 final\_item\_set will all possible categories that has high support and might fit the supportThreshold > 0.25  
 condition  
  
 return: None  
 T(n) = 1 + 5n + n^2 + nlogn + [2 \* O(n^3) from item\_list function]  
 S(n) = n + n^2 + [2 \* O(n^2) from item\_list function]  
 """* # filter out the items that could be supporthreshold > 0.25  
 item\_count = (  
 self.transaction\_matrix.sum().sort\_values(ascending=False).reset\_index() # step:nlogn space:n  
 )  
 item\_count.rename( # step:n space:0  
 columns={  
 item\_count.columns[0]: "item\_name",  
 item\_count.columns[1]: "count",  
 },  
 inplace=True,  
 )  
 item\_count = pd.DataFrame(data=item\_count) # step:n^2 space:n^2  
 # first time filter, for one item  
 self.original\_item\_set.extend( # step:1 space:n  
 item\_count[item\_count["count"] > 10]["item\_name"].tolist() # step:n space:0  
 )  
 # second time filter, for two items together  
 # step:O(n^3) from item\_list function space:O(n^2) from item\_list function  
 new\_set = self.item\_lists(self.original\_item\_set)  
 # third time filter, for three items together, where it will reach the minimum support I constructed  
 # fourth time could be constructed, but it's too low  
 # step:O(n^3) from item\_list function space:O(n^2) from item\_list function  
 new\_set2 = self.item\_lists(new\_set)  
 for i in self.original\_item\_set: # step:n space:0  
 self.final\_item\_set.append([i])  
 # combine all three times items together  
 self.final\_item\_set.extend(new\_set) # step:n space:0  
 self.final\_item\_set.extend(new\_set2) # step:n space:0  
  
 def clean(self):  
 *"""Clean method for Transaction data, overwrite the function in base class  
 Call transaction\_matrix function to form a matrix  
 Call itemset function to form a final itemset  
  
 return: None  
 To conclude: the Apriori algorithm:  
 T(n) = O(n^3)  
 S(n) = O(n^2)  
 """* # form the transaction matrix  
 self.transaction\_matrix()  
 # built the itemset  
 self.itemset()  
  
 def support(self, left, right):  
 *"""Support: Transaction contain both X and Y / Total number of transaction  
 get the left item appearances and right item appearances to calculate the support  
  
 left: string/List  
 right: string/List  
 return: float  
 """* # left and right can be a string or a list  
 if isinstance(left, str):  
 numerator = self.transaction\_matrix[self.transaction\_matrix[left] == 1]  
 else:  
 for i in left:  
 numerator = self.transaction\_matrix[self.transaction\_matrix[i] == 1]  
 if isinstance(right, str):  
 numerator = self.transaction\_matrix[self.transaction\_matrix[right] == 1]  
 else:  
 for j in right:  
 numerator = self.transaction\_matrix[self.transaction\_matrix[j] == 1]  
 # calculate the support based on counts  
 denominator = self.transaction\_matrix.shape[0]  
 numerator = numerator.shape[0]  
 supp = numerator / denominator  
 return supp  
  
 def confidence(self, left, right):  
 *"""Confidence: Transaction contain both X and Y / Transaction contain X  
 get the left item appearances and right item appearances to calculate the confidence  
  
 left: string/List  
 right: string/List  
 return: float  
 """* # left and right can be a string or a list  
 if isinstance(left, str):  
 numerator = self.transaction\_matrix[self.transaction\_matrix[left] == 1]  
 else:  
 for i in left:  
 numerator = self.transaction\_matrix[self.transaction\_matrix[i] == 1]  
 denominator = numerator.shape[0]  
 if isinstance(right, str):  
 numerator = self.transaction\_matrix[self.transaction\_matrix[right] == 1]  
 else:  
 for j in right:  
 numerator = self.transaction\_matrix[self.transaction\_matrix[j] == 1]  
 # calculate the confidence based on formula  
 numerator = numerator.shape[0]  
 conf = numerator / denominator  
 return conf  
  
 def lift(self, left, right):  
 *"""Lift: Transaction contains both X and Y / ((Transaction contains X) \* (Transaction contains Y))  
 get the left item appearances and right item appearances to calculate the lift  
  
 left: string/List  
 right: string/List  
 return: float  
 """* # left and right can be string or list  
 if isinstance(left, str):  
 numerator = self.transaction\_matrix[self.transaction\_matrix[left] == 1]  
 else:  
 for i in left:  
 numerator = self.transaction\_matrix[self.transaction\_matrix[i] == 1]  
 if isinstance(right, str):  
 numerator = self.transaction\_matrix[self.transaction\_matrix[right] == 1]  
 else:  
 for j in right:  
 numerator = self.transaction\_matrix[self.transaction\_matrix[j] == 1]  
 numerator = numerator.shape[0]  
  
 if isinstance(left, str):  
 denominator\_x = self.transaction\_matrix[self.transaction\_matrix[left] == 1]  
 else:  
 for i in left:  
 denominator\_x = self.transaction\_matrix[self.transaction\_matrix[i] == 1]  
 denominator\_x = denominator\_x.shape[0]  
 if isinstance(right, str):  
 denominator\_y = self.transaction\_matrix[self.transaction\_matrix[right] == 1]  
 else:  
 for j in right:  
 denominator\_y = self.transaction\_matrix[self.transaction\_matrix[j] == 1]  
 # calculate the lift based on formula  
 denominator\_y = denominator\_y.shape[0]  
 lift = numerator / (denominator\_x \* denominator\_y)  
 return lift  
  
 def explore(self, supportThreshold=0.25):  
 *"""given the supportThreshold, to make the transaction items' support, confidence and lift  
 in this format: ['A'] ==> ['B'] | Sup = | Conf = | Lift =  
  
 return: None  
 """* # saving all the rules  
 rules = []  
 for i in self.final\_item\_set:  
 for j in self.final\_item\_set:  
 state = True  
 if i != j and len(i) == 1 and len(j) == 1:  
 rules.append(  
 Rule(  
 i,  
 j,  
 self.support(i, j),  
 self.confidence(i, j),  
 self.lift(i, j),  
 )  
 )  
 if len(j) - len(i) == 1:  
 for k in i:  
 if k not in j:  
 state = False  
 break  
 if state == True:  
 rules.append(  
 Rule(  
 i,  
 j,  
 self.support(i, j),  
 self.confidence(i, j),  
 self.lift(i, j),  
 )  
 )  
 # sort the rules based on support value  
 for i in range(1, len(rules)):  
 for j in range(i - 1):  
 if rules[j].support < rules[i].support:  
 rules[j], rules[i] = rules[i], rules[j]  
 # check if the support is larger than the supportThreshold, then print out  
 for k in range(10):  
 if rules[k].support >= supportThreshold:  
 print(rules[k])  
  
  
class Rule:  
 *"""It is a Base class for saving the data of left, right, support, confidence and lift  
 It's a clear way to show the data  
 """* def \_\_init\_\_(self, left, right, support, confidence, lift):  
 *"""The Base class Rule constructor  
  
 left: string  
 right: string  
 support: float  
 confidence: float  
 lift: float  
 return: None  
 """* self.left = left  
 self.right = right  
 self.support = support  
 self.confidence = confidence  
 self.lift = lift  
  
 def \_\_str\_\_(self):  
 *"""The output format of Rule  
  
 return: string  
 """* return f"{self.left} ==> {self.right} | Sup = {self.support} | Conf = {self.confidence} | Lift = {self.lift}"

Time complex analysis:

It conclude in two functions, item\_list (main algorithm):

T(n) = 3 + 3n + 11n^2 + 2n^2\*logn + 7n^3 = O(n^3)

S(n) = 7n + 2n^2 = O(n^2)

Itemset(support algorithm):

T(n) = 1 + 5n + n^2 + nlogn + [2 \* O(n^3) from item\_list function] = O(n^3)

S(n) = n + n^2 + [2 \* O(n^2) from item\_list function] = O(n^2)

More specific details will be inside the note of item\_list and itemset functions.

KDTreeKNNclassifier:

Summary, discussion and explanation:

Build\_tree method will transfer the data frame into a tree structure based on the helper class Tree()

Train method will get the data frame and the true label for further use. Build\_tree method will be called to form a tree prepared for the test method

Closest method will get the linear distance from the current node’s value to the other two subtree value

Nearest\_neighbour will find the nearest neighbor of the node we want

Test method will get all the test data and nearest k and provided all the true labels for each node for nearest k neighbors

Code:

class kdTreeKNNClassifier(ClassifierAlgorithm):  
 *"""It is a subclass called kdTreeKNNClassifier  
 and is inherited from the base class ClassifierAlgorithm.  
  
 It will be dealing with kd Tree KNN specifically.  
 """* def \_\_init\_\_(self):  
 *"""This is the constructor for kdTreeKNNClassifier,  
 it is inherited from the base class ClassifierAlgorithm.  
 """* super().\_\_init\_\_()  
 print("The kd Tree Classifier Constructor is inherit.")  
 self.num\_attributes = 0  
 self.trainingData = None  
 self.train\_true\_label = None  
 self.testData = None  
 self.k = 5  
 self.kdtree = None  
  
 def build\_tree(self, lst, label, depth):  
 *"""Built tree based on lst, label, and depth and the class Tree()  
  
 lst: list  
 label: list  
 depth: int  
 return: Tree  
 T(n): 13 + 3n + nlogn + 2\*T(n/2) -> recursion structure  
 T(n) = O(n^2)  
 S(n): 8 + 4n + 2\*S(n/2) -> recursion structure  
 S(n) = O(nlogn)  
 """* tree = Tree() # step:1 space:1  
 tree.depth = depth # step:1 space:1  
 tree.axis = tree.depth % self.num\_attributes # step:3 space:1  
 if len(lst) == 0: # step:3 space:1  
 return None # step:1 space:1  
 values = [x[tree.axis] for x in lst] # step:n space:n  
 index = np.argsort(values) # step:nlogn space:n  
 vec\_list = lst[index] # step:n space:n  
 lbls = label[index] # step:n space:n  
 tree.location = len(vec\_list) // 2 # step:2 space:1  
 tree.value = vec\_list[tree.location] # step:1 space:1  
 tree.label = lbls[tree.location] # step:1 space:1  
 tree.leftChild = self.build\_tree(  
 vec\_list[0 : tree.location], lbls[0 : tree.location], depth + 1  
 ) # step:T(n/2) space:S(n/2)  
 tree.rightChild = self.build\_tree(  
 vec\_list[tree.location + 1 :], lbls[tree.location + 1 :], depth + 1  
 ) # step:T(n/2) space:S(n/2)  
 return tree  
  
 def train(self, trainingData, train\_true\_label):  
 *"""This is a train function for kd Tree KNN.  
 It will override the train function from the base class ClassifierAlgorithm.  
  
 return: None  
 T(n) = 1 + 2n + 2n^2 + O(n^2) (from build\_tree function)  
 T(n) = O(n^2)  
 S(n) = 1 + n + n^2 + O(nlogn) (from build\_tree function)  
 S(n) = O(n^2)  
 Compare to simple KNN, the time complexity looks same, they are both O(n^2), but  
 simple KNN will have a more constant time complexity. The Kd-tree time complexity might  
 fluctuate a lot, it might be higher than simple KNN, if the tree didn't split the data  
 well. But it also might be saving time than KNN.  
 For space complexity, the Kd-tree method will be higher than simpleKNN for sure, because  
 we need extra space for saving tree structure.  
 """* print("The kd Tree Classifier Train is build")  
 self.trainingData = trainingData # step:n^2 space:n^2  
 self.train\_true\_label = train\_true\_label # step:n space:n  
 self.num\_attributes = self.trainingData.shape[1] # step:1 space:1  
 self.trainingData = np.array(  
 [  
 self.trainingData.iloc[i, :].to\_numpy()  
 for i in range(self.trainingData.shape[0])  
 ]  
 ) # step:n^2 space:0  
 self.train\_true\_label = self.train\_true\_label.to\_numpy() # step:n space:0  
 self.kdtree = self.build\_tree(  
 self.trainingData, self.train\_true\_label, 0  
 ) # step:O(n^2) space:O(nlogn) by build\_tree function  
  
 def closest(self, value, tree1, tree2):  
 *"""Check which children has smaller distance  
  
 value: float  
 tree1: tree  
 tree2: tree  
 return: tree, float  
 T(n) = 6  
 T(n) = O(c) -> constant  
 S(n) = 5  
 S(n) = O(c) -> constant  
 """* dist\_to\_rt1 = np.linalg.norm(value - tree1.value) # step:1 space:1  
 dist\_to\_rt2 = np.linalg.norm(value - tree2.value) # step:1 space:1  
 if dist\_to\_rt1 < dist\_to\_rt2: # step:3 space:1  
 return tree1, dist\_to\_rt1 # step:2 space:0  
 else:  
 return tree2, dist\_to\_rt2 # step:2 space:2  
  
 def nearest\_neighbor(self, tree, lst):  
 *"""check the nearest neightbor of the tree  
  
 tree: tree  
 lst: list  
 return: tree  
 T(n) = 28 + 2\*O(c)  
 T(n) = O(c)  
 S(n) = 4 + 2\*O(c)  
 S(n) = O(c)  
 """* if tree is None: # step:2 space:1  
 return None # step:2 space:0  
 if lst[tree.axis] < tree.value[tree.axis]: # step:3 space:1  
 next\_branch = tree.leftChild # step:1 space:1  
 other\_branch = tree.rightChild # step:1 space:1  
 else:  
 next\_branch = tree.rightChild # step:1 space:0  
 other\_branch = tree.leftChild # step:1 space:0  
 temp = self.nearest\_neighbor(next\_branch, lst) # step:T(c) space:S(c)  
 if temp is not None: # step:2 space:1  
 best, dist = self.closest(lst, tree, temp) # step:T(c) space:S(c)  
 else:  
 best = tree # step:1 space:0  
 dist = np.linalg.norm(lst - tree.value) # step:2 space:0  
 dist\_prime = abs(lst[tree.axis] - tree.value[tree.axis]) # step:4 space:1  
 if dist\_prime < dist: # step:3 space:1  
 temp = self.nearest\_neighbor(other\_branch, lst) # step:T(c) space:S(c)  
 if temp is not None: # step:2 space:1  
 best, temp\_dist = self.closest(  
 lst, best, temp  
 ) # step:T(c) space:S(c)  
 else:  
 best = tree # step:2 space:0  
 return best # step:2 space:0  
  
 def test(self, testData, k=5):  
 *"""This is a test function for kd Tree KNN.  
 It will override the train function from the base class ClassifierAlgorithm.  
  
 return: list[string]  
 T(n) = 5 + 2n + 4n^2  
 T(n) = O(n^2)  
 S(n) = 3 + 5n + n^2  
 S(n) = O(n^2)  
 Compare to simple KNN, the time complexity looks same, they are both O(n^2), but  
 simple KNN will have a more constant time complexity. The Kd-tree time complexity might  
 fluctuate a lot, it might be higher than simple KNN, if the tree didn't split the data  
 well. But it also might be saving time than KNN.  
 For space complexity, the Kd-tree method will be higher than simpleKNN for sure, because  
 we need extra space for saving tree structure.  
 """* print("The kd Tree Classifier Test is build")  
 if k is not None: # step:2 space:1  
 self.k = k # step:1 space:1  
 self.testData = testData # step:n^2 space:n^2  
 test\_shape = self.testData.shape # step:1 space:1  
 test\_result = [None] \* test\_shape[0] # step:n space:n  
 for i in range(test\_shape[0]): # step:n space:n  
 test\_vec = self.testData.iloc[i, :].to\_numpy() # step:n^2 space:n  
 neighbor = self.nearest\_neighbor(  
 self.kdtree, test\_vec  
 ) # step:n^2 space:n  
 test\_result[i] = neighbor.label # step:n^2 space:n  
 return test\_result # step:2 space:0

Time complex analysis:

It conclude in two functions, train():

T(n) = 1 + 2n + 2n^2) + O(n^2)(from build\_tree function) = O(n^2)

S(n) = 1 + n + n^2 + O(nlogn)(from build\_tree function) = O(n^2)

test():

T(n) = 5 + 2n + 4n^2 = O(n^2)

S(n) = 3 + 5n + n^2 = O(n^2)

More specific details will be beside the codes.