CS-IFS: new model for classification problem

Nguyễn Quốc Anh, Vũ Việt Anh,
Vũ Hữu Nam Anh, Đỗ Việt Cường, Đỗ Hoàng Long
HUST
SOICT

Abstract

Classification is one of the most common problems in supervised learning of Machine Learning. There are many methods to handle and achieve high accuracy with this problem. In this report, we will present a new method based on the knowledge of fuzzy sets that performs the probabilities of some fuzzy objects with high confidence. We will talk a little bit about fuzzy sets, basic concepts and some laws of fuzzy sets. Then, we will present bit by bit the algorithm implementation of the CS-IFS model. To represent the experimental results of the proposed algorithm, we run on multiple data sets, which are random data sets to show the algorithm's compatibility with other known data sets. Through this, we discovered some properties of the algorithm and the data sets that are compatible with it, and we also thought about some interesting things that could help improve the algorithm's performance in future.

Keyword: Fuzzy set

I. Introduction

In the current era, the rapid development of information technology (IT) has led to the development of many other fields. It can be said that IT is changing the shape of the world economy, helping humanity to take the first firm steps on the path of knowledge economy, e-commerce... However, the remarkable development of IT has significantly increased the number of information transactions on the Internet, especially electronic mail, electronic news, etc. According to statistics of Broder et al (2008), about every 6 to 10 months that amount of information doubles, besides the speed of information change is also extremely fast. A serious request for us is how to organize and search for information in the most effective way and classify information as one of the reasonable solutions for this request. However, given the volume of data and the urgency of the situation,

manual categorization is impossible. As a result, the need to create tools that could categorize, anticipate, and group items into classes automatically arose, which is also where the classification problem originated.

Classification problem is the process of classifying a data object into one or more certain classes using a classification model. This model is built based on the previously built data set with labels (also known as training set), the main process of classification is the process of labeling data objects. This problem is one of the most common in Machine Learning and has many practical applications. It is easy to cite a few examples of it such as customer segmentation, prospecting or forecasting the risk of employee turnover. The classification problem exists in both supervised learning, unsupervised learning, and reinforcement learning. Up to now, there have been many highly effective classification models for each suitable dataset such as: Decision Trees, Support Vector Machines, K nearest neighbors, ... However, through research, we find that there are not many models to solve the problem based on fuzzy elements (also called fuzzy sets).

In this report, we provide a method that draws on fuzzy set theory and fuzzy logic. The algorithm's goal is to obfuscate the object's attribute data in order to calculate the distance between each item and the class (cluster) center and, in turn, determine the labels that should be applied to each object. We give an alternative solution to the problem in Part II of the study based on the problem and suggested solutions in Part I. Some of the terms and concepts used to model the issue are presented in Part II. The suggested method, how to fuzzy data, how to figure out where each layer's center is, and how to figure out how far an item is from the layer's center are all covered in Part III. We offer some experimental findings from various sizable data sets in Part IV. In the discussion, we will comment on the advantages and disadvantages of the model and give some ideas for future development of the model.

II. Fuzzy set

1. Concept of fuzzy set

Set: a collection of elements which have the same property. (Ex: the collection of student at 12^{th} grade, the collection of operation, …)

However, in practice these concepts cannot be defined unambiguously. For example, let's say a student has a GPA, but that student may have a 6,5 or 7,9 without us knowing the specific score. Or we fail to determine the specific age of the term middle age. From the above examples it can be seen that it is not possible to define as unambiguously as in the usual concepts of sets. Such cases

Given **U** *is a reference universe, the fuzzy subset* **A** *on* **U** *is determined by the membership function* μ_A*, assigning to each element u of* **U**, *a value* $\mu_A(u)$, *with* $0 \leq \mu_A(u) \leq 1$, *to indicate the degree to which element u belongs to the fuzzy set* **A**. *In other words, the fuzzy subset* **A** *on* **U** *is determined by the mapping:*

$$\mu_A: \mathbf{U} \rightarrow [0,1]$$

are often called fuzzy sets, the elements of which have no criteria for "belonging". So, we can define the term fuzzy set as below:

2. Operations on the fuzzy sets

2.1. Compare fuzzy sets:

Let A and B be two fuzzy subsets on the reference universe U with two membership function μ_A and μ_B, then we have:

- Two fuzzy sets A and B are called equal: denoted A = b, if $\forall u \in U$ then $\mu_A(u) = \mu_B(u)$
- Fuzzy set A is contained in fuzzy set B: symbol $A \subseteq B$, if $\forall u \in U$ then $\mu_A(u) \leq \mu_B(u)$

2.2. Operation on the fuzzy set:

- Commutative
- Associative
- Set-like distribution

2.3. The complement of a complement set:

The complement of fuzzy set A, denoted $\hat{A}$ is a fuzzy subset on U with the membership function denoted $\mu_{\bar{A}}(u)$ and define as follows:

$$\forall U \in U, \mu_{\bar{A}}(u) = 1 - \mu_A(u)$$

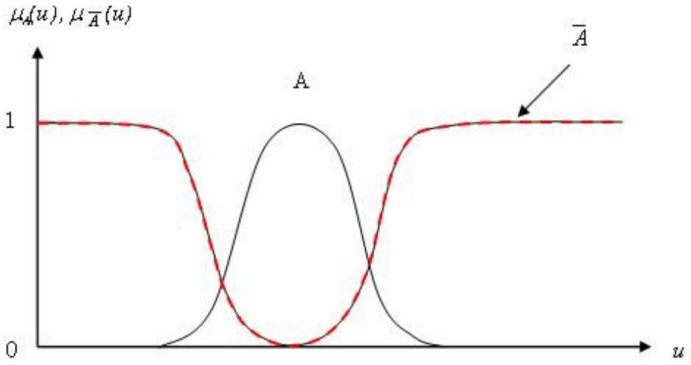


Figure 1: The complement of fuzzy set

3. Properties

- Similar to the normal set, the fuzzy set also has the following properties:
  $A \cap B \subset A, A \subset A \cup B$
  $A \cap X = A, A \cup X = X$
  $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- With A, B is a fuzzy set:
  $A \subset B$ and $B \subset A \leftrightarrow A = B$
- Some properties on the complement:
  $\bar{\bar{A}} = A; \overline{U} = \emptyset; \overline{\emptyset} = U$
- Duality formula:
  $\overline{A \cup B} = \bar{A} \cap \bar{B}$
  $\overline{A \cup B} = \bar{A} \cup \bar{B}$

III. Methodology

1. Input data and visualization

1.1. Input data

General problem: Having m objects {O_1, O_2, …, O_m} which is labeled from L_1 to L_k based on n feature A_1 to A_n. We have the demonstration table as below:

Object	Feature (Inputs)					Class (label)
	A_1	A_2	A_3	….	A_n	
O_1	r_{11}	…	…	…	r_{1n}	L_1
O_2	r_{21}	…	…		r_{2n}	L_2
O_3	r_{31}	…	…	…	r_{3n}	L_1
O_4	r_{41}	…	…		r_{4n}	L_3
O_5	r_{51}	…	…		r_{5n}	L_2
O_6	r_{61}	…	…		r_{6n}	L_k
…	…	…	…	…	…	
O_m	r_{m1}				r_{mn}	L_4

With r_{ij}: real number

1.2. Visualization the features in dataset.

1.2.1. Introduction

Before building the model, we should have insight data. To do that, we have built EDA library that knows basic statistical information about each feature of real numeric data and can represent the distribution of data points for users to provide a suitable solution.

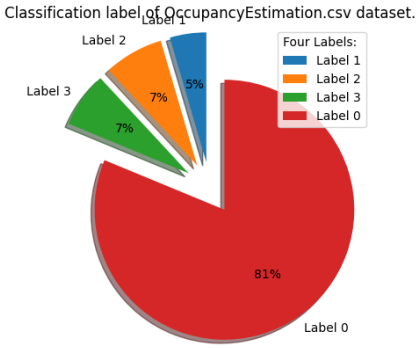
1.2.2. Screening basic statistical information

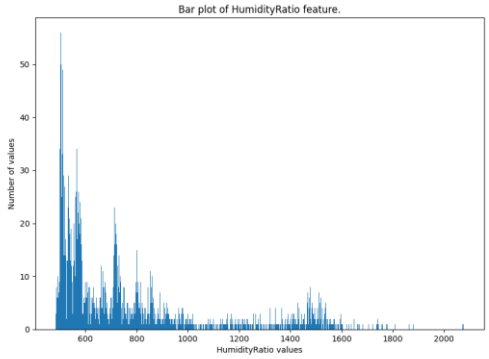
Both features in dataset have some important stats such as mean, standard deviation, ... EDA library will show to user number of unique values, mean value, standard deviation value, min value and max value. With label feature, we screen the number of label in the dataset so that users know if their problem is a binary or multi-label classification problem.

1.2.3. Plot the data distribution

The EDA library provides several graphing functions to give users the most insight into the data. There are 3 main chart types in the library which are column chart, horizontal column chart and pie chart. A type of column chart used to show the distribution of data in columns that are not usually labeled columns. From there, users can completely come up with reasonable solutions to process data such as normalizing, scaling data according to minmax or uniform distributed. The remaining 2 histograms are used to represent the distribution labels in the problem. Pie chart can help users

see data imbalances to come up with a reasonable solution in data division.





2. Splitting of training and testing sets

This class was created to support the CS-IFS model. The input data sets are often in raw form with a format that is not guaranteed to the input requirements of the problem. In addition, the data sets have not been divided into training and test sets, so a class is required to be able to accomplish the above goal. That's why we developed TrainAndTestSplitting class to make the job easier.

2.1. Introduction

In class TrainAndTestSplitting, we will read the input data and then convert the data into a suitable form for CS-IFS model. After we have read the data, we will continue to separate them into two sets, the training set and the testing set. We will ask the user to enter parameters such as: size of training and testing sets, data split type if desired. Regarding the data splitting method, we have built two main ways that are random division or classification division. Each method has its own characteristics and methods and will be discussed in the following sections.

2.2. Reading and writting data

In this section, we will mainly use a few tools of the csv library to read and write csv format files. After reading the csv files, we will save it as a list of data points to serve to divide into 2 sets of training and testing. After we have finished splitting the training and testing set according to the size and manner that the user wants, we just need to write it to 2 files in csv format to complete the work in this part.

2.3. Splitting data

We implement two main splitting data methods such as: random spitting and stratified splitting. Each data set has a unique attribute, which is why we need to develop a variety of data splitting methods. We must find an appropriate splitting to increase the power and enhance the learning capacity of the CS-IFS model.

In random splitting, we only shuffle all data points in data list and then take the number of data point equivalent to the training size of user requesting for the training set. The rest data points will be the testing set.

In stratified sampling, we first need to calculate the proportion of each class in the original data set. Then we have to divide the data into 2 training and test sets to ensure that the proportion of each

class in the training and test sets is the same as in the original data set. The purpose of this method is to produce a uniform distribution both in the original data set and the two sets after being split. This will help the algorithm to ensure that what it has learned in the training set will meet in the same way as in the testing set.

3. Method

In this paper, we focus on predicting the label of data based on the knowledge of the fuzzy set proposed in the previous section. Firstly, we have to fuzzy each data point into a reliable and a surgeno (unreliable) point. Secondly, the center point for each label will be calculated and fuzzyred by the truth label. Thirdly, we have to calculate the weight of each feature then calculate the distance between each data point and center of that point according to our formula to prepare for final step that is when the data is labeled based on the minimum distance with the suitable unit of measurement. We have the pseudo code as below.

CS-IFS ALGORITHM (D[1...(n + 1), 1…m], D[n + 1]: truth label, **criterion**: positive real value)

Step 1: Data fuzzification: change r_{ij} into (y_{ij}, n_{ij})

Step 2: Calculate the center point L_i of each class from data and fuzzy it

Step 3: Calculate the weight of each feature

Step 4: Calculate the distance between each data point to its center point

Step 5: Put each data point to its class based on the minimum distance with the suitable unit of measurement

Step 6: Calculate the accuracy between the number of correctly predicted labels with the total number of data points

3.0. Note for code implementation:

We have several variables with their functions as follows:

```
self.dataValue = None #save the value of all cells in csv table
self.dataHeader = None #save the name of all columns in csv table
self.numberOfHeader = None #save the number of columns in csv file without the label columns
self.numberOfLabel = None  #save the label of cases in csv table
self.label = None #save the name of labels
self.labelValue = None #save the label of all instances in csv table
self.predictLabel = None #save the predict label of all instances in csv table
self.predictLabelT = None #save the predict label of all instances in test set
self.numberOfInstances = None #save the number of instances in csv table
self.numberOfTest = None #save the number of instances in the test set
self.weights = None #save the weights of all features in the cs_ifs model
self.result = None #save the distance between the instances with each center in center list
self.Tresult = None #save the distance between the instances with each center in center list
self.RelTableTrain = None #save the result of IF function in the train set
self.SurgenoTableTrain = None #save the result of NON_IF function in the train set
self.RelTableTest = None #save the result of IF function in the test set
```

```
self.SurgenoTableTest = None #save the result of NON_IF function in the test set
self.RelCenterTable = None #save the result of IF function of center list
self.SurgenoCenterTable = None #save the result of NON_IF function of center list
self.accuracy = 0  #save the accuracy of cs_ifs model with the csv file
self.time = 0 #save total time to updating the weights
self.measure = "Default" #save the measurement to evaluate the distance between 2 points
self.evaluation = "Accuracy" #save the method to evaluate the result of cs-ifs model
self.p = 1 #save the degree of function if user want to evaluate by mincowski
```

3.1. Training phase

3.1.1. Data fuzzification

1. Idea

In the first step, we have to change each feature's point into reliable feature point and surgeno's feature point (or unreliable feature point). To perform this process, we have two main function as below:

+ Reliable function: $y_{ij} = \frac{r_{ij} - \min\limits_{i=1,2,\dots m} r_{ij}}{\max\limits_{i=1,2,\dots m} r_{ij} - \min\limits_{i=1,2,\dots m} r_{ij}}$ (1)

+ Surgeno function (unreliable function): $n_{ij} = \frac{1 - y_{ij}}{1 + y_{ij}}$ (2)

This step simulates as the theory above about the fuzzy set or you can easily understand that is a transformation between a space to two space with the same dimensions. The value of y_{ij} similar to the Minmax scale method because we will transform each r_{ij} value according to the maximum and minimum range of the A_i attribute to get the value of the membership (reliable) function, this is also the step to normalize the input data for our method and ensure the value of data is always non-negative.

2. Code implementation

The source code of the reliable function calculation is shown as below:

```
#Calculation Reliability function
#Reliability function: y[i, j] = (r[i, j] - min r[i, j]) / (max r[i, j] - min r[i, j]) (with j
= 1, 2, ... m) (1)

    def RelFunction(self, columnName, RelTable, isCalCenter = False, centerValue = [[]],
isTrain = True):
        #check the dataset is training or testing set
        if isTrain == True:
            number = self.numberOfInstances
            columnData = self.data[columnName].values.tolist()
        else:
            number = self.numberOfTest
```

```
            columnData = self.test[columnName].values.tolist()
        minValue = min(columnData) #find min value of each feature
        maxValue = max(columnData) #find max value of each feature
        idxColumn = self.dataHeader.index(columnName)
        if isCalCenter == False: #if True, calculate the reliable value of data points
            for idx in range(number):
                RelTable[idx][idxColumn] = (columnData[idx] - minValue) / (maxValue -
minValue)
        else: #calculate the reliable value of center points
            for idxLabel in range(self.numberOfLabel):
                RelTable[idxLabel][idxColumn] = (centerValue[idxLabel][idxColumn] - minValue)
/ (maxValue - minValue)
```

Next, we will go to the code for calculating surgeno function:

```
#Calculation Surgeno fuction
# Surgeno function:     n[i, j] = (1 - y[i, j]) / (1 + y[i, j]) (2)

    def SurgenoFunction(self, columnName, SurgenoTable, RelTable, isCalCenter = False, isTrain
= True):
        idxColumn = self.dataHeader.index(columnName)
        if isCalCenter == False: #check for calculating for center points list or not
            if isTrain == True: #if True, the list is data points in training set
                length = self.numberOfInstances
            else: #the list is data points in testing set
                length = self.numberOfTest
        else: #calculating the center points list
            length = self.numberOfLabel
        for idx in range(length):
            SurgenoTable[idx][idxColumn] = (1 - RelTable[idx][idxColumn]) / (1 +
RelTable[idx][idxColumn])
```

After implementing 2 main methods, we have a method to receive the input from user and control the functionality of those methods.

```
#Create two table using two function below to use for the next steps
    def CreateFuzzyTable(self, isTrain = True):
        if isTrain == True:
            RelTable = [[0 for i in range(self.numberOfHeader)] for j in
range(self.numberOfInstances)]
            SurgenoTable = [[0 for i in range(self.numberOfHeader)] for j in
range(self.numberOfInstances)]
        else:
            RelTable = [[0 for i in range(self.numberOfHeader)] for j in
range(self.numberOfTest)]
            SurgenoTable = [[0 for i in range(self.numberOfHeader)] for j in
range(self.numberOfTest)]
        for columnName in self.dataHeader[:self.numberOfHeader]:
```

```
            self.RelFunction(columnName, RelTable, isTrain = isTrain)
            self.SurgenoFunction(columnName, SurgenoTable, RelTable, isTrain = isTrain)
        return RelTable, SurgenoTable
```

This method will return 2 table respectively: reliable table and surgeno table based on 2 functions with the same name.

3.1.2. Calculating the center points

1. Idea

After step 1, we have to find the center points of labels after fuzzification whole input data. To determine each feature's value of center points, we have to classify all data points into k classes based on its truth label. Then we receive k classes as below:

$$S_{L_p} = \left\{(C_j, r_j^p) \mid j = 1, 2, ..., n\right\} \left(r_j^p = \frac{1}{|L_p|} \sum_{q=1}^{|L_p|} r_{ij} \,,\ p = 1, 2, ..., k \right)$$

Now, it's an important process that we have to fuzzy all center points $S_{L_p} = \left\{\left(C_j, (y_j^{L_p}, n_j^{L_p})\right) \mid j = 1, 2, ..., n\right\}$

by using formula: $y_j^{L_p} = \frac{r_j^p - \min\limits_{i=1,2,...m} r_{ij}}{\max\limits_{i=1,2,...m} r_{ij} - \min\limits_{i=1,2,...m} r_{ij}}$, $n_j^{L_p} = \frac{1 - y_j^{L_p}}{1 + y_j^{L_p}}$, for all p = 1, 2, ..., k

We hope that you will follow the above procedure for best results. If you compute the fuzzy center points directly based on the fuzzy input data points, your model could be very bad.

2. Code implementation

```
def CalClusterCenter(self):
        dictLabel = {}
        dictCenterLabel = {}
        dictElementsLabel = {}
        for label in self.label:
            dictLabel[label] = []
            dictCenterLabel[label] = [0 for i in range(self.numberOfHeader)]
            dictElementsLabel[label] = 0
        for instance in range(self.numberOfInstances):
            for label in self.label:
                if label == self.labelValue[instance]:
                    dictLabel[label].append(instance)
                    dictCenterLabel[label] = [dictCenterLabel[label][idx] \
                        + self.dataValue[instance][idx] for idx in range(self.numberOfHeader)]
                    dictElementsLabel[label] += 1
        RelCenterTable = [[0 for i in range(self.numberOfHeader)] for j in
range(self.numberOfLabel)]
        SurgenoCenterTable = [[0 for i in range(self.numberOfHeader)] for j in
range(self.numberOfLabel)]
```

```
        centerValue = []
        for eachLabel in self.label:
            dictCenterLabel[eachLabel] = [dictCenterLabel[eachLabel][idx] /
dictElementsLabel[eachLabel] \
                for idx in range(self.numberOfHeader)]
            centerValue.append(dictCenterLabel[eachLabel])
        for columnName in self.dataHeader[:self.numberOfHeader]:
            self.RelFunction(columnName = columnName, RelTable = RelCenterTable, isCalCenter =
True, centerValue = centerValue)
            self.SurgenoFunction(columnName = columnName, SurgenoTable = SurgenoCenterTable,
RelTable = RelCenterTable, isCalCenter = True)
        return RelCenterTable, SurgenoCenterTable
```

3.1.3. Calculating the weights

1. Idea

The CS-IFS algorithm requires user input of **criterion**. This term denotes the standard difference between two consecutive weight updates. If the result between the two updates of each attribute is less than **criterion**, then 1 criterion is completed, and when all the attributes meet the requirements, the algorithm will find out optimally weights.

Before calculating the weight list, we have to find T_j (j = 1, 2,, n) which is characteristic value for each attribute A_j (j = 1, 2,..., n) as the formula below:

$$T_j = \frac{\sum_{i=1}^{m} \left| S(y_{ij}, n_{ij}) - S(n_{ij}, y_{ij}) \right|}{m} \quad (3)$$

In formula (4), there is a new term S which is denoted for a nonlinear component. We can calculate S as the formula below:

$$S((y_{ij}, n_{ij})) = \frac{[3 + 2y_{ij} + (y_{ij})^2 - n_{ij} - 2(n_{ij})^2]e^{2y_{ij} - 2n_{ij} - 2}}{6} \quad (4)$$

After having T and S components, we have to initialize the weights $w_j^{(0)} = \frac{1}{n}$ (j = 1, 2,..., n). Then we will update the weight $w_j^{(t+1)} = \frac{w_j^{(t)} T_j}{\sum_{j=1}^{n} w_j^{(t)} T_j}$ (5) (with (t) is time t, j = 1,2,..., n) until the different between two consecutive times of all features are both smaller than **criterion**.

$$w_j^{(t+1)} - w_j^{(t)} \leq \textbf{criterion} \; \forall j = 1, 2, \ldots, n$$

As you can see, S is a nonlinear function. If we only apply a linear function to determine S, it means that you can completely transform the feature value T from the fuzzy data points. So your weight

can be represented in linear form from the original fuzzy data point. Equation (4) produces a nonlinear transformation that can yield many desirable results

After section **3.1.3**, you will receive the optimal weights after (*) updates, whole weights list can be denoted as $w^* = (w_1^{(*)}, w_2^{(*)}, ..., w_n^{(*)})$.

2. Code implementation

```
def CalWeights(self, RelTable, SurgenoTable, criterion = 10 **(-4)):
        #initialize the values for weight list
        self.weights = [1 / self.numberOfHeader for i in range(self.numberOfHeader)]
        passCriterion = 0
        #the progress to update weight values as the funciton above
        while passCriterion < self.numberOfHeader:
            passCriterion = 0
            newWeight = [0 for i in range(self.numberOfHeader)]
            componentT = [self.CalWeightsComponentT(idxColumn, RelTable, SurgenoTable) for
idxColumn in range(self.numberOfHeader)]
            denominatorW = 0
            for idx in range(self.numberOfHeader):
                denominatorW += self.weights[idx] * componentT[idx]
            for idx in range(self.numberOfHeader):
                newWeight[idx] = self.weights[idx] * componentT[idx] / denominatorW
                dif = abs(newWeight[idx] - self.weights[idx])
                if dif < criterion:
                    passCriterion += 1
                self.weights[idx] = newWeight[idx]
            self.time += 1
```

In the code above, we used 2 new methods to calculate S component and T component. Now, we will implement these methods.

```
# Calculation T component with the function below:
    # T[j] = |S(y[i, j], n[i, j]) - S(n[i, j], y[i, j])| / m
    # with: m is the number of instances
    # i = 1, 2, ..., m

    def CalWeightsComponentT(self, idxColumn, RelTable, SurgenoTable):
        numeratorT = 0
        for idx in range(self.numberOfInstances):
            y = RelTable[idx][idxColumn]
            n = SurgenoTable[idx][idxColumn]
            numeratorT += abs(self.CalWeightsComponentS(y, n) - self.CalWeightsComponentS(n,
y))
        componentT = numeratorT / self.numberOfInstances
        return componentT
```

```
    # Calculation S component with the function below:
    # S(y[i, j], n[i, j]) = (3 + 2 * y[i, j] + y[i, j] ^ 2 - n[i, j] - 2 * n[i, j] ^ 2) *
exp(2 * y[i, j] - 2 * n[i, j] - 2) / 6 (5)

    def CalWeightsComponentS(self, y, n):
        componentS = (3 + 2 * y + (y) ** 2 - n - 2 * (n) ** 2) * math.exp(2 * y - 2 * n - 2) /
6
        return componentS
```

3.1.4. Calculating distance

1. Idea

After receiving the optimal weights list, we can calculate the distance between each data point to its center point according to formulas below with p = 1, 2, ..., k and i = 1, 2, ..., m

Default:

$$d\left(S_i, S_{L_p}\right) = \sum_{j=1}^{n} w_j^{(*)} |S_i(y_{ij}, n_{ij}) - S_{L_p}(y_j^{L_p}, n_j^{L_p})|$$

Hamming(2 function):

$$d\left(S_i, S_{L_p}\right) = \sum_{j=1}^{n} w_j^{(*)} \frac{\left|y_{ij} - y_j^{L_p}\right| + |n_{ij} - n_j^{L_p}|}{2}$$

Hamming(3 function):

$$d\left(S_i, S_{L_p}\right) = \sum_{j=1}^{n} w_j^{(*)} \frac{|y_{ij} - y_j^{L_p}| + |n_{ij} - n_j^{L_p}| + |h_{ij} - h_j^{L_p}|}{3}$$

$$\text{with } h_{ij} = 1 - y_{ij} - n_{ij}$$

Manhattan:

$$d\left(S_i, S_{L_p}\right) = \sum_{j=1}^{n} w_j^{(*)} \frac{(|y_{ij} - y_j^{L_p}| + |n_{ij} - n_j^{L_p}|)}{y_{ij} + n_{ij} + y_j^{L_p} + n_j^{L_p}}$$

Ngân:

$$d\left(S_i, S_{L_p}\right) = \sum_{j=1}^{n} w_j^{(*)} \frac{(|y_{ij} - y_j^{L_p}| + |n_{ij} - n_j^{L_p}|)/4 + |\max\left(y_{ij}, n_j^{L_p}\right) - \max(y_j^{L_p}, n_{ij})|/2}{3}$$

Mincowski:

$$d\left(S_i, S_{L_p}\right) = |\sum_{j=1}^{n} w_j^{(*)} |S_i(y_{ij}, n_{ij}) - S_{L_p}(y_j^{L_p}, n_j^{L_p})|^p|^{\frac{1}{p}}$$

2. Code implementation

Because, we have too many functions as stated above, we only demo the default measurement case. The rest of functions can be implemented similar to the code below:

```
#Calculation the distance between each instance and the labels in the label list
    def CalDistance(self, isTrain = True, measure = "Default"):
        self.measure = measure.lower().strip() #change name of measurement to lowercase
        if isTrain == True: #if True, use for training set and otherwise for the testing set
            length = self.numberOfInstances #number of points to calculate distance
        else:
            length = self.numberOfTest
        result = [[0 for idx in range(self.numberOfLabel)] for j in range(length)]
        if self.measure == "default": #check the measurement
            for idx in range(length):
                for idxLabel in range(self.numberOfLabel):
                    distance = 0
                    for idxCol in range(self.numberOfHeader):
                        if isTrain == True: #if True, use for training set
                            distance += self.weights[idxCol] *
abs(self.CalWeightsComponentS(self.RelTableTrain[idx][idxCol], \
                                self.SurgenoTableTrain[idx][idxCol]) -
self.CalWeightsComponentS(self.RelCenterTable[idxLabel][idxCol], \
                                    self.SurgenoCenterTable[idxLabel][idxCol]))
                        else: #use for testing set
                            distance += self.weights[idxCol] *
abs(self.CalWeightsComponentS(self.RelTableTest[idx][idxCol], \
                                self.SurgenoTableTest[idx][idxCol]) -
self.CalWeightsComponentS(self.RelCenterTable[idxLabel][idxCol], \
                                    self.SurgenoCenterTable[idxLabel][idxCol]))
                    result[idx][idxLabel] = distance

        if isTrain == True: #save the result of calculation with the respectively dataset
            self.result = result
        else:
            self.Tresult = result
```

3.1.5. Classification

1. Idea

In the fifth step, we only need to put S_i belong the class $S_{L_{p^*}}$ such that $d(S_i, S_{L_{p^*}}) = \min_{p=1,2,...,k}\{d(S_i, S_{L_p})\}$.

2. Code implementation

```
#Classification the label of instances
    def ClassificationLabel(self, isTrain = True):
        if isTrain == True: #if True, use for training set otherwise for testing set
            length = self.numberOfInstances #number of points to classify (training set)
        else:
            length = self.numberOfTest #number of points to classify (testing set)
        predictLabel = [0 for i in range(length)]
        for idx in range(length):
            if isTrain == True:
                minDistance = min(self.result[idx]) #minimum value in dataset (training set)
                k = self.result[idx].index(minDistance) #index of label in labels list (*)
            else:
                minDistance = min(self.Tresult[idx]) #minimum value in dataset (testing set)
                k = self.Tresult[idx].index(minDistance) #(*)
            predictLabel[idx] = self.label[k]
        if isTrain == True: #if True, save the result to evaluate for training set
            self.predictLabel = predictLabel
        else: #save the result to evaluate for testing set
            self.predictLabelT = predictLabel
```

3.1.6. Evaluation

1. Idea

Before evaluating the result, we will introduce some evaluation method that are used in our model. Firstly, we have to create the confuse matrix to find necessarily value:

		Actual values	
		Positive (1)	***Negative (0)***
Predicted values	***Positive (1)***	**TP**	**FP**
	Negative (0)	**FN**	**TN**

Note:

TP: (true positive) You predicted positive and it's true.

FP: (false positive) you predicted positive and it's false.

FN: (false negative) you predicted negative and it's false.

TN: (true negative) you predicted negative and it's true.

TP, FP, FN, TN will be used for binary classification problem. But with multi-classification problem, we only use TP, FP, FN to evaluation the result of our model. We will have some evaluation method as follows (with multi-classification problem, the value of TN is equal to 0):

Accuracy:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

Sensitivity:

$$SEN = \frac{TP}{TP + FN}$$

Specificity:

$$SPE = \frac{TN}{TN + FP}$$

Precision:

$$PRE = \frac{TP}{TP + FP}$$

F1 score:

$$F1 = \frac{2TP}{2TP + FP + FN}$$

Matthews correlation coefficient:

$$FMI = \frac{TP\ TN - FP\ FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

2. Code implementation

Because one of the most important method in evaluation phase is createConfuseMatrix, so we only introduce about this method and control method. Both evaluation methods which are recommended above will be implemented easily and you should try by yourself.

```
    #Create confuse matrix based on the result list
    def createConfuseMatrix(self):
        if len(self.setLabel) == 2: #check the dataset is binary or multi classification
            self.binary = True
        self.confuseMatrix = [[0 for col in range(self.numberOfLabel)] for row in
range(self.numberOfLabel)]
        for i in range(self.numberOfInstances):
            k = self.setLabel.index(self.truth[i]) #label of the actual value
            j = self.setLabel.index(self.predict[i]) #label of the predicted value
            if k == j: #if True, means that the label is predicted correctly (True)
                self.confuseMatrix[k][j] += 1
```

```
                if self.binary == True and k != 0: #add TN for binary classification
                    self.TN += 1
                else: #add for TP for multi-classification
                    self.TP += 1
            else: #predicted not correctly and use for both binary and multi classification
                self.confuseMatrix[j][k] += 1
                if j < k:  #add for FN
                    self.FN += 1
                elif j > k: #add for FP
                    self.FP += 1
```

Next, we will go to control method in evaluation phase.

```
#evaluate the result of our model based on demand of user
def Run(self):
        self.createConfuseMatrix() #create confuse matrix to find TP, TN, FP, FN
        method = self.method.lower().strip() #change evaluation method to lowercase
        result = 0
        if method == "accuracy": #call accuracy evaluation method
            result = self.accuracy()
        elif method == "sensitivity": #call sensitivity evaluation method
            result = self.sensitivity()
        elif method == "specificity": #call specificity evaluation method
            result = self.specificity()
        elif method == "precision": #call precision evaluation method
            result = self.precision()
        elif method == "f1_score": #call f1 score evaluation method
            result = self.f1_score()
        elif method == "matthews_correlation_coef": #call matthews coef relation method
            result = self.Matthews_correlation_coef()
        return result
```

3.2. Testing phase

3.2.1. Data fuzzification

The first thing in the testing phase is that we fuzzy the input data points of the test set according to the formulas available in the training phase. This step is completely similar to step 2.1 of the training part, except that it is used for the testing set.

3.2.2. Calculating distance

After the training phase, we have obtained the optimal weight for the classification problem and used it for the distance formula mentioned in section **3.1.4**. Unlike the training phase, we already have a set of weights and just need to put it on the test dataset after it has been fuzzyred.

3.2.3. Classification and accuracy

The first step of this part is similar to section **3.1.5** and **3.1.6**.

IV. Experimental

1. Dataset

The table below shown some of information related to datasets:

Dataset name	# features	# classes	# instances in dataset	# instances in training set	# instances in testing set
AlgerianForestFires	10	2	244	169	75
BreastCancerWinconsin	10	2	683	545	138
CervicalCancerBehaviorRisk	19	2	73	56	17
Humidity	5	2	9752	7800	1952
Kahraman	5	4	403	258	145
OccupancyEstimation	16	2	10129	8102	2027

Comment: All 6 datasets have different number of data points, there are 2 datasets with large number of data points, the rest have relatively little data. This can lead to underfitting when the data sets do not meet the size requirements. In addition, all data sets have a small number of labels, so this is also an advantage to test the problem from the beginning.

2. Result of datasets

After applying CS-IFS algorithm for datasets, we receive some interesting result is shown as the tables below:

DATASET NAME / CRITERION		1	10^{-1}	10^{-2}	10^{-3}	10^{-4}
ALGERIANFORESTFIRES	Training	0,940	0,905	0,745	0,740	0,740
	Testing	0,960	0,880	0,667	0,653	0,653
BREASTCANCERWINCONSIN	Training	0,936	0,936	0,895	0,791	0,791
	Testing	0,899	0,899	0,840	0,768	0,768
CERVICALCANCERBEHAVIORRISK	Training	0,839	0,839	0,768	0,768	0,768
	Testing	0,938	0,938	0,750	0,750	0,750
HUMIDITY	Training	0,954	0,993	0,993	0,992	0,992
	Testing	0,964	0,991	0,992	0,990	0,990
KAHRAMAN	Training	0,760	0,760	0,818	0,814	0,810
	Testing	0,697	0,697	0,862	0,869	0,869
OCCUPANCYESTIMATION	Training	0,911	0,911	0,870	0,870	0,870
	Testing	0,910	0,910	0,866	0,866	0,866

Table 1. Result related to criterion (with measure = "Default", evaluation = "accuracy")

Beside, we will have two graphs which is shown as below for easy understanding:

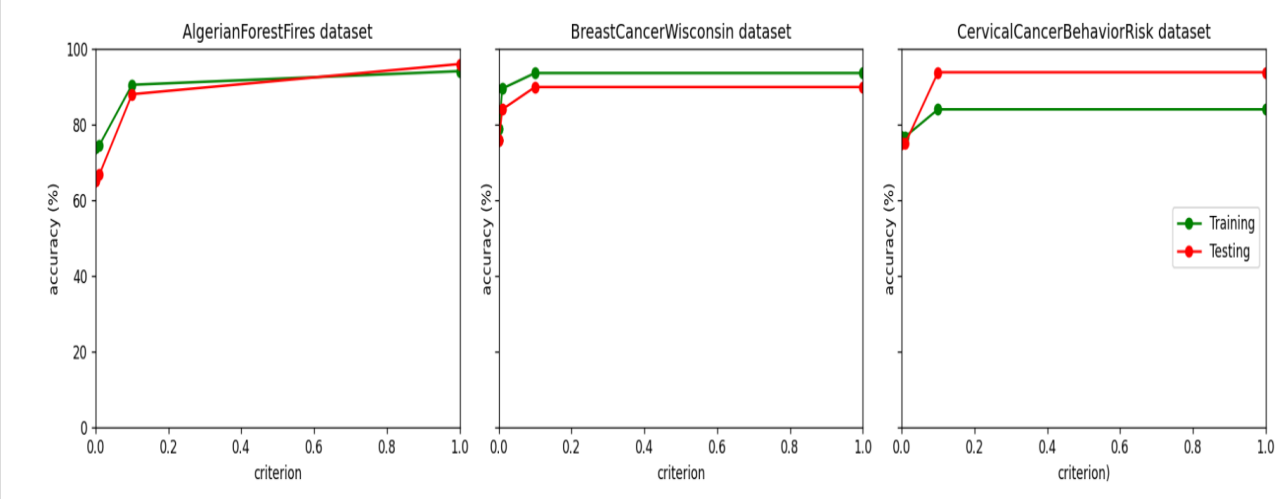


Figure 2: Result of the first three datasets (changing criterion).

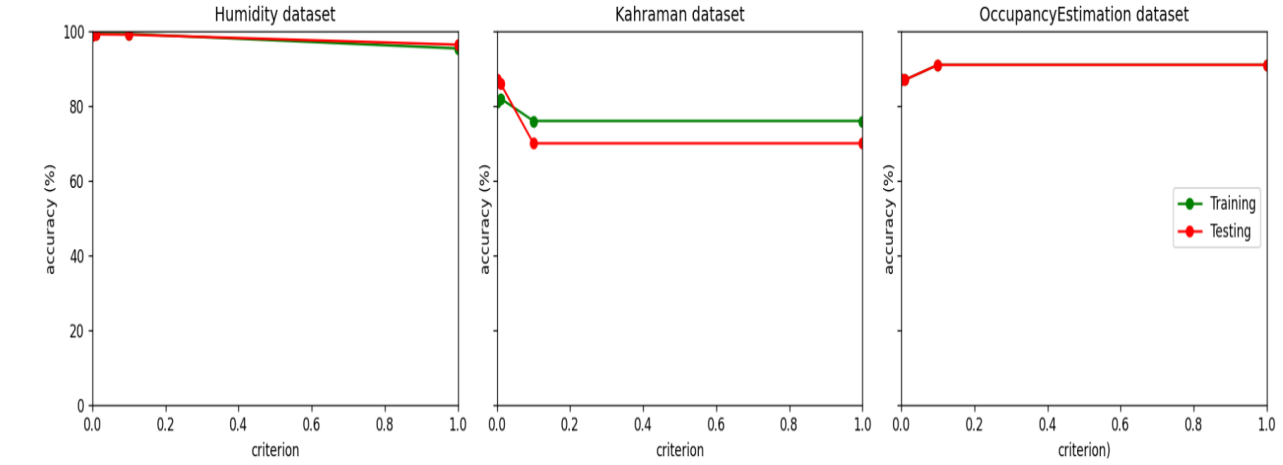


Figure 3: Result of the last three datasets (changing criterion)

DATASET NAME / MEASUREMENT		DEFAULT	MANHATTAN	HAMMING	MINCOWSKI (P = 2)
HUMIDITY	Training	0,992	0,975	0,971	0,992
	Testing	0,948	0,937	0,936	0,948
KAHRAMAN	Training	0,824	0,785	0,785	0,825
	Testing	0,774	0,755	0,755	0,774
OCCUPANCYESTIMATION	Training	0,869	0,869	0,869	0,863
	Testing	0,870	0,870	0,870	0,860

Table 2. Result related to measurement (with criterion = 10^{-4}, evaluation = "accuracy")

Next, we will go to the results of measurement in our model.

DATASET NAME / EVALUATION	SPECIFICITY	SENSITIVITY	PRECISION	F1 SCORE

HUMIDITY	Training	0,970	0,999	0,992	0,995
	Testing	0,803	0,999	0,935	0,966
KAHRAMAN	Training	0,000	0,924	0,876	0,904
	Testing	0,000	0,872	0,911	0,872
OCCUPANCYESTIMATION	Training	0,000	0,901	0,960	0,930
	Testing	0,000	0,902	0,961	0,930

Table 3. Results related to evaluation (with criterion = 10^{-4}, measurement = "Default")

3. Conclusion

As you can see, the results of all 6 datasets are more than 70% accurate in both training and test sets. Some data sets even have more than 90% accuracy, which is an acceptable result when the new algorithm is being implemented in the early stages. As can be easily seen, the data sets with a large number of data points (greater than 5000 data points) and with a small number of labels needing a small number of categories achieve the expected classification results. This can be explained because the data has met enough size requirements for the algorithm to detect and learn the characteristic properties of the dataset. With smaller datasets, most of the accuracy is at 80%, this is because the size is not enough for the algorithm to learn the features of the dataset.

With different measurement methods and different evaluation methods, the model will give different results. This can serve the user's desire to see the results obtained by the problem because each different evaluation represents a different point of view.

In data set A and dataset B, a relatively obvious overfit occurred because the results obtained in the training set were significantly higher than those obtained in the test set. In other data sets, this phenomenon does not occur.

V. Discussion

In the report, we mentioned some interesting things about our model. The model gives good results with some datasets and promises to be better in the future. However, it also exposes some disadvantages, such as: overfitting occurs frequently and we update weights are not based on a specific objective function.

For overfitting, we intend to develop some additional tools or support methods that are similar to some of the other classification methods such as data-point or a batch points.

As for the objective function, we will try to develop the objective function for our method. Things like updating the weights will be based on the change in the derivative of the objective function.

In addition, we will add a few methods of data splitting to fit small or medium-sized datasets which don't have high accuracy or may occur underfitting phenomenon.

VI. Reference

[1] Presentation slide of researcher Nguyen Xuan Thao

[2] Reference book provided by Prof. Pham Van Hai