E.T.S. of Industrial Engineering,
Computer Science and Telecommunication

STUDY OF DIFFERENT KNN ALGORITHM VERSIONS

International Degree in Computer Engineering

Final Degree Project

Javier Sospedra Legarda

Tutor: Jose Antonio Sanz Delgado

Pamplona, 25/02/2022

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Javier Sospedra Legarda

ACKNOWLEDGMENTS

I am really grateful to Jose Antonio Sanz Delgado, director of this final degree project and to all the people that helped me finish this project.

ABSTRACT

K-Nearest Neighbor algorithm has been proven to be a simple and effective method for classification problems in machine learning. This Final Degree Project is based on the investigation of the KNN (K-Nearest Neighbors) algorithm introducing some changes that improve said algorithm: fuzzy logic, fuzzy intervals and evolutionary algorithms.

First, a model using fuzzy logic and a model with fuzzy intervals are created, which improve to a certain extent the accuracy of the original model (KNN). Next, the evolutionary algorithm is used to try to improve the performance of the model further. The problem is the time required for the convergence of this last algorithm. Therefore, it is intended to make a comparison between all these models and see the difference between them in both performance and time.

Key Words: Classification, KNN, Fuzzy, evolutionary, algorithm.

.

INDEX

Content

1. INTRODUCTION AND OBJECTIVES

Nowadays, we are collecting massive amount of data and using it for several purposes. The main purpose is to make more accurate decisions and to make them quicker. The way we do that is through different machine learning algorithms that use this data to make better predictions. It can be applied to several fields such as personalised marketing, self-driving cars, fraud detection, voice assistants…

Machine learning, according to IBM, is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy [1]. Machine learning models have the ability to observe patterns that humans would overlook. It uses the data continuously to learn over time.

One of the most important tasks that can be achieved with this great amount of data using machine learning is classifying. The aim in classification is to label in groups, so we are able to make better decisions creating the ability to predict. There are some well-known big companies (Google, Meta…) that are using classification to get more knowledge about our interests such as creating groups of people with similar hobbies.

A simple and powerful algorithm for these classification problems is the supervised machine learning K-Nearest Neighbor (KNN) algorithm [2]. This model is trained saving all the training labelled data, so it would be like a model itself, and then makes predictions based on this previous data that is already labelled. For the prediction of the class, it takes into account the k nearest neighbors data points of the example we want to classify based on the training examples it saved at the beginning.

Although KNN is a very powerful algorithm for classification, there have been presented some improved versions of this original KNN model such as the original Fuzzy K-Nearest Neighbor (FKNN), Fuzzy K-Nearest Neighbor with Intervals (IVFS KNN) and the Evolutionary K-Nearest Neighbor with Intervals version, which each of these models is supposed to improve the previous one.

This Fuzzy version [3] makes use of fuzzy logic to increase KNN's original accuracy. It assigns a membership from [0, 1] to a class for each instance so it describes how sure it can be that an example belongs to a specific class making use of a parameter called kInit. It removes the full membership that original KNN used and instead applies the idea that instances which are close to the center of their class will have almost full membership, on the other hand, instances nearby the decision boundaries will share about half of their membership between the different classes. Then, the memberships are weighted to generate a final vote

for each neighbor and class and achieve an improved classification introducing another parameter m.

The IVFSKNN [4] model is used to try to reduce the difficulty in assigning values to the hyper parameters of FKNN. It uses fuzzy interval logic to achieve that assigning an interval for each membership. The minimum and maximum value will change based on how relevant and how many neighboring instances there are near the training point. Moreover, this model also makes use of the parameter kInit and two m values m_a and m_b for the interval making it more flexible.

As explained, both previous versions of KNN introduce the parameters kInit and m to make the classification. Although other optimization algorithms could be applied for selecting the best values for these parameters, the evolutionary version [5] is introduced for that. It uses the genetic algorithm CHC with a double codification of the chromosome mixing binary representation for the kInit parameter selection and real representation for m_a and m_b optimization.

The main objective of this project is to make a comparison between these improved algorithms and check if they really have a better performance than the original KNN algorithm and achieve more accurate predictions. Another feature to take into account is the time it takes to train these different models. As it is known, the last Evolutionary KNN model is a genetic algorithm which needs a huge amount of time to finish its execution. Therefore, will it really be worth using this algorithm considering the time it takes to finish its training?

2. PRELIMINARIES

2.1. MACHINE LEARNING

We could describe machine learning as a branch of artificial intelligence (AI) that uses computer systems and the use of data and algorithms to learn like humans do and make decisions. With machine learning, the precision of the decisions made improve through experience with the use of more and more data.

Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks [6].

Machine learning programs can perform tasks without being explicitly programmed to do so. It involves computers learning from data provided so that they carry out certain tasks.

There are simple tasks assigned to computers where it is possible to program algorithms telling the machine exactly how to do the thing, we want to execute to solve the problem; the computer doesn't need to learn anything. For more complex tasks, it can be challenging for a human to manually create the needed algorithms. It turns out to be more effective to let the machine create its own algorithm and probably with more time and more data it will improve that algorithm. This way we don't need anyone programming a specific algorithm.

Machine learning is known more and more every day. More people hear about it and there are more advances and uses in this discipline.

Figure 1: Number of machine learning Google searches over time from Google Trends

In Figure 1, we can see the huge increase on the numbers of machine learning Google searches these last years. This huge increase comes because the words machine learning or artificial intelligence appears a lot more in the news, on our mobile phones and on our daily life in general.

Machine learning is usually divided into 3 categories: Supervised Learning, Unsupervised Learning and Reinforcement Learning.

2.1.1. SUPERVISED LEARNING

The machine receives data from some inputs and some goal outputs. Then, it creates a mathematical model to learn and get a generalised rule that relates those inputs and outputs and can make future predictions for new unseen inputs [7].

The first input and outputs data given to the machine that is used for training is known as training data. As we see in a graphic summary in Figure 2, when the model has learned this training data, we will be sure it really has learned when we set some test data that the machine has never seen and it predicts the output for those new examples and we obtain the performance of the system.

The algorithms that are used in supervised learning are classification and regression. Classification algorithms are used when the outputs are restricted to a known set of values (1, 2, 3) and regression algorithms are used when the outputs can have any numerical value. (124.2, 148…).

An example of classifications algorithms would be the spam mail filtering. The output would be “spam” (1) or “no spam” (0). We would have the emails has input and we can see the output is restricted to only 2 values. On the other hand, an example of regression algorithms would be a house’s price prediction based on the number of dormitories and bathrooms it has. The outputs in this case could be any number such as 127.201$ or 523.125$.

One of the great uses of supervised learning is looking for similarities. We can use both classification and regression algorithms for measuring how similar 2 things are. We see this type of learning in important applications such as YouTube, that used recommendations systems making classifications on how one person has similar interests with another one. It is also used in face recognition systems, sentiment analysis, speaker verification, spam detection, image recognition…

Some of the main algorithm that are used in supervised learning are Neural Networks, which imitates the neurons in the human brain to learn some data

using weights for each connection which adjust based on the loss function calculated by the process of gradient descendent [8]; Naïve Bayes, that is a classification approach that makes use of the principle of class conditional independence from the Bayes Theorem and it is used a lot in problems such as spam detection and text classification; Linear Regression, which is used to make predictions based on one or more variables; Logistic Regression, unlike Linear Regression, it is used when the dependent variable is categorical and not continuous (0 or 1); Support Vector Machine (SVM), which it creates an hyperplane where the distance between 2 classes is at its maximum and it is used for both classification and regression problems; Random Forest, it is a group of decision trees which join for reducing variance and making better predictions [9]; and finally K-Nearest Neighbor (KNN), which classifies data bases on the distance to other data, the idea is that 2 data points that are found near each other are similar. This last algorithm will be explained better later because it will have great importance in this project.

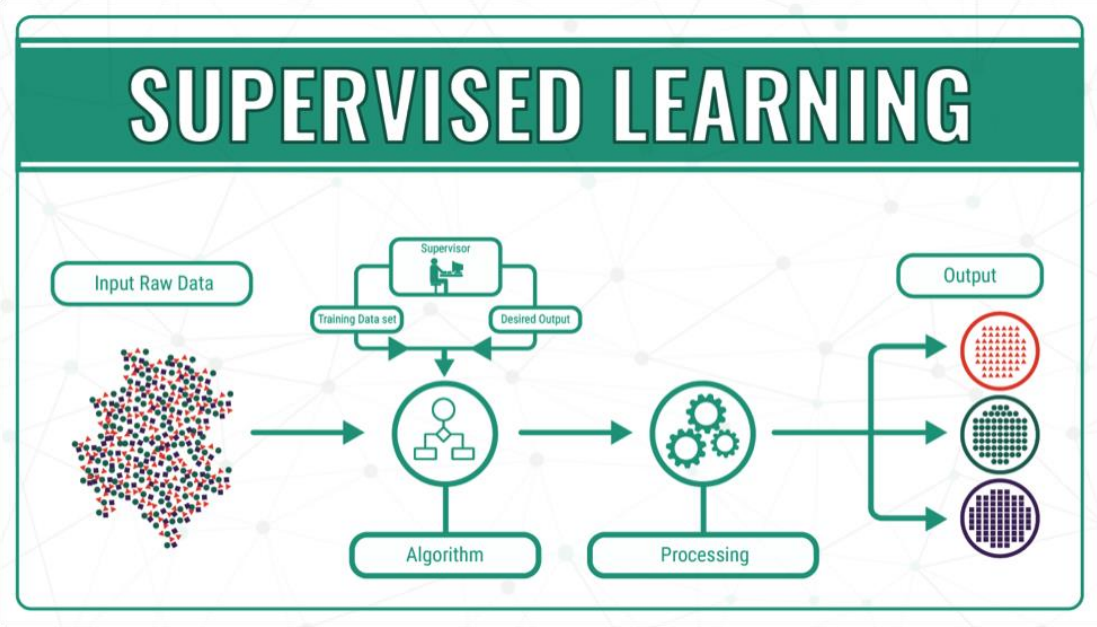


Figure 2: Supervised Learning

2.1.2. UNSUPERVISED LEARNING

In unsupervised learning, unlike supervised learning, it only takes a set of input data and finds a proper clustering of the data points. So basically, this type of method learns from data points that have not being classified before and it

identifies possible groups from the data as we can observe in Figure 4. The goal is to find patters in the data that classifies them without the need for human intervention.

When we have some clusters created, we assume that the data points in one group are similar between them and are not similar with the ones in another cluster.

We have several unsupervised learning algorithms such as association rules, dimensionality reduction, autoencoders or apriori algorithms but one of the main ones is Clustering, which we can observe in Figure 3 the functionality it has. It identifies similarities and differences between data points and we can difference between exclusive and overlapping, hierarchical and probabilistic clustering.

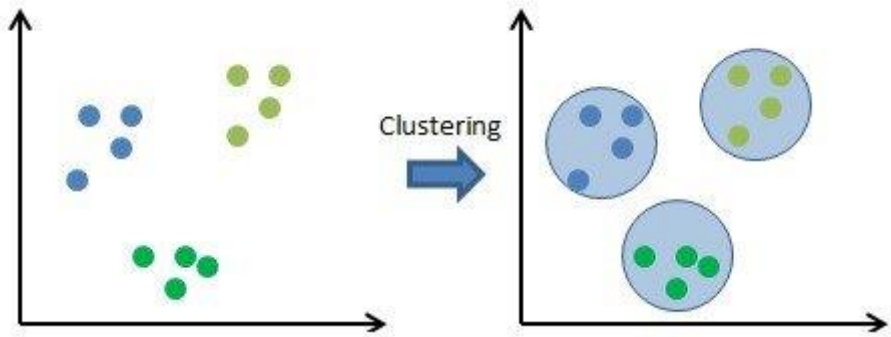


Figure 3: Clustering

Unsupervised learning is used for categorizing texts in their respective topic, for computer vision and object recognition, anomaly detection, recommendations engines…

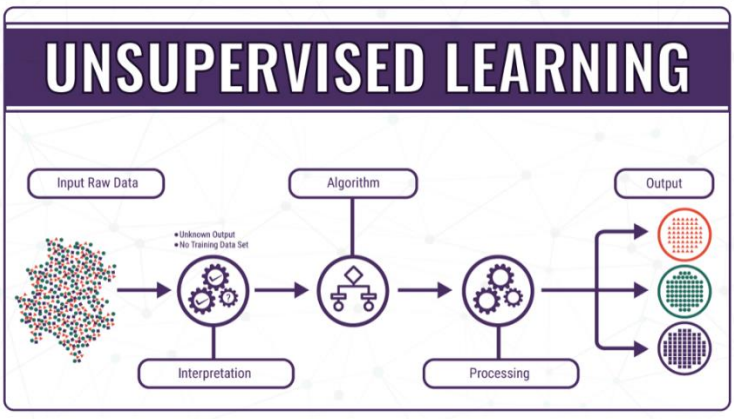


Figure 4: Unsupervised Learning

2.1.3. REINFORCEMENT LEARNING

Reinforcement learning [10][11] is based on some agents placed in a specific environment and have to take actions so their reward maximizes. Basically, when the agent does something "good" in the environment, it will receive a positive reward meaning that action that it took in that environment was good, otherwise, when the agent does something "bad", it will receive a negative reward meaning it doesn't have to take that action in that scenario as we can observe in Figure 5. It is used in many disciplines such as game theory, simulations, information theory, statistics and genetics algorithms.

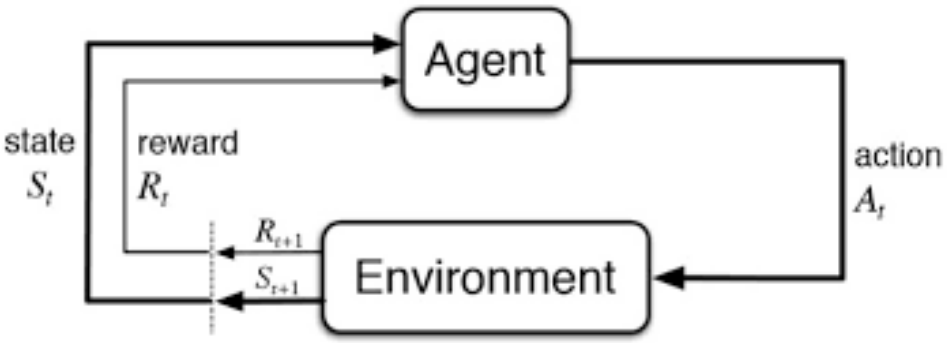


Figure 5: Reinforcement Learning

Unlike supervised learning, this type of learning doesn't need labelled input and output data. Instead, it uses exploration and exploitation to learn which are the optimal actions for the different states in the environment. The environment is typically in the form of Markov decision process.

Reinforcement learning is usually used in learning to play a game against a human, autonomous vehicles, trading…

2.2. CLASSIFICATION

As explained before, classification is a supervised learning problem in which the computer program learns from the given data and makes new predictions or classifications based on the trained model.

2.2.1. PERFORMANCE METRICS AND MODEL EVALUATION

The most important part after we have finished training our model is the evaluation of it to check its accuracy and efficiency. One of the most common problems in machine learning can be overfitting. This occurs when the model knows “too well” the data which it has trained with and gets a very good accuracy in it, but later in the unseen data doesn’t perform well enough. There are several ways in which we can evaluate a classifier and prevent this mentioned problem. We can check them below:

- **Holdout Method**

It is the most common way to evaluate the classifier. The data is divided into 2 parts in this method, one train set and one test set over 80% and 20% respectively as an example. The first set is used for training the model with its data and the second set is unseen data which will be used to test how well the model is predicting this new data.

- **Cross-Validation**

K-fold-cross-validation can be used to check if the model is over fitted.

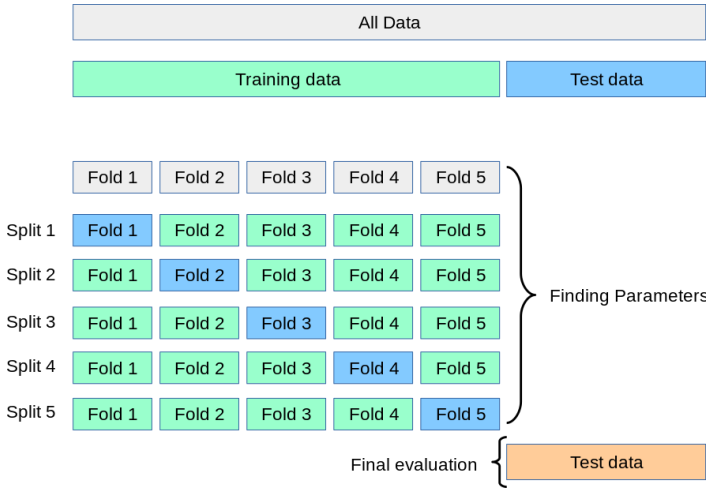


Figure 6: Cross Validation from https://scikit-learn.org/stable/modules/cross_validation.html

This method works by making a random partition of the data set into k exclusive data subsets as we see in Figure 6, where each of them is of the same size. One of these k subsets is kept for testing and the others are used to train the model. For the classification, a procedure called stratification that

maintains the same percentage of examples of each class in each k fold is used. The same process is used for all k folds. In this project we will use this method with a k fold of 10.

Now, these are the ways to be sure how well is predicting our classifier:

- **Confusion Matrix**

Each row of the matrix represents the instances of the real class while each column represents the instances of the predicted class. This is a good way to check if the model is having trouble predicting some of the classes.

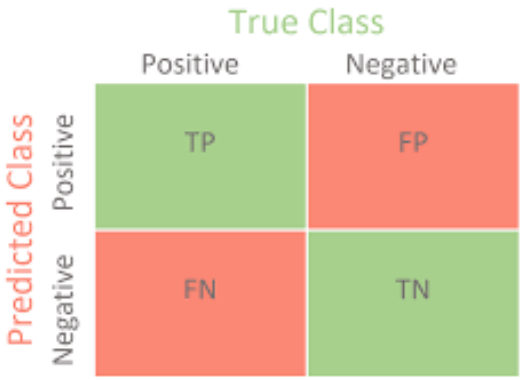


Figure 7: Confusion Matrix

As we see in Figure 7 which presents the confusion matrix, we need to understand some concepts:

- True Positive (TP): Number of correct predictions where the instance is positive.

- True Negative (TN): Number of correct predictions where the instance is negative.

- False Positive (FP): Number of incorrect predictions where the instance is positive.

- False Negative (FN): Number of incorrect predictions where the instance is negative.

- **Accuracy**

It is the relationship between the number of correct predictions and the total number of predictions done.

$$\text{Accuracy} = \frac{\text{Num.Correct Predictions}}{\text{Num.Total Predictions}}$$

- **Error rate**

It is the relationship between the number of incorrect predictions and the total number of predictions done.

$$\text{Error Rate} = \frac{\text{Num.Incorrect Predictions}}{\text{Num.Total Predictions}}$$

These 2 last methods can give problems when there are a lot more instances of one class than from another. For example, in a cancer detection problem, the chances of actually having cancer are very low. For this example, let's say 10% of the patients have cancer and the other 10% don't have it. In this type of problem, we really don't want to miss a patient who is having cancer but isn't detected. If the model say that anyone has cancer gives, it will give an accuracy of 90%. The model didn't really work well but it only predicted all the examples as the same class. For this reason, we need better metrics like the next ones:

- **Precision**

It is the relationship between the number of instances that result to be positive which the classifier has predicted as positive.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$$

- **Recall**

It is the relationship between the number of instances that are positive and the ones which the model identified.

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

- **True Negative Rate**

It is the percentage of negative instances out of the total actual negative instances.

$$\text{True Negative Rate} = \frac{\text{TN}}{\text{TN+FP}}$$

- **F-Score**

It is the harmonic mean of the precision and recall values

$$\text{F-Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision+Recall}}$$

- **ROC Curve**

A ROC curve (Receiver Operating Characteristic Curve) is a graph for visualizing the performance of a classification model showing the relationship between the true positive rate and the false positive rate [12].

$$\text{TPR} = \frac{\text{TP}}{\text{TP+FN}} \qquad FPR = \frac{\text{FP}}{\text{FP+TN}}$$

TPR = True Positive Rate

FPR = False Positive Rate

TP = True Positive

FP = False Positive

FN = False Negative

TN = True Negative

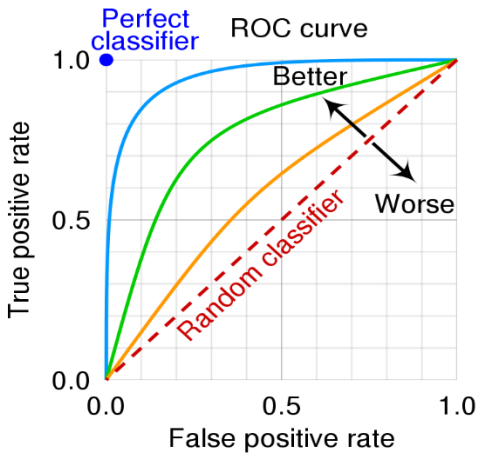


Figure 8: ROC Curve from https://commons.wikimedia.org/wiki/File:Roc_curve.svg

As we can observe in Figure 8, when the curve produced by the TPR and FPR is a straight line from the bottom left corner to the top right corner (red one) is when the model is classifying randomly, it is just guessing without any previous learning. From that base, we see that if the line goes below it, the model will get worse, so the objective is having the curve at the top left corner and be an almost perfect classifier so when the FPR is getting higher the TPR gets really high at the beginning almost to 1 and continues at very high values over the increase of FPR.

2.3. FUZZY LOGIC

2.3.1. DESCRIPTION

The term fuzzy logic was introduced with the 1965 proposal of fuzzy set theory by scientist Lotfi Zadeh. Fuzzy logic had, however, been studied since the 1920s, as infinite valued logic by Łukasiewicz and Tarski [13][14].

The term “fuzzy” refers to something that is difficult to perceive, indistinct or vague. Sometimes it is difficult to say if something is either true or false, it can be something in the middle.

Classical logic only makes statements that are either true or false (1 or 0). However, there are also propositions with variable answers. In the fuzzy system there is no true or false, there is no absolute truth, there is an intermediate value which is partially true or false.

This degree of partial truth has a range between 0 and 1 that states the probability of knowledge or ignorance of the statement as Figure 9 shows.

Fuzzy logic is based on the observation that people make decisions based on imprecise information which doesn’t use a number quantification. Fuzzy models can recognise, represent and use data and information that are vague and lack precision.

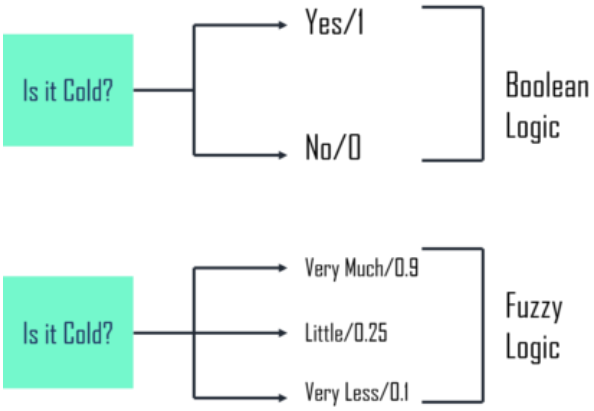


Figure 9: Fuzzy Logic Example from https://www.edureka.co/blog/fuzzy-logic-ai/

Fuzzy logic has been applied to many areas, from control theory to artificial intelligence.

This type of logic is used for the second KNN model created. It doesn't take into account complete memberships, on the contrary, it uses partial memberships from a range from 0 to 1. A fuzzy set is defined as a group of elements which have degrees of membership defined by a membership function.

2.3.2. MEMBERSHIP FUNCTION

The membership function is a function that defines how each point in the input space is mapped to a membership value between 0 and 1.

It allows to quantify different terms and represent a fuzzy set graphically. A membership function for a fuzzy set A on the universe of discourse X is defined as $\mu A: X \rightarrow [0,1]$

It quantifies the degree of membership of the element in X to the fuzzy set A.

- x-axis represents the universe of discourse.
- y-axis represents the degrees of membership in the [0, 1] interval.

There are largely three types of fuzzifiers:

- Singleton fuzzifier
- Gaussian fuzzifier
- Trapezoidal or triangular fuzzifier

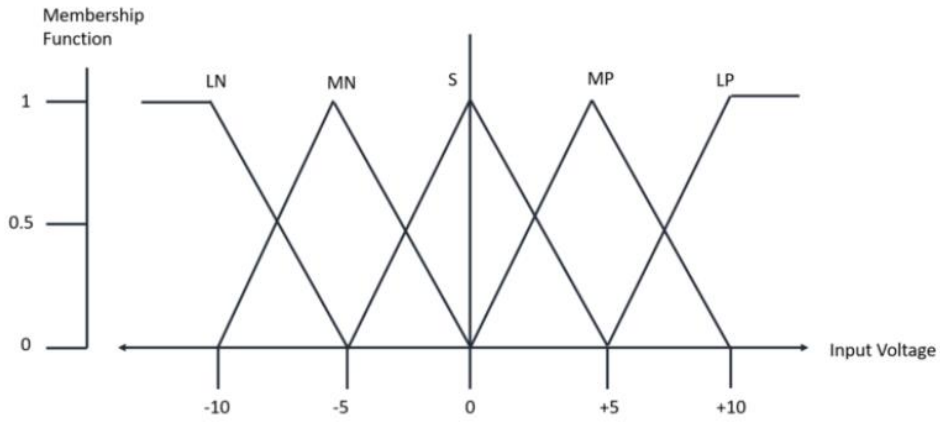


Figure 10: Triangle membership graph

2.4. INTERVAL VALUED FUZZY SETS

An interval valued fuzzy set (IVFS) is defined by a mapping F from the universe U to the set of intervals in [0, 1]. Let F(u) = [F∗(u), F∗ (u)]. The union, intersection and complementation of IVFS is obtained by canonically extending fuzzy set operations to intervals [14].

The main difference between fuzzy and intervals is that in the first case it tells us that the membership value of an element to a specific set is a number within the range [0, 1] and on the other hand, interval describes not as a number, but as an interval membership.

This interval-valued fuzzy sets are used in the third model created for KNN to improve the original one.

2.5. GENETIC ALGORITHMS

Optimization is the process to generate the final best solution to a given problem. Each optimization problem consists of 3 elements:

- **Objective function**: It is the function that needs to be optimized. We can have a problem that in the optimization a maximizing function is used and others that need a minimizing optimization.

- **Variables**: These are variables that directly affect to the objective function

- **Restrictions**: These are conditions that allow the variables to have some values and don't permit them to have other values.

The type of optimization that genetic algorithm use is much more effective than other algorithm in very large solutions spaces because if not suing this algorithm it may never finish the search for th optimal solution.

Moreover, this method is better than other machine learning algorithms because are much less sensitive to variations in the initial parameters.

A genetic algorithm is a search heuristic process inspired by Charles Darwin's theory of natural evolution. This algorithm selects the fittest individuals, by a natural selection process, and merges them as reproducing in a way that new individuals appear as offspring on the next generation. This type of algorithm will be used for the fourth algorithm developed in this project.

Genetic algorithms are a part of evolutionary computing. They are adaptive and use the information observed in time to guide the future search.

2.5.1. SEARCH OF SOLUTIONS

The set of all the possible solutions to a given problem forms the search space or set of solutions. Each solution is a point in that space and genetic algorithms will search for the best solution inside this search space.

Not all the space is going to be known by this algorithm, it is going to start by knowing some of it and then later, it will generate new points in that search space until it founds the solution.

2.5.2. GENETIC ALGORITHM PROCESS

The process of the genetic algorithm usually looks like this:

1- **Initial population**

A random population is created with N individuals in it called chromosomes, which are the solutions, that are a group of genes (parameters). So firstly, each individual has random genes.

2- Fitness function

This function determines how fitted each individual is and each individual's fit score will be used later as a probability for being selected for the next generation.

3- Selection

At this moment, it is time to select the fittest individuals which will reproduce and pass their DNA to the next generation.

4- Crossover

For each pair of parents selected based on their fitness score, there will be new offspring created by a crossover function that will join the genes from each parent. These new offspring will be then added into the new population

5- Mutation

Some of the genes of some individuals will be mutated with a low random probability set at the beginning of the process. Mutation is used for creating diversity within the population.

6- Replacement

The final step in in a genetic algorithm's iteration is replacement. It replaces the old population with a new population based on the generated offspring.

2.5.3. CHROMOSOME REPRESENTATIONS

A chromosome is a group of genes where all the information from an individual is. It can represent the information in several ways:

- **Binary coding**

It is a very common way to represent chromosomes. Each chromosomes uses a ones and zeros sequence like this:

0	1	0	0	1	0	1

where each gene corresponds to one bit.

This could be use, if we want to activate or use some attributes and not others. In the previous example attributes 2, 5 and 7 will be used but attributes 1, 3, 4 and 6 will not be used for that specific chromosome.

- **Integer coding**

There are problems when it is better to use other type of representations rather than using binary coding. In this representation, the chromosomes are strings or integers.

Generally, the individual

1	0	4	7	2

will have five genes each of them being 1, 0, 4, 7 and 2.

- **Real coding**

When using this representation, the chromosomes are real values which are inside and interval previously established.

Generally, the individual

13.2	6.8

will have two genes each of them being 13.2 and 6.8.

- **Permutation coding**

This representation is very useful when the order is important in the specific problem. Each gene of the chromosome will be a position inside this representation of the chromosome and are given by strings of integers.

For example,

1	5	4	3	6	2

could be a chromosome that represents the order to use to visit some cities travelling the least number of kilometres possible, so it in this example it will start in city 1 and finish in city 2 in the order of the genes of the individual.

2.5.4. CROSSOVER AND MUTATION METHODS

For each for these representations, there are different ways to proceed with the crossover and mutation process [15]:

1. **Crossover**
   a. Crossover on a point

   Given two binary chromosomes of length N, choose an integer x between 0 and N-1 and mix the first x genes of chromosome 1 with the N-1 second genes of chromosome 2 and then mix the first x genes of chromosome 2 with the N-1 second genes of chromosome 1.

   b. Crossover on n points

   It is a generalization of the previous one taking n points instead of 1.

   c. Uniform Crossover

   In contrast to the previous cases, the uniform crossing considers each binary gene individually and decides randomly from which parent it will be inherited. For this process, we generate a string of N random values between 0 and 1. If the value is below a specific parameter (normally 0.5), the value of the first parent is taken for that gene. If not, the value of the second parent. The second descendant is obtained by the inverse operation.

   d. HUX Crossover

   The idea is to introduce diversity generating very different offspring from their parents. It exchanges half of the genes which are different from their parents. With this type of crossover, we guarantee that the offspring have a maximum Hamming distance to their respective parents. It is used for binary representations.

   The Hamming distance is a metric for comparing two binary data strings. While comparing two binary strings of equal length, Hamming distance is the number of bit positions in which the two bits are different. It is used a lot in binary representation problems and it will be used later in one of the models.

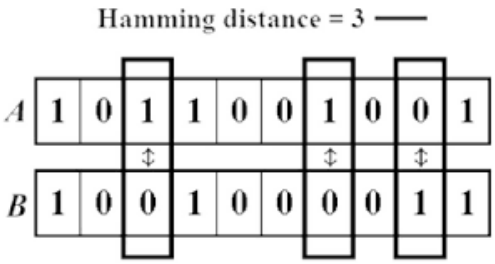


Figure 11: Hamming distance

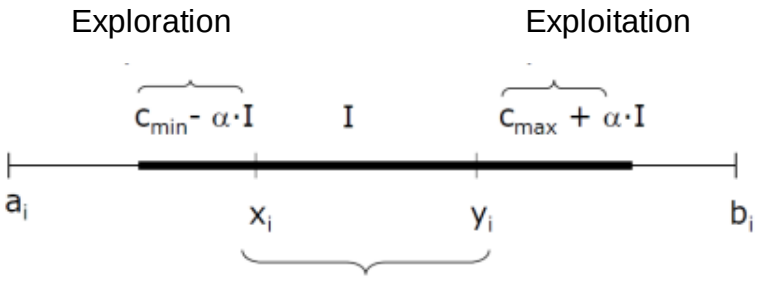
e. Blend Crossover (BLX)

It is used for real coding chromosomes. Having 2 chromosomes X and Y, BLX generates a new individual by choosing randomly a value in the interval:

$$[C_{min} - I * \alpha, C_{max} + I * \alpha]$$

$$C_{max} = \max\{x_i, y_i\}, C_{min} = \min\{x_i, y_i\}$$

$$I = C_{max} - C_{min}, \alpha \in [0, 1]$$



This way, we achieve the pursuit of both exploration and exploitation as we can observe in the image above.

f. Parent Centric Crossover (PCBLX)

PCBLX is very similar to BLX, it gets 2 individuals X and Y and generates a new chromosome choosing a random value between the interval:

$$[x_i - I * \alpha, x_i + I * \alpha] \text{ where}$$

$$I = abs(x_i - y_i), \alpha \in [0, 1]$$

In BLX, exploration was on both chromosomes and also exploration between them, on the contrary, in PCBLX there is exploitation and exploration nearby each chromosome but each offspring will be similar only to one of the parents. This crossover is also used for chromosomes using real representations.

2. Mutation

a. Binary Mutation

Each gene is considered separately. The idea is that each gene can change its value, from 0 to 1 and from 1 to 0, with a probability p.

b. Random Reset Mutation

It works the same way as binary mutation, but using integer coding. Taking into account a probability p, a gene of the chromosome will be changed for another one choosing it randomly from the possible values.

c. Random Reset Mutation

It is used for integer representations and it works adding a small positive or negative value to each of the genes of the individuals with a probability p. It uses a symmetric probability distribution, meaning that the probability for choosing x and -x is the same and the closer x is to 0, the more likely to be chosen.

d. Uniform Mutation

The value of a floating coding gene is changed to another random value in the interval [k1, k2] with a probability p.

e. Non-Uniform Mutation

It is one of the most common mutation processes for floating point encodings. The idea is to set a gaussian probability distribution with zero mean and a previously set deviation. For each of the genes, we choose a random real value according to the gaussian distribution with a probability p. We add this value which can be positive or negative and we get the new value for that specific gene. If the sum or subtraction is outside the interval

of all the possible values, we will take the respective border value of the interval.

f. Exchange Mutation

The simplest procedure for permutation encoding chromosomes is to simply swap two positions so the 2 genes will be exchanged.

g. Shake Mutation

The positions of a substring of th genes are randomly exchanged.

3. Selection

a. Roulette Method

It assigns a proportional selection probability to the fitness value of that chromosome, so the probability of an individual to be chosen would be: P (Ci) = f (Ci) / (f (C1) + ... + f (Ck)).

It can have premature convergence problems when the fitness values are very different from each other because the individuals which have a better fitness value will dominate the population easily. Moreover, when the fitness values are very similar, the new population will be selected randomly.

b. Lineal Order Method

It is based on the fitness values order of the individuals. Each chromosome has a selection probability S_i based on that order.

It uses ranges for making sure that all individuals have positive probabilities and it avoids the creation of individuals that provoke a premature convergence.

$$P(S_i) = \frac{range(fSi)}{N*(N+1)/2}$$

c. Tournament Method

It is one of the most used selection methods in genetic algorithms. The idea of the method is to make the chromosomes compete in “tournaments” with each other in groups and choosing the winner or winners of each tournament.

First, we select n individuals with the same probability, they will compete and be selected based on their fitness function and finally we repeat the process until we achieve the desired number of parents. There existed two types of tournaments: with replacement and without replacement.

d. Random Selection Method

For each i position, we obtain a random position between 1 and N and we exchanged it with the position i we are in. N/2 crossovers are made, being the parents 2*i and 2*i-1.

4. Replacement

It is the process in which the individuals of the population are replaced by the generated offspring. There are random and deterministic replacement methods. It differs depending on the genetic algorithm model.

There are two important genetic algorithm models: the generational model, which generates a completely new population with new chromosomes; and the stationary model, which two parents are chosen with the selection method and then the genetic operators are used. It works very efficiently when the worst individuals are replaced.

Generational Model

a. Classic Replacement

The complete population is replaced for the new offspring.

b. Elitism

The best chromosome of the population is never being replaced so we never remove the best individual.

c. High Elitisim

It merges the old population with the new one and then select the best N individuals.

Stationary Model

a. Replacement the worst individual

It generates high selective pression. It only replaces the chromosome if the offspring is better than the worst individual of the actual population.

b. Restricted tournament

The most similar one among N individuals is replaced (usually N=3).

c. Oldest replacement

The oldest chromosome is replaced.

d. Random replacement

A random chromosome is replaced.

e. Worst among like

The worst chromosome of the set of N individuals which are most similar of the generated offspring is replaced. It looks for some balance between diversity and selective pression.

2.5.5. CHC

2.5.5.1. DESCRIPTION

CHC (Crossover elitism population, Half uniform crossover combination, Cataclysm mutation) is one of the most used genetic algorithms nowadays due to its balance between diversity and convergence [16].

It introduces new concepts to genetic algorithms like elitism replacement, selection and crossover diversity with the HUX crossover method for binary chromosomes and incest prevention methods and the concept of reset.

2.5.5.2. METHODS

The idea is to generate new offspring different from their parents preserving the best individuals of the population. For this reason, CHC uses a random selection method explained previously. Moreover, it introduces this wanted diversity with the Uniform Crossover (HUX), also explained previously, which exchanges half of the genes which are different from their parents and it guarantees having a maximum Hamming distance. CHC is also characteristic because it doesn't use a mutation method.

2.5.5.3. INCEST PREVENTION

As this algorithm wants to prevent crossing very similar chromosomes, it introduces the idea of incest prevention. Hamming distance is used for calculating the distances between parents and then crossing the ones which are different from each other.

There exists a crossover threshold for crossing parents which differ in a certain number of bits. If the different number of bits is less than the threshold the individual is not copied.

The crossover_threshold starts with a value n/4 being n the length of the chromosome. The condition used is if half the Hamming distance divided by two is greater than the crossover threshold, then the crossover will be made. If there aren't new offspring created the threshold will be decreased by 1.

2.5.5.4. RESET

When the population has reached to a premature convergence, CHC renews the population as we observe in Figure 13. This occurs when the crossing threshold is less than 0. There are two ways for doing this process:

- Using the best element and including a copy of it.
- Using the best or parts of the best individuals and choosing the rest ones randomly

When the reset is done, the threshold value is also reset to its initial value n/4.

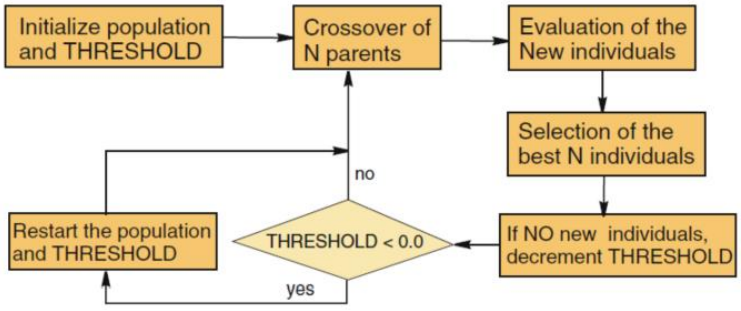


Figure 12: CHC evolutive model

2.5.5.5. STOP CRITERION

There are three main reasons the algorithm will stop for which are: when the optimal solution is achieved, when the number of maximum iterations has been reached and when there have passed three resets without improvement.

2.5.5.6. CODIFICATIONS

Although, CHC is designed for binary codification, it can be used with real codification too making some adaptions.

For real codification BLX or PCBLX crossover methods are used for maintaining a balance between exploitation and exploration. Moreover, the reset process maintains the best individual and the rest ones are generated randomly.

For incest prevention, a transformation from real to binary is needed. The real values are coded in binary using #Bitsgene, a number of bits per gen previously established, being n*#Bitsgene the length of the binary transformed chromosome. The initial threshold in this case is (n * #Bitsgene) / 4. In this procedure, Gray codification is used for this reason to transform a real value into a sequence of bits and then later calculate the Hamming distance.

It uses the Hamming distances too and the real coded threshold is reduced by #Bitsgene when resetting.

2.5.5.7. DOUBLE CODIFICATION

There are problems in which we will have to mix both real and binary encoding. In these cases, the methods for crossover will be applied differently for each part of the chromosome.

When using this multiple codification, there will be 2 thresholds, the one for real representation (n1*#Bitsgene)/4 and the one for binary representation n2/4, being n1 the length of real values in the chromosome and n2 being the length of binary genes in the chromosome. That means, n1+n2=n and n will be the total length of the chromosome.

For crossover, PCBLX will be applied to the real coded part and HUX for the binary part checking in both cases its respective thresholds. When using this double codification, the incest prevention will be applied as follows:

```
if RDistance/2 > RThreshold and BDistance/2 > BThreshold
    Apply both HUX and PCBLX crossover combined
if RDistance/2 > RThreshold
    Apply PCBLX to the real part and copy the binary part
if BDistance/2 > BThreshold
    Apply HUX to the binary part and copy the real part
```

being RDistance and BDistance the Hamming distances for real and binary encoding respectively and RThreshold and BThreshold the thresholds for real and binary encoding respectively.

2.6. DEVELOPMENT ENVIRONMENT

2.6.1. PYTHON

The programming language selected for this project is Python due to several reasons.

Python is very easy to understand and has a simple syntaxis. Moreover, a lot of people use of Python so there is a big community that comes in handy for doubts or errors that a programmer can have while coding. Finally, one of the main reasons is because this project is about a machine learning algorithm (KNN) and Python is very useful in this area because it has lots of big libraries about this topic.

2.6.2. JUPYTER NOTEBOOK

The chosen framework for making all the experiments and analysis of this final degree project is Jupyter Notebook.

Jupyter Notebook is a web-based interactive computational environment for creating notebook files (.ipynb) based on cells that can contain text or code.

Jupyter Notebook is very easy to use specifically when you want to have all organised and you can just execute each cell at a time. It is also extremely useful for making experiments and for making analysis of different executions.

2.6.3. DEAP LIBRARY

As explained previously, the fourth model developed in this project (the evolutionary one) has been developed using a genetic algorithm.

The Distributed Evolutionary Algorithms in Python (DEAP) library is an evolutionary computation framework for fast prototyping and testing ideas in Python that has been used for this purpose. This library is very powerful because it has many types of crossover functions, mutation functions, different chromosome types developed in it so it makes easier and quicker to developed any idea or project someone wants to develop.

3. PROBLEM STATEMENT

The goal of this project is to make a comparison of four different K-Nearest Neighbor algorithms: original K-Nearest Neighbor (KNN), original Fuzzy K-Nearest Neighbor (Fuzzy-KNN), Fuzzy K-Nearest Neighbor with Intervals (IVFS KNN) and the Evolutionary K-Nearest Neighbor with Intervals version.

3.1. ORIGINAL K-NEAREST NEIGHBOR

3.1.1. DESCRIPTION

This algorithm is the base one, the original K-Nearest Neighbor (KNN) [2][17]. It is an easy to implement supervised machine learning algorithm that can be used for both regression and classification problems. Although it is an easy to implement algorithm, it works very well in many problems and different situations.

This KNN approach doesn't create a whole model, it saves all the data so it could be defined as a model itself. The idea is based on using a distance as similarity function between the training data points and the test point that we want to classify, so it assumes that points that are near each other, have things in common. It finds K points of the data set that have more in common with the test point so that we classify it taking into account this K objects. That is the reason the KNN algorithm is an example of what it is called a lazy learning technique, that is, a technique that waits until it gets the test point to generalize using the training data.

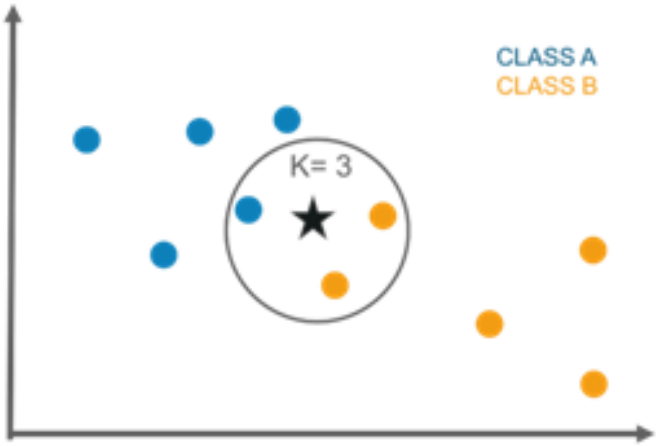


Figure 13: K-Nearest Neighbor Algorithm with K = 3 from https://www.jparzival.com/blog/algoritmo-knn/

As we can see in Figure 14, the training points are the ones in yellow and blue and the test point which we want to classify is the star one. We are using K=3 so the approach will take into account the 3 training points which are nearer the star point. In this case we get 2 yellow ones and 1 blue point, so the prediction we will give is to classify the test point as yellow class.

3.1.2. ALGORITHM

The basic algorithm would be:

Algorithm 8.1 Basic kNN Algorithm

Input : D, the set of training objects, the test object, $\mathbf{z}$, which is a vector of attribute values, and L, the set of classes used to label the objects

Output : $c_z \in L$, the class of z

foreach object $\mathbf{y} \in D$ **do**

| Compute $d(\mathbf{z}, \mathbf{y})$, the distance between $\mathbf{z}$ and $\mathbf{y}$;

end

Select $N \subseteq D$, the set (neighborhood) of k closest training objects for z;

$c_z = \underset{v \in L}{\operatorname{argmax}} \sum_{y \in N} I(v = class(c_y))$;

where $I(\cdot)$ is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

Algorithm 8.1 provides a summary of the nearest-neighbor classification method. Given a training set D and a test object z, which is a vector of attribute values and has an unknown class label, the algorithm computes the distance (or similarity

between z and all the training objects to determine its nearest-neighbor list. It then assigns a class to z by taking the class of the majority of neighboring objects. Ties are broken in an unspecified manner, for example, randomly or by taking the most frequent class in the training set.

The storage complexity of the algorithm is O(n), where n is the number of training objects. The time complexity is also O(n), since the distance needs to be computed between the target and each training object. However, there is no time taken for the construction of the classification model. Thus, KNN is different from most other classification techniques which have moderately to quite expensive model-building stages, but very inexpensive O(constant) classification steps.

3.1.3. METRIC'S EVALUATION

An important issue is the selection of the distance measure. The most common ones are the Euclidean (Figure 15) and Manhattan (Figure 16) distances:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(yi - xi)^2}$$

Euclidean distance

$$d(x, y) = \sqrt{\sum_{i=1}^{n}|yi - xi|}$$

Manhattan distance

We know that the Euclidean distance gives problems when the number of attributes increases. Attributes need to be scaled for not giving more importance to attributes with high values such as price of a house over attributes like number of bathrooms in a house which are lower.

One of the other main concerns that can affect the KNN's performance is the selection of the k value. When we choose a too low value for k it will affect negatively as it will be affected by noise points. On the contrary, when a too large k value is used, the neighborhood might be too big an contain many points including many points from other classes.

There are different methods to combine the votes for each class. In first place, we have the main and simple way that is taking a majority vote between these k instances. This approach can be a problem due to the k neighbors can vary a lot in their distance to the test point, so it would be interesting to take more into account the vote from one neighbor that is the closest than the one which is

farther away. This gives the idea to use a more complex method that takes into account weights of each instance's distance to the test point. The approach used is using the reciprocal of the squared distance:

$$w = \frac{1}{d(y,z)^2}$$

3.2. ORIGINAL FUZZY K-NEAREST NEIGHBOR

3.2.1. DESCRIPTION

In this version [3], training instances of KNN are labelled using a different method. It uses memberships in the range of [0, 1] and not full membership as it was in original KNN where it was just 0 if the instance wasn't in that class or 1 if it was. For the contrary, this approach uses a degree of membership for each class. For instance, an example that would be mainly of class A and not much of class B, might have membership of 0.95 of class A and a membership of 0.05 of class B. This approach has been shown to be an effective improvement of the original KNN algorithm achieving better accuracy rates in classification problems.

This membership is obtained by this function:

$$U_c(x) = \begin{cases} 0.51 + (nn_c/kInit) * 0.49 & \text{if c} = \omega \\ (nn_c/kInit) * 0.49 & \text{otherwise} \end{cases}$$

Figure 14: Membership Function Fuzzy KNN

were *nnc* are the number of instances that belong to class *c* found between the *kInit* (usually a value between 3 and 9) neighbors of *x*. This works because we are setting the class of an instance to which originally belonged with more than half of the membership (0.51), on the other hand, the rest is divided between the rest of the other classes depending on the neighbourhood of the instance.

Using the Euclidean distance in this method is the main choice when the attributes are normalized in the [0, 1] range.

Later for the final vote for each class and neighbor we use the next equation:

$$V(k_j, c) = \frac{Uc(kj) * 1 / (\left|\left|Q - kj\right|\right|)^{2/(m-1)}}{\sum_{i=1}^{k} 1/(\left|\left|Q - ki\right|\right|)^{2/(m-1)}}$$

m being a parameter generally 2 and m > 1 and *kj* being the j-th nearest neighbor, Q is the instance to classify and $U_c(k_j)$ is the membership value of the k_j neighbor to the class *c*.

The votes are obtained by taking into account the memberships of the k neighbors, the Euclidean norm and the memberships are weighted to get the final classification.

Although, the vote selection is pretty similar to the original KNN talking about the majority of votes, in the case of Fuzzy KNN, the use of this weight labeling approach and the weighted votes allows getting a more precise classification among the instances which are near the decision boundaries. Moreover, this method lets one example vote for more than one class at a time taking into account its neighborhood.

3.3. FUZZY K-NEAREST NEIGHBOR WITH INTERVALS

3.3.1. DESCRIPTION

Although, Fuzzy K-Nearest Neighbors is a great improvement in this KNN algorithm, it lacks in knowledge just taking into account one value between [0, 1] for the memberships' assignation, contrary to the possibility of giving an interval valued membership as the new model we are going to describe. This has direct relation with the two parameters: *m* for the voting process of the k nearest neighbors and *kInit* for the initial memberships process.

The solution for this problem is Fuzzy K-Nearest Neighbor with Intervals (IVFS KNN) [4], an improvement for the original Fuzzy algorithm. IVFS introduces fuzzy sets as memberships with a lower and upper bound. This interval is also used at the final moment of the voting process and gives us more flexibility as far as the selection of *kInit* and *m* parameters is concerned. We also should take in mind that the length of the interval, the difference between the upper bound and the

lower bound, gives us information about the degree of uncertainty the classification may have.

3.3.2. MEMBERSHIP PROCESS

The use of intervals for the memberships can help with the use of several values for *kInit* at the same time obtaining each membership with the next equation:

$$U_c(x, kInit) = \begin{cases} 0.51 + (nn_c/kInit) * 0.49 & \text{if } c = \omega \\ (nn_c/kInit) * 0.49 & \text{otherwise} \end{cases}$$

getting a membership value for each instance *x* and *kInit* value. So finally, the interval membership of an instance *x* of the training set to a class will be represented as an interval like this:

$$U_c(x) = [U_{c\text{-left}}(x), U_{c\text{-right}}(x)]$$

$$U_{c\text{-left}}(x) = \min[U_c(x, kInit)]$$

$$U_{c\text{-right}}(x) = \max[U_c(x, kInit)]$$

This way, a better and more accurate classification is obtained because of the aggregation of the different kInit's results so that the final election becomes less relevant and because of several important points:

- Training points identified incorrectly (noise) that only have instances of other classes nearby will get only half membership ([0.51, 0.51]), on the contrary, the membership to the other classes will descript the real nature of these training points.

- Training points detected at the center of their classes that are only surrounded by other instances of the same class will have all the membership ([1.0, 1.0]) and no membership ([0.0, 0.0]) to the rest of the other classes. This works the same way as original KNN and original Fuzzy KNN.

- Training points located over the boundaries and surrounded by different training instances that belong to different classes being some of them of the same class, but other instances belonging to other classes will have a special membership:

    - The lower value of the membership to c, min[U_c(x, kInit)] (U_{c-left}(x)), can be used for measuring how relevant and how many neighboring instances with a class different to c are. The higher the number of neighboring instances and closer to the training instance they will be, the closer to 0.51 this lower value will be so that this membership assures to be more than half because of its proximity to instances of the same class.

    - The upper value of the membership to c, $U_{c-right}$(x), can be used for measuring how far away the first neighbor not belonging to c is. It will be 1.0 if it is not among the first nearest neighbors (taking into account the parameter kInit chosen), and it would be a bit lower if the number and position of the neighboring instances not belonging to c is slightly bigger.

    - The lower value of the membership to the rest of classes will be 0.0, unless one of the first nearest neighbors belongs to that class. The upper value can be used too as a relative way of measuring the presence of this class among the neighborhood of the training instance, never greater than 0.49 to assure it doesn't get more than half of the membership due to its lack of proximity to instances of the same class.

3.3.3. VOTING PROCESS

The votes cast by each neighbor in the computation of the decision rule can also be represented by intervals. In this expression, the parameter m can be used to vary the influence of the neighbors, depending on the specific value chosen.

If m = 2, the vote of each neighbor is weighted by the reciprocal of the squared Manhattan distance. As m increases, distances between the different neighbors will be evenly weighted, and therefore the relative distances will have less effect on the determination of the votes (with m = 3 the weight becomes the reciprocal of the Euclidean distance). Similarly, if m is decreased, the relative distances will have a greater effect, reducing the contribution of the furthest instances (as m approaches 1).

Although the choice recommended was to simply let m = 2, it is possible to consider this parameter in a more flexible way, by introducing intervals. This allows a range of possible values of m to be considered instead of a single one, resulting in a more general voting mechanism.

To represent this, the equation becomes:

$$V(k_j, c) = U_c(k_j) * D(k_j)$$

Where

$$D(k_j) = [\min(D(k_j, m_a), D(k_j, m_b)),$$
$$\max(D(k_j, m_a), D(k_j, m_b))]$$

$$D(k_j, m) = \frac{1\ /(\left||Q-kj\right||)^{2/(m-1)}}{\sum_{i=1}^{k} 1/(\left||Q-ki\right||)^{2/(m-1)}}$$

and m_a, m_b are the minimum and maximum values chosen for the parameter m. Note that since the elements of $D(k_j)$ are intervals, their product must be computed as:

$$[a1, a2] * [b1, b2] = [a1*b1, a2*b2]$$

3.3.4. COMBINATION OF VOTES

Once the votes have been calculated, we obtain the final classification as the class with the highest vote value. In the case of IV-KNN, the votes to each class are calculated as follows:

$$V(c) = \sum_{j=1}^{k} V(kj, c)$$

where the addition of two intervals is obtained by

$$[a1, a2] + [b1, b2] = [a1+b1, a2+b2]$$

After all votes for each and every class have been computed, all the intervals are converted to a single value getting the center of the interval so the final classification is obtained as the class which has the highest center of interval. In the case of a tie, only the first nearest neighbor is considered to untie the classification.

3.4. EVOLUTIONARY KNN WITH INTERVALS

3.4.1. DESCRIPTION

As is detailed previously, the performance of the classifier is dependent on the selection of parameter values kInit, m_a and m_b parameters. Although these values could be searched and fixed experimentally, the proposal in this case is the use of evolutionary algorithms for optimizing this selection.

There are two optimizations needed, for parameters m_a and m_b, the optimization process needs to find two real values within a range. On the other hand, the selection process of a set of values for kInit can be more complicated.

Furthermore, as the goal is to obtain the optimal values for this representation of the configuration, an optimization algorithm has to be used. For this reason, genetic algorithms are a suitable option for this algorithm given its great capabilities for optimization problems and in this case, the well-known CHC algorithm is used to get an approach solution.

For this improvement of the KNN with intervals version, the equations for obtaining memberships for each instance and for the voting procedure are the same ones as for the KNN with intervals version one. Even though, with this new version the parameter values (kInit and m) used for classification in membership and voting procedures will be optimized with a genetic algorithm so the classification accuracy is higher.

3.4.2. REPRESENTATION

The representation has two parts, the part for selecting de kInit value that uses a binary codification where the maximum value for kInit is chosen and it is created an array with same number of genes as that maximum value; and on the other hand, the other part for choosing the m_a and m_b values using real codification which will be the last two genes of the final chromosome.

The idea is to define a process for choosing some specific values which don't have to be a whole interval like {1, 2, 3, 4, 5, 6, 7, 8, 9} for kInit but to create the possibility of just using some of the kInit values being non-correlative between them with a maximum value of 9 in this example [5].

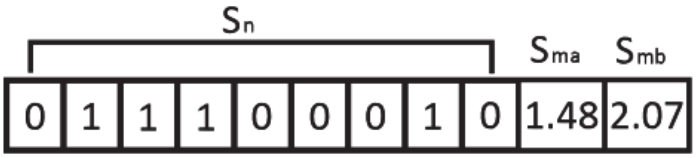


Figure 15: Valid configuration of Evolutionary KNN. The values of kInit chosen are 2, 3, 4 and 8 (from the interval (1, 9)) and the values of ma and mb are 1.48 and 2.07.

In Figure 19, we can observe an example of a valid configuration for the evolutionary KNN version. It is constructed by a binary array of S_n digits, in which a value $S_n = 1$ shows that n is chosen as a value for kInit, and on the other hand, a value $S_n = 0$ shows that n is not chosen, and finally two real values which are m_a and m_b.

In the example in Figure 19, the values 2, 3, 4 and 8 have been chosen for the kInit parameter as we see ones in their respective place of S_n, moreover, the interval [1.48,2.07] has been chosen for the *m* parameter. This configuration could be useful in such cases where the memberships should be constrained to the four nearest neighbors of each training instance, but then extended by considering an instance at a greater distance.

3.4.3. METHODS

The crossover methods used are the ones typically used in the CHC genetic algorithm.

There is a real coded part for the m values where the PCBLX crossover method will be applied. On the other hand, there is a binary coding for kInit parameters where HUX method will be applied.

As we know, we are not applying any mutation method for CHC. But we apply a similar method as the random selection method but choosing the odd individuals and the even ones.

Moreover, we are using a high elitism replacement method where we introduce the best individual of the population in the next generation population and where both parents and offspring compete for being in the next generation population.

The fitness function for these chromosomes will be the accuracy on the training set.

4. RESULTS

The aim of this project is to compare these four versions of the KNN algorithm and to observe the different accuracy they obtain. For this reason, these four algorithms have been tested with several known datasets.

4.1. DATASETS

The datasets used are some well-known Keel datasets [18] used a lot for testing machine learning models which appear with their respective number of instances or examples, their attributes or variables and the number of classes that the respective KNN algorithm version has to predict:

Table 1: Datasets considered in the study

Dataset	Instances	Attributes	Classes
appendicitis	96	7	2
balance	562	4	3
banana	4770	2	2
bands	334	19	2
bupa	310	6	2
cleveland	267	13	5
dermatology	323	34	6
ecoli	303	7	8
glass	196	9	7
haberman	276	3	2
hayes	144	4	3
heart	243	13	2
hepatitis	69	19	2
ionosphere	316	33	2
iris	135	4	3
led7digit	450	7	10
mammographic	744	5	2
marketing	6185	13	9
monk	388	6	2
movement_libras	324	90	15
newthyroid	194	5	3
page	4925	10	5
penbased	9892	16	10
phoneme	4864	5	2
pima	691	8	2

ring	6660	20	2
satimage	5792	36	7
segment	2079	19	7
sonar	188	60	2
spambase	4138	57	2
spectfheart	241	44	2
tae	136	5	3
texture	4950	40	11
thyroid	6480	21	3
titanic	1981	3	2
twonorm	6660	20	2
vehicle	762	18	4
vowel	891	13	11
wdbc	513	30	2
wine	160	13	3
winequality	1440	11	11
winequality	4409	11	11
wisconsin	616	9	2
yeast	1336	8	10

There are 44 datasets with many different numbers of examples, attributes and classes per dataset. Here there is a simple description of mean, median and other interesting data about the datasets:

	Instances	**Attributes**	**Classes**
count	44.0	44.0	44.0
mean	1964.39	17.73	4.73
std	2548.15	17.96	3.62
min	69.0	2.0	2.0
25%	261.0	6.0	2.0
50%	537.5	12.0	3.0
75%	4205.75	20.0	7.0
max	9892.0	90.0	15.0

4.2. PERFORMANCE METRIC AND CONFIGURATIONS

After having explained where our models are going to be tested, an evaluation method needs to be described. The accuracy score metric is used for testing each dataset and getting its results. Moreover, for this approach, a 10 K cross-validation method is used, so the final result for each dataset will be the mean of the 10 different accuracies we get for each of the groups of validations there are.

The configuration for the Fuzzy K-Nearest Neighbor will be: kInit = 3 and m = 2. On the other hand, the Interval-Valued Fuzzy KNN will have [4, 5, 6, 7, 8] as the interval for the kInit parameter and finally, m_a will be 1.5 and m_b will be 2. The original KNN doesn't use kInit or m parameters, so for all the models a comparison for different k values, which is the parameter all models have in common, will be done for searching the best k values for future comparison with the evolutionary algorithm.

4.3. K VALUE COMPARISON

All the four models have a common parameter k, the k nearest neighbors' number, which hasn't been selected yet. For these first three models, a search for an optimal value for this parameter has been made.

We can observe Table 2, 3, 4 and 5 which show an analysis of the first three version of KNN for different k values where each column is a different model and each row is a dataset. Each column contains the accuracy of each model for a specific dataset. Moreover, we can observe the second to last row in which the average accuracy for each model for all the 44 datasets is calculated and in the last row we can see the number of datasets in which each model has achieved the best accuracy value.

For Table 2 and k = 3, there is an improved accuracy for the FKNN and the intervals models with no difference between these two while the original KNN gives the worst result. Moreover, taking into account the number of wins in the last row there isn't any special improvement as we can see that the original KNN and the IVFS model finish with a tie with more wins than the Fuzzy model.

Table 2: Analysis of different KNN algorithms for k = 3

Dataset	KNN	Fuzzy	Iv Fuzzy
appendicitis	83.36	83.36	81.45
balance	81.27	81.6	81.44
banana	88.6	88.62	88.13
bands	69.35	70.13	72.05
bupa	59.63	61.7	61.99
cleveland	54.54	54.54	54.27
dermatology	96.9	96.9	96.9
ecoli	80.97	82.49	82.48
glass	71.95	74.29	73.18
haberman	70.27	69.3	67.33
hayes-roth	55.62	65.62	66.25
heart	77.41	77.04	77.04
hepatitis	81.99	81.99	81.99
ionosphere	85.18	85.46	85.46
iris	94.0	94.0	94.0
led7digit	69.2	69.0	69.0
mammographic	79.28	79.01	78.77
marketing	29.41	29.78	29.32
monk-2	96.55	79.96	78.59
movement_libras	83.06	85.56	86.39
newthyroid	95.37	96.32	97.23
page-blocks	95.91	96.02	96.0
penbased	99.36	99.36	99.38
phoneme	88.64	90.08	90.43
pima	73.06	72.93	72.79
ring	71.77	67.58	71.2
satimage	90.83	90.63	90.85
segment	96.02	96.23	96.75
sonar	83.07	84.5	85.0
spambase	89.62	91.04	91.08
spectfheart	72.68	72.32	72.68
tae	53.75	67.0	65.71
texture	98.75	98.8	98.91
thyroid	93.97	93.88	93.79
titanic	70.1	70.1	70.1
twonorm	96.54	96.58	96.51
vehicle	71.51	71.28	71.76
vowel	97.88	98.59	98.69
wdbc	96.13	96.3	96.12
wine	95.52	95.52	95.52
winequality-red	58.85	66.79	66.23
winequality-white	58.78	66.5	66.52
wisconsin	96.52	96.96	97.25
yeast	54.99	56.94	56.0
Average	79.73	80.51	80.51
Number of wins	16.0	12.0	16.0

For Table 3 and k = 5, there is an improved accuracy for the FKNN and IVFS models also with no much difference between them while the original KNN gives the worst result again. In this case, in terms of number of wins there is a clear difference because the IVFS model wins 19 of the 44 datasets while the first two models share about the other half of the wins almost equally.

Table 3: Analysis of different KNN algorithms for k = 5

Dataset	KNN	Fuzzy	Iv Fuzzy
appendicitis	85.0	87.0	85.09
balance	83.2	87.2	87.2
banana	89.09	89.38	88.98
bands	68.86	70.0	70.71
bupa	61.06	63.17	62.31
cleveland	56.31	56.64	57.0
dermatology	96.33	96.35	96.62
ecoli	80.99	82.77	83.38
glass	70.18	71.66	71.66
haberman	67.92	67.32	66.01
hayes-roth	44.38	65.62	65.62
heart	80.74	79.63	80.37
hepatitis	86.27	83.42	83.42
ionosphere	85.17	84.33	84.6
iris	96.0	95.33	95.33
led7digit	68.4	70.4	70.4
mammographic	80.85	79.65	79.75
marketing	29.95	30.46	30.61
monk-2	94.98	89.43	81.33
movement_libras	80.56	82.22	83.89
newthyroid	93.98	93.98	95.41
page-blocks	95.8	95.96	96.13
penbased	99.25	99.23	99.3
phoneme	87.99	89.64	90.04
pima	73.06	72.66	73.06
ring	69.2	60.47	62.96
satimage	90.78	90.26	90.44
segment	95.63	96.28	96.49
sonar	84.52	83.57	85.02
spambase	90.06	90.97	91.47
spectfheart	73.45	75.73	73.83
tae	55.04	66.29	66.38
texture	98.53	98.49	98.67
thyroid	94.0	93.97	94.07
titanic	74.28	74.28	74.28
twonorm	96.99	96.95	96.92
vehicle	72.1	70.69	71.16
vowel	95.15	96.87	98.28
wdbc	96.65	96.65	96.83
wine	95.49	96.05	95.49

winequality-red	59.66	68.48	68.35
winequality-white	57.57	67.58	67.52
wisconsin	96.95	97.11	97.39
yeast	57.01	59.23	58.56
Average	79.75	80.98	80.96
Number of wins	13.0	12.0	19.0

For Table 4 and k = 7, we start to observe a clear improvement in the average accuracy of the IVFS model with more than 81% average accuracy and a slightly worse result for the Fuzzy model compared with k = 5 even though it still improves original KNN's average accuracy. In this case, there is a clear winner too in terms of number of wins with IVFS model winning 21 out of the 44 datasets, that is nearly 50% of wins for all the datasets while the first two models share about the other half of the wins with a greater number of wins for part of the original model of KNN.

Table 4: Analysis of different KNN algorithms for k = 7

Dataset	KNN	Fuzzy	Iv Fuzzy
appendicitis	87.0	87.0	87.0
balance	88.32	88.65	88.49
banana	89.55	89.7	89.62
bands	70.32	69.58	70.84
bupa	61.95	64.6	65.21
cleveland	56.99	57.62	57.31
dermatology	96.34	96.06	96.06
ecoli	82.77	82.76	82.17
glass	69.79	70.36	71.77
haberman	70.23	70.92	68.29
hayes-roth	34.38	60.0	64.38
heart	78.89	78.89	80.0
hepatitis	86.77	88.76	88.67
ionosphere	84.03	84.31	84.6
iris	96.67	95.33	94.67
led7digit	68.6	69.0	69.6
mammographic	80.25	79.52	79.75
marketing	30.21	30.86	31.01
monk-2	89.4	83.23	81.64
movement_libras	75.56	76.11	81.67
newthyroid	92.58	92.58	93.98

page-blocks	95.54	95.69	96.02
penbased	99.12	99.1	99.16
phoneme	87.71	89.16	90.06
pima	72.92	73.71	73.19
ring	67.47	56.23	58.91
satimage	90.57	89.93	90.09
segment	95.11	95.93	96.19
sonar	81.17	77.83	83.12
spambase	89.78	90.76	91.28
spectfheart	77.21	78.68	77.17
tae	51.08	65.0	65.0
texture	98.25	98.25	98.42
thyroid	93.97	93.81	93.97
titanic	73.74	73.74	73.74
twonorm	97.11	97.03	97.05
vehicle	72.57	70.69	71.16
vowel	90.0	94.14	97.47
wdbc	97.18	97.01	97.18
wine	94.93	94.93	94.93
winequality-red	59.48	67.73	69.23
winequality-white	57.02	66.58	68.58
wisconsin	97.1	97.25	96.81
yeast	58.22	59.71	59.77
Average	79.26	80.42	81.02
Number of wins	14.0	9.0	21.0

For Table 5 and k = 9, we can observe a worse result for all the models compared with other almost all the other k values. IVFS is still the best one followed by the Fuzzy one with the worst model being again KNN, but with a decrease in average accuracy in all of them. On the other hand, in this last comparison, there is also a clear winner too in terms of number of wins with IVFS model winning 24 out of the 44 datasets, that is more than 50% of the wins for all the 44 datasets while the first KNN model gets 18 wins which is also great and finally we have the worst result for the Fuzzy model with just 2 wins for all the datasets.

Table 5: Analysis of different KNN algorithms for k = 9

Dataset	KNN	Fuzzy	Iv Fuzzy
appendicitis	87.0	87.0	87.0
balance	89.29	88.8	88.8
banana	89.89	89.77	89.7
bands	69.67	69.84	70.21
bupa	62.72	65.37	65.99

cleveland	55.28	57.92	57.97
dermatology	95.5	95.5	95.79
ecoli	82.75	82.17	83.06
glass	69.0	67.05	70.33
haberman	71.24	71.58	71.26
hayes-roth	30.0	47.5	61.88
heart	80.37	80.0	81.11
hepatitis	85.52	84.09	84.32
ionosphere	84.32	83.75	84.31
iris	95.33	94.67	94.0
led7digit	70.0	70.8	71.6
mammographic	80.49	80.24	79.52
marketing	30.64	31.16	30.94
monk-2	82.0	79.02	79.96
movement_libras	71.11	71.67	78.33
newthyroid	92.55	92.55	92.55
page-blocks	95.49	95.43	95.69
penbased	99.04	99.0	99.13
phoneme	87.16	88.71	89.56
pima	73.58	73.32	73.72
ring	65.89	53.88	55.82
satimage	90.29	89.57	90.05
segment	94.98	95.28	95.93
sonar	74.95	71.64	75.45
spambase	89.84	90.21	90.99
spectfheart	76.79	77.54	77.56
tae	54.33	62.38	65.67
texture	98.16	98.02	98.27
thyroid	94.03	93.72	93.82
titanic	76.69	76.69	76.69
twonorm	97.19	97.15	97.19
vehicle	70.8	68.68	70.8
vowel	83.74	90.81	96.57
wdbc	97.18	97.01	97.01
wine	94.93	94.38	94.93
winequality-red	59.23	66.23	68.35
winequality-white	56.37	65.01	67.62
wisconsin	97.1	96.66	96.81
yeast	59.03	59.78	60.18
Average	78.66	79.35	80.60
Number of wins	18.0	2.0	24.0

4.4. FINAL COMPARISON

As observed in Tables 2, 3, 4 and 5, original KNN algorithm has its best performance with k = 5 with an average accuracy in the 44 datasets of 79.75%. Secondly, the improved Fuzzy KNN version performed with an 80.98% of average accuracy in the 44 datasets with k = 5 too. Finally, the Interval-Valued Fuzzy KNN version had the best performance with 81.02% average accuracy in the 44 datasets using k = 7.

The objective of this project was to study the supposed improvement of the other KNN algorithms. The results are clear, the Fuzzy KNN model makes an improvement in the average accuracy of its predictions comparing it with the original KNN. Moreover, the Interval-Valued Fuzzy version outperformed not only the original version, but the Fuzzy version one too with a final accuracy of 81.02%. With this study, it is sure that the fuzzy and interval fuzzy theory can be used for making the K-Nearest Neighbor algorithm more accurate.

The best configurations for the k parameter previously calculated will be used for the final comparison with the evolutionary KNN model.

The evolutionary model will use a configuration of: a k neighbors value of 7 because of the previous study that gave the best result for the previous model IVFS with a k value of 7, the kInit possible values will be within the interval [0, 9], the 2 m possible values will be within (1, 4] and the population size used in the genetic algorithm for the parameters search will be 50.

Table 6 has the same structure of the previous tables in terms of how rows and columns are presented which the addition of a new column for the Evolutionary model.

In Table 6 we can observe the improvement of each model in the average row. The Evolutionary model outperforms the IVFS models with an average accuracy for the 44 datasets of 81.43%. The accuracies for the other models decrease the closer we get to the original KNN. In terms of number of wins, there is a new clear winner which is also the Evolutionary model with 19 wins out of the 44 datasets, followed by the past winner IVFS with 13 wins. These two models really outperformed KNN and FKNN models in this last row with more than double of wins.

Table 6: Accuracy comparison between different KNN algorithms

Dataset	KNN	Fuzzy	Iv Fuzzy	Evolutionary
appendicitis	85.0	87.0	87.0	84.85
balance	83.2	87.2	88.49	88.36
banana	89.09	89.38	89.62	89.37
bands	68.86	70.0	70.84	67.12
bupa	61.06	63.17	65.21	62.17
cleveland	56.31	56.64	57.31	57.68
dermatology	96.33	96.35	96.06	95.32
ecoli	80.99	82.77	82.17	82.35
glass	70.18	71.66	71.77	67.51
haberman	67.92	67.32	68.29	70.31
hayes-roth	44.38	65.62	64.38	65.62
heart	80.74	79.63	80.0	77.78
hepatitis	86.27	83.42	88.67	86.11
ionosphere	85.17	84.33	84.6	92.46
iris	96.0	95.33	94.67	95.56
led7digit	68.4	70.4	69.6	70.25
mammographic	80.85	79.65	79.75	82.97
marketing	29.95	30.46	31.01	30.77
monk-2	94.98	89.43	81.64	87.12
movement_libras	80.56	82.22	81.67	83.33
newthyroid	93.98	93.98	93.98	95.45
page-blocks	95.8	95.96	96.02	96.29
penbased	99.25	99.23	99.16	99.55
phoneme	87.99	89.64	90.06	91.07
pima	73.06	72.66	73.19	73.31
ring	69.2	60.47	58.91	64.66
satimage	90.78	90.26	90.09	90.89
segment	95.63	96.28	96.19	95.96
sonar	84.52	83.57	83.12	87.3
spambase	90.06	90.97	91.28	90.94
spectfheart	73.45	75.73	77.17	78.17
tae	55.04	66.29	65.0	65.42
texture	98.53	98.49	98.42	98.79
thyroid	94.0	93.97	93.97	93.7
titanic	74.28	74.28	73.74	74.62
twonorm	96.99	96.95	97.05	96.35
vehicle	72.1	70.69	71.16	69.46
vowel	95.15	96.87	97.47	99.33
wdbc	96.65	96.65	97.18	99.42
wine	95.49	96.05	94.93	96.3
winequality-red	59.66	68.48	69.23	67.42
winequality-	57.57	67.58	68.58	66.63
wisconsin	96.95	97.11	96.81	96.6
yeast	57.01	59.23	59.77	58.17
Average	79.75	80.98	81.02	81.43
Number of wins	6.0	7.0	13.0	19.0

Table 7: Time comparison between different KNN algorithms

Dataset	KNN	Fuzzy	Iv Fuzzy	Evolutionary
appendicitis	0.05	0.07	0.09	121.78
balance	0.06	0.31	0.36	736.06
banana	0.25	2.66	3.89	6212.18
bands	0.06	0.32	0.24	397.64
bupa	0.06	0.15	0.21	369.68
cleveland	0.07	0.24	0.26	384.75
dermatology	0.06	0.34	0.36	536.29
ecoli	0.04	0.34	0.37	511.72
glass	0.04	0.2	0.25	303.71
haberman	0.04	0.12	0.25	319.62
hayes-roth	0.05	0.1	0.12	178.02
heart	0.05	0.12	0.18	291.3
hepatitis	0.04	0.06	0.07	85.03
ionosphere	0.05	0.17	0.23	401.06
iris	0.04	0.09	0.12	171.33
led7digit	0.06	0.59	1.04	880.86
mammographic	0.06	0.29	0.87	903.4
marketing	1.56	17.3	17.82	18710.62
monk-2	0.05	0.18	0.26	519.31
movement_libras	0.08	0.76	0.78	887.41
newthyroid	0.04	0.13	0.16	269.25
page-blocks	0.48	7.52	10.16	10386.99
penbased	2.74	37.47	38.96	38635.97
phoneme	0.29	4.33	6.25	7182.49
pima	0.05	0.6	0.44	889.22
ring	1.76	16.01	22.04	19148.53
satimage	1.35	13.14	18.69	147130.55
segment	0.2	3.56	3.98	4915.25
sonar	0.03	0.11	0.14	274.58
spambase	2.29	12.5	16.31	13631.15
spectfheart	0.08	0.12	0.17	338.2
tae	0.06	0.09	0.11	193.64
texture	1.18	12.62	15.87	24289.39
thyroid	2.5	15.5	20.96	17330.22
titanic	0.11	0.8	1.29	2617.3
twonorm	1.9	17.11	22.75	21191.05
vehicle	0.08	0.57	0.71	1233.04
vowel	0.12	1.25	2.06	2007.56
wdbc	0.06	0.28	0.43	745.89
wine	0.03	0.1	0.15	228.91
winequality-red	0.12	2.84	3.64	3251.1
winequality-	1.46	11.41	18.47	13037.09
wisconsin	0.12	0.42	0.48	844.18
yeast	0.17	1.93	3.03	3182.54
Average	0.45	4.2	5.34	8315.36

In Table 7 the structure changes, the four columns are the same four different models, but the rows are no longer accuracies but the time of execution for each dataset and model. The last row is the average time it took each model for the execution of each dataset.

We can clearly observe in Table 7 that the time execution for these four models is very important to take into account. The first three models have a really low time of average execution with less than 6 seconds per dataset all of them. On the other hand, the Evolutionary model takes more than a thousand times more to finish its execution than the previous models.

Figure 18 is a bar graphic comparing for all the 44 datasets the number of instances and the time of execution for the Evolutionary model. As we can observe, when the number of instances increases the time of execution also grows.

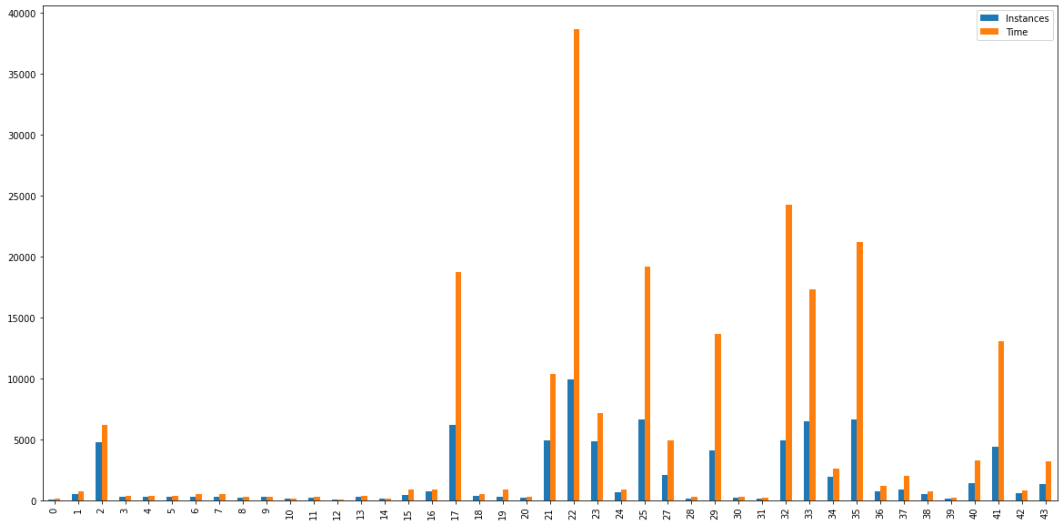


Figure 16: Bar figure representing number of instances and time of execution for the Evolutionary model for all the 44 datasets

Moreover, Figure 19 shows for all the 44 datasets the number of instances in axis x and the time of execution for the Evolutionary model in axis y. It clearly shows the relation between the number of instances and the time of execution required. When using the genetic algorithm each individual has to go over all the training instances each iteration, so when the instances increase it makes the same effect in the time needed for execution, while the number of classes or the number of attributes of the datasets doesn't really affect it.

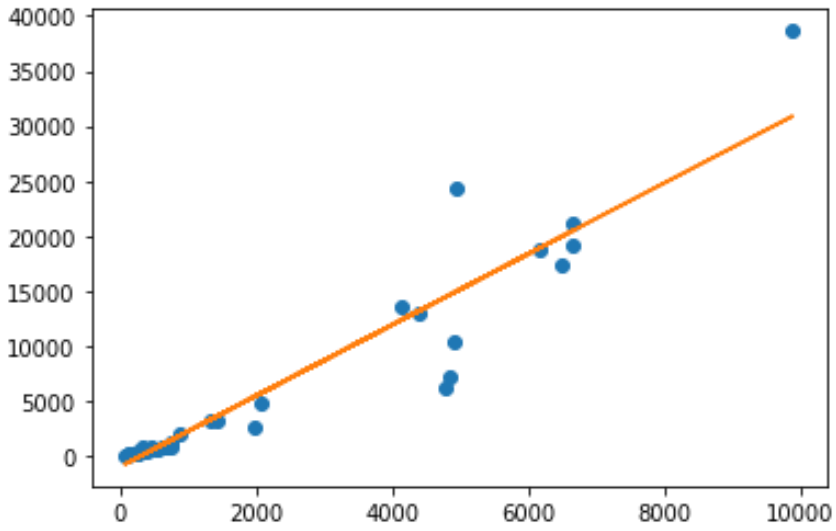


Figure 17: Scatter figure with regression line representing the number of instances and time of execution for the Evolutionary model in all the 44 datasets

This final study in Tables 6 and 7 presents clearly how each version of the KNN algorithm improves the previous one. That is a good sign and demonstrates the power of the interval fuzzy theory and genetic algorithms. The evolutionary version finished with the best average accuracy of all the datasets. It searched the proper configuration for the parameters for each dataset using the CHC genetic algorithm and then, uses that configuration to predict the test set and obtain its final accuracy. The other fact to take into account that is the enormous time we have seen the Evolutionary model takes to execute which really gets you thinking if this last model is really worth it.

5. RESULTS AND FUTURE LINES

It has been demonstrated that there are techniques to use that can be applied to simple algorithms such as K-Nearest Neighbor to improve its ability to predict. In this case, these techniques were fuzzy theory, interval fuzzy theory and genetic algorithms.

The ability to create memberships for the instances with interval fuzzy theory has really helped improving the performance of the algorithm without taking a lot of time to train these models as observed in Table 7. On the other hand, genetic algorithms, specifically CHC in our study, has also outperformed all the other presented versions of KNN including the Interval Fuzzy one. Unfortunately, the amount of time it takes to execute the Evolutionary KNN with Intervals is incredibly huge. Is this time really worth it taking into account the little improvement (in comparison with the time) it has?

In my opinion, if there is an important task to carry out and there is enough hardware available to improve the time to execute the evolutionary version of KNN can really be helpful. Otherwise, the use of the Interval-Valued Fuzzy algorithm will be adequate due to the good performance it has too and the low amount of time it takes to execute in comparison with the evolutionary one.

In the future, we expect to continue studying more techniques that can improve the prediction of different models within classification and moreover the machine learning area. The idea is to find new methods to improve accuracy and also increasing the speed of execution. An aspect to take into consideration would be using better hardware for being able to use this accurate evolutionary version of KNN making a huge improvement in speed too. Other possible methods could be using other type of genetic algorithms for the evolutionary model or trying to modify the parameters of the algorithm such as number of individuals or selection, crossover and replacement methods of the evolutionary population.

6. BIBLIOGRAPHY

[1] Judith Hurwitz and Daniel Kirsch. *Machine Learning for dummies.* IBM, 2018.

[2] Michael Steinbach and Pang-Ning Tan, K-Nearest Neighbors, 2009, 151-159.

[3] James M. Keller, Michael R. Gray, And James A. Givens, Jr. A Fuzzy K-Nearest Neighbor Algorithm. IEEE Transactions on Systems, Man, And Cybernetics, Vol. SMC-15, nº. 4, 1985. 580-585.

[4] Joaquín Derrac, Francisco Chiclana, Salvador García, Francisco Herrera. An Interval Valued K-Nearest Neighbors Classifier. 2015. 378-384.

[5] Joaquín Derrac, Francisco Chiclana, Salvador García, Francisco Herrera. Evolutionary fuzzy k-nearest neighbors algorithm using interval-valued fuzzy sets. 2015. 144-163.

[6] Ethem Alpaydin, Introduction to Machine Learning (Fourth ed.). MIT. pp. xix, 2020, 1–3, 13–18.

[7] Osisanwo F.Y., Akinsola J.E.T., Awodele O., Hinmikaiye J. O., Olakanmi O., Akinjobi J. International Journal of Computer Trends and Technology (IJCTT) – 2017, Vol. 48, 128-138.

[8] Choi RY, Coyner AS, Kalpathy-Cramer J, Chiang MF, Campbell JP. Introduction to machine learning, neural networks, and deep learning. Trans Vis Sci Tech. 2020; 9(2): 14.

[9] Frederick Livingston. Implementation of Breiman's Random Forest Machine Learning Algorithm, 2005.

[10] Hu, J.; Niu, H., Carrasco, J., Lennox, B.; Arvin, F., Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning, IEEE Transactions on Vehicular Technology, 2020, Vol. 69, Nº. 12, 14413-14423.

[11] Akalin, N., Loutfi, A., Reinforcement Learning Approaches in Social Robotics, 2021.

[12] Ke Tang, Rui Wang, Tianshi Chen, Towards Maximizing the Area Under the ROC Curve for Multi-Class Classification Problems, 2011, 483-488.

[13] José Antonio Sanz, Alberto Fernández, Humberto Bustince, Francisco Herrera. Improving the Performance of Fuzzy Rule-Based Classification Systems with Interval-Valued Fuzzy Sets and Genetic Amplitude Tuning. 2013.

[14] Didier Dubois, Henri Prade. Institut de Recherche en Informatique de Toulouse, France. Interval-valued Fuzzy Sets, Possibility Theory and Imprecise Probability, 2005, 314-319

[15] Mikel Galar Idoate, José Antonio Sanz. Computación Evolutiva y Algoritmos Bioinspirados, Smart Cities Department, UPNA.

[16] Larry J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination, Foundations of Genetic Algorithms, Elsevier, Volume 1, 1991. 265-283.

[17] T. M. Cover, Member, IEEE, and P. E. Hart, Member, IEEE. Nearest Neighbor Pattern Classification, 1967, 21-27.

[18] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. Journal of Multiple-Valued Logic and Soft Computing 17:2-3 (2011) 255-287.