Predicting non-alcoholic fatty liver disease using classification algorithms

1st Nguyen Vu Duong
Computer Science
Univeristy of Information Technology - VNUHCM
Ho Chi Minh City, Viet Nam
20520465@gm.uit.edu.vn

2nd Pham Phuoc An
Computer Science
Univeristy of Information Technology - VNUHCM
Ho Chi Minh City, Viet Nam
20520375@gm.uit.edu.vn

Abstract

Knowing who is going to be a patient at high risk for non-alcoholic fatty liver disease is crucial in designing the right treatment for each individual. By implementing classification algorithms into NAFLD data, we can determine who is at high risk or not, treatments will be applied promptly to improve patient's health. The main aim of this study was to apply machine learning (ML) methods to identify significant classifiers of NAFLD. This study focused on the machine learning (K-nearest neighbors and Decision Tree) on the NAFLD dataset include a community cohort of all adults diagnosed with NAFLD in Olmsted County, MN between 1997–2014 was constructed using the Rochester Epidemiology Project database. The final results are compared and evaluated to find the most effective model based on different evaluation criteria. The experimental results show that Decision Tree algorithm was better than K-nearest neighbors algorithm in terms of the accuracy, and precision measurement, with 74.2%, and 82.6% on the death classification task, respectively. However, the K-nearest neighbors algorithm's recall is higher than the Decision Tree's about 5.2%.

I. Introduction

A. What is NAFLD ?

Nonalcoholic fatty liver disease (NAFLD) is an umbrella term for a range of liver conditions affecting people who drink little to no alcohol. As the name implies, the main characteristic of NAFLD is too much fat stored in liver cells. NAFLD is increasingly common around the world, especially in Western nations. In the United States, it is the most common form of chronic liver disease, affecting about one-quarter of the population. Some individuals with NAFLD can develop nonalcoholic steatohepatitis (NASH), an aggressive form of fatty liver disease, which is marked by liver inflammation and may progress to advanced scarring (cirrhosis) and liver failure. This damage is similar to the damage caused by heavy alcohol use. The primary purpose of this research is to create a machine learning model that can predict if a person is at risk of non-alcoholic fatty liver disease or not based on some elements of anthropometry. We are going to use two classification algorithms including K-nearest neighbors and Decision Tree for this research. Pham Phuoc An will do K-nearest neighbors algorithm and Nguyen Vu Duong will take responsibility for Decision Tree algorithm.

B. Purpose of choosing model

We have chosen two models, Decision Tree and K-Nearest Neighbors (K-NN), for the following reasons:

- When observing the dataset, we noticed a correlation between the input data and the labels. Therefore, we selected the Decision Tree model, as it is known for its ability to capture and utilize such relationships in the data.
- We chose K-NN because it is a lazy learning model, meaning it relies on the distances between neighbors to make predictions. We believe that this approach could be beneficial in our case, as it can potentially capture local patterns and similarities within the data.

The suitability and effectiveness of Decision Tree and K-NN models should be evaluated through experimentation and comparison to find which is a better model.Additionally, we mentioned our interest in understanding whether a lazy model can outperform a learning model.

C. Purpose of choosing hyperparameter tuning method

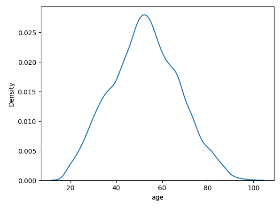
In this project, GridSearch is selected for tuning because it is straightforward, simple to comprehend, and simple to use. This approach ensures a thorough search and doesn't overlook any promising combinations by testing all possible combinations of the provided parameter values. Grid search is quick and efficient for small datasets. But when there are many parameters and parameter values, GridSearch might take a long time and require a lot of processing resources.

II. DATA

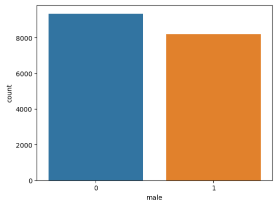
Dataset contains data from a population study on non-alcoholic fatty liver disease (NAFLD). The dataset includes individuals with this condition and a corresponding control group, who are monitored for metabolic status, cardiovascular criteria, and mortality. This dataset consists of 17,549 rows and 10 columns.

- id: Participant's study identification code
- age: Age of the participant
- male: 0 for female, 1 for male
- weight: Weight of the participant (in kilograms)
- height: Height of the participant (in centimeters)
- bmi: Body Mass Index (BMI)
- case.id: Identifier for the corresponding NAFLD case
- futime: Time until death or last follow-up
- status: 0 = alive at last follow-up, 1 = deceased

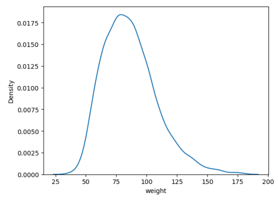
A. *Visualizing Dataset Makeup*



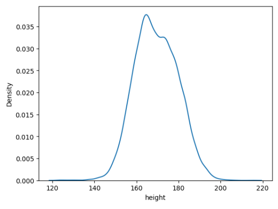
(a) Numeric of age



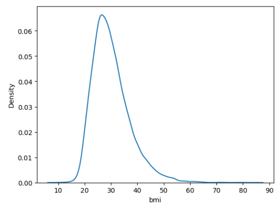
(b) Gender



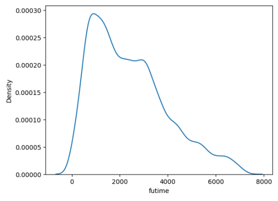
(c) Weight



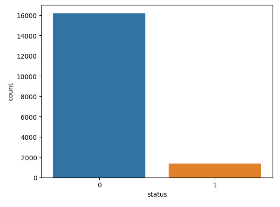
(d) Height



(e) Body Mass Index (BMI)



(f) Fuitime

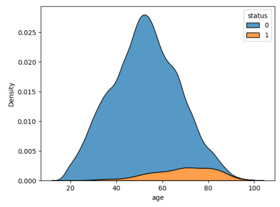


(g) Status

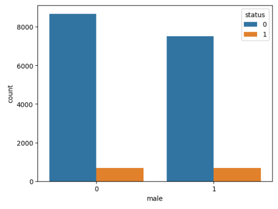
Fig. 1: Dataset Visualization

In general, the plot a, c, d and e in Fig. 1 exhibit a considerable similarity to a Gaussian distribution, which is commonly referred to as a normal distribution in the context of machine learning. Next, from the plot b of Fig. 1, we can see that the number of males and females does not differ significantly. However, there is a considerable disparity in the number of labels in plot g and it is undeniable that this dataset is imbalanced. That means we need to balance the data and the details will be discussed in the preprocessing section.

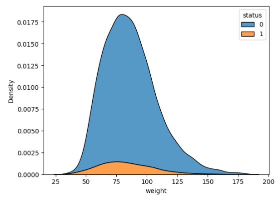
B. *Visualizing Correlations With Label*



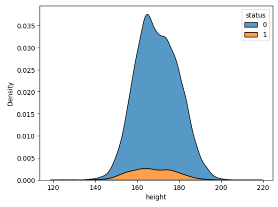
(a) Age



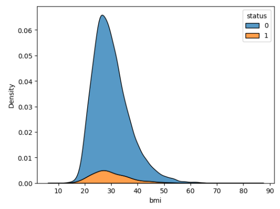
(b) Gender



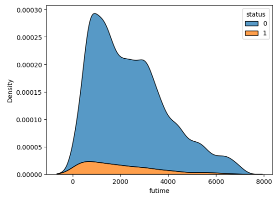
(c) Weight



(d) Height



(e) Body Mass Index (BMI)



(f) Fuitime

Fig. 2: Visualizing Correlations With Label

From Fig. 2, it can be seen that individuals at a higher risk of mortality are typically older and female, ranging from approximately 65 to 80 years of age, with a BMI index between 25 and 30. That BMI range also indicates a high level of overweight and severe obesity.

III. Method

A. *K-Nearest Neighbors*

1) *Definition*

K-Nearest Neighbors (K-NN) classifies a given based on the groups of its surrounding 'k' number of neighbors. It uses feature similarity to predict the cluster that the new point will fall into.

2) *Properties*

The properties of K-NN are that they are a lazy learning algorithm and a non-parametric method. In addition, it is used for both Classification and Regression.

a) Lazy learning

Lazy learning means the algorithm takes almost zero time to learn because it only stores the data of the training part (no learning of a function). The stored data will then be used for the evaluation of a new query point.

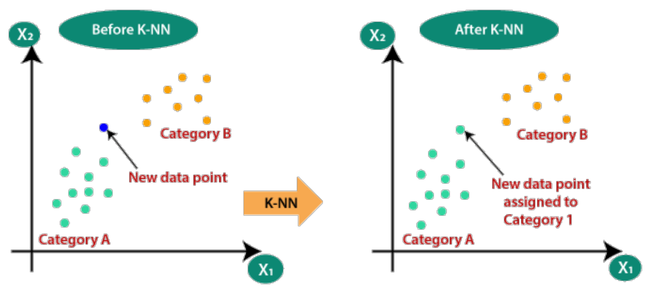


Fig. 3: Example of K-NN.

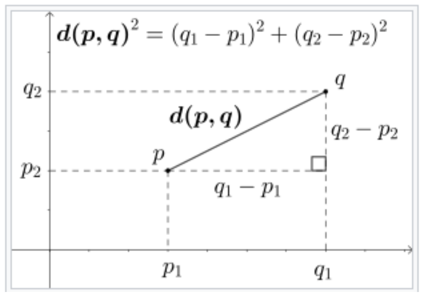


Fig. 4: Example of Euclidean Distance.

K-NN belongs to a subcategory of non-parametric models that is described as instance-based learning. Models based on instance-based learning are characterized by memorizing the training dataset, and lazy learning is a special case of instance-based learning that is associated with no (zero) cost during the learning process.

b) Non-parametric

The non-parametric method refers to a method that does not assume any distribution. Therefore, K-NN does not have to find any parameter for the distribution. While in the parametric method, the model finds new parameters, which in turn will be used for the prediction purpose. The only hyperparameter (provided by the user to the model) K-NN has is K, which is the number of points that needs to be considered for comparison purpose.

3) Metric for distance computation

There are many methods to measure distance such as Hamming Distance, Euclidean Distance, Manhattan Distance or Minkowski Distance, but the most popular one is the Euclidean distance (for smaller dimension data) and Manhattan distance.

a) Euclidean Distance

Euclidean Distance is defined as crow flies. It calculates the distance between two real-valued vectors. Euclidean distance is calculated as the square root of the sum of the squared differences between the two vectors.

$$\begin{aligned} d(p,q) &= d(q,p) \\ &= \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \ldots + (q_n-p_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \end{aligned} \tag{1}$$

In (1), q and p are two different points which have n dimensions

b) Manhattan

Manhattan Distance also called the Taxicab distance or the City Block distance, calculates the distance between two real-valued vectors. It is perhaps more useful to vectors that describe objects on a uniform grid, like a chessboard or city blocks. The taxicab name for the measure refers to the shortest path that a taxicab would take between city blocks (coordinates on the grid). Manhattan distance is calculated as the sum of the absolute differences between the two vectors.

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n} |q_i - p_i|} \tag{2}$$

In (2), q and p are two different points which have n dimensions

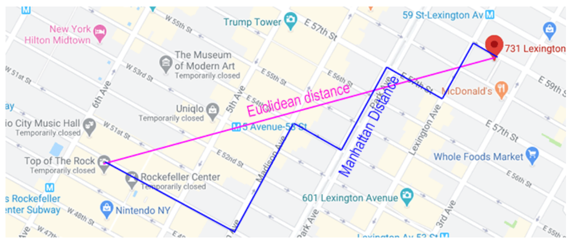


Fig. 5: Example of Manhattan distance

In Fig. 5, the pink line denotes the Manhattan distance between two points, whereas the blue poly line represents the Euclidean distance between those.

4) Process

In the training phase, the model will only store the data points.

In the testing phase, all the distance from the query point to the other points from the training phase is calculated to classify each point in the test dataset.

5) Pros and cons

a) Pros

- The algorithm is simple and easy to comprehend and implement.
- The training phase of K-nearest neighbor classification is much faster compared to other classification algorithms, as there is no need to train a model for generalization. This is why K-NN is known as the simple and instance-based learning algorithm.
- K-NN does provide a relatively high accuracy compared to other algorithms.
- K-NN can be useful in case of nonlinear data. It can be used with the regression problem. Output value for the object is computed by the average of k closest neighbors value.

b) Cons

- 'Time complexity' and 'space complexity' is enormous, which is a major disadvantage of K-NN.
  Time complexity refers to the time the model takes to evaluate the class of the query point, while space complexity refers to the total memory used by the algorithm. If we have n data points in training and each point is of m dimension. Then time complexity is of order O(n*m), which will be huge if we have higher dimension data. Therefore, K-NN is not suitable for high dimensional data.
- Another disadvantage is called 'The curse of dimensionality'.
  It is important to mention that K-NN is very susceptible to overfitting due to the curse of dimensionality. The curse of dimensionality describes the phenomenon where the feature space becomes increasingly sparse for an increasing number of dimensions of a fixed-size training dataset. Intuitively, we can think of even the closest neighbors being too far away in a high-dimensional space to give a good estimate.

6) Hyperparameters

K-Nearest Neighbors only has one hyperparameter 'k', which stands for the number of neighbors required for computation purpose. One thing to notice here, if the value of K is even, it might create problems when taking a majority vote because the data has an even number of classes. Therefore, choose K as an odd number when the data has an even number of classes and an even number when the data has an odd number of classes.

7) Tuning hyperparameter

For a better result, we employ Grid Search to obtain the optimal hyperparameter value. Grid search is a tuning technique that attempts to compute the optimum values of hyperparameters. It is an exhaustive search that is performed on the specific parameter values of a model. In the process, it will be searching all possible hyperparameter values, evaluating, memorizing, and then produce the best value for the model.

In K-Nearest Neighbors, the only hyperparameter is the 'k' number of neighbors, so we give Grid Search a range of 'k' values from 1 to 99 for input. In addition, Scikit-learn offers us another attribute for computing the metric distance as well as weights function. About the metric to use for distance computation, we use 'Euclidean' metric and 'Manhattan' metric to figure out the distance. We also use 'uniform' and 'distance' weights function for prediction.

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

After processing, the optimal values we have are:

- 'metric': 'mahattan'
- 'n_neighbors': 99
- 'weights': 'distance'

8) Protocol

- Step 1: Choose a value for K. K should be an odd number because we only have two classes, 0 or 1.
- Step 2: Find the distance of the new point to each of the training data.
- Step 3: Count the K nearest neighbors to the new data point.
- Step 4:
  For classification, count the number of data points in each category among the k neighbors. The new data point will belong to a class that has majority vote.
  For regression, the value for the new data point will be the average of the k neighbors.

B. Decision Tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data (training data). Decision tree algorithm is a data mining induction techniques that recursively partitions a dataset of records using depth-first greedy approach or breadth-first approach until all the data items belong to a particular class.

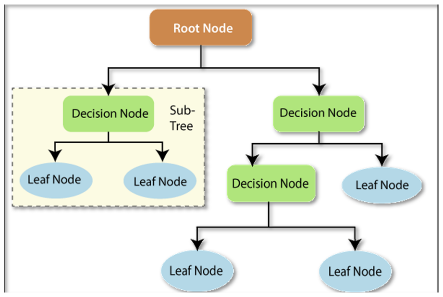


Fig. 6: Example of Decision Tree.

A decision tree structure is made of root, internal, and leaf nodes. It is a flow chart like tree structure, where every internal node denotes a test condition on an attribute, each branch represents the result of the test condition (rule), and each leaf node (or terminal node) shows an outcome (categorical or continues value). A root node is the parent of all nodes and as the name suggests it is the topmost node in the tree. As decision trees mimic the human level thinking, so it's so simple to grab the data and make some good interpretations. Decision tree is constructed in a divide and conquer approach. Each path in the decision tree forms a decision rule. Generally, it utilizes greedy approach from top to bottom.

1) Classification Tree

Decision tree classification technique is performed in two phases: tree building and tree pruning. Tree building is performed in top-down approach. During this phase, the tree is recursively partitioned till all the data items belong to the same class label. It is very computationally intensive as the training dataset is traversed repeatedly. Tree pruning is done in a bottom-up manner. It is used to improve the prediction and classification accuracy of the algorithm by minimizing over-fitting problem of tree. Over-fitting problem in decision tree results in misclassification error. Classification is the task of giving objects to categorize which have many diverse applications.

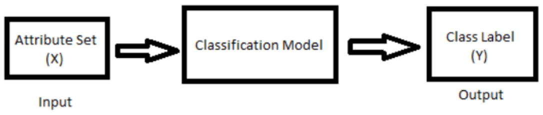


Fig. 7: Flowchart of Classification Tree.

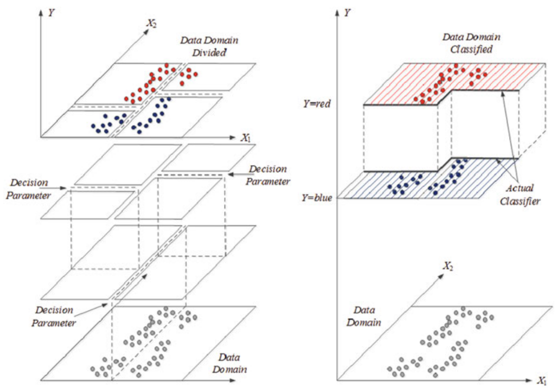


Fig. 8: The Classification Tree is illustrated in 3D using two classes with domain division properties. Response variable Y has two discrete values, red or blue.

2) *Decision Tree Algorithm*

Decision Tree algorithms are used to split the attributes to test at any node to determine whether splitting is "Best" in individual classes. The resulting partitioned at each branch is PURE as possible, for that splitting criteria must be identical.

There are several types of Decision Tree algorithms such as: *Iterative Dichotomies 3* (ID3); *Successor of ID3* (C4.5); *Classification And Regression Tree* (CART); *Chi-squared Automatic Interaction Detector* (CHAID); *Multivariate Adaptive Regression Splines* (MARS); *Generalized, Unbiased, Interaction Detection and Estimation* (GUIDE); *Conditional Inference Trees* (CTREE); *Classification Rule with Unbiased Interaction Selection and Estimation* (CRUISE); *Quick, Unbiased, Efficient, Statistical Tree* (QUEST). Table I showed comparison between the frequently used algorithms for the decision tree.

3) *Attribute selection measure*

While building a Decision tree, the main thing is to select the best attribute from the total features list of the dataset for the root node as well as for sub-nodes. The selection of best attributes is being achieved with the help of a technique known as the Attribute selection measure.

TABLE I: Comparison decision tree algorithms.

Algorithms name	Classification	Description
CART (Classification and Regression Trees)	Uses Gini Index as a metric.	By applying numeric splitting, we can construct the tree based on CART.
ID3 (Iterative Dichotomister 3)	Uses Entropy function and Information gain as metrics.	The only concern with the discrete values. Therefore, the continuous dataset must be classified within the discrete dataset.
C4.5	The improved version on ID3.	Deals with both discrete as well as a continuous dataset. Also, it can handle the incomplete datasets. The technique, called "PRUNNING", solves the problem of over filtering.
C5.0	Improved version of the C4.5.	C5.0 allows to whether estimate missing values as a function of other attributes or apportions the case statistically among the results.
CHAID (Chi-square Automatic Iteration Detector)	Predates the original ID3 implementation.	For a nominal scaled variable, this type of decision tree is used. The technique detects the dependent variable from the categorized variables of a dataset.
MARS (multi-adaptive regression splines)	Used to find the best split.	In order to achieve the best split, we can use the regression tree based on MARS.

a) *Entropy*

Entropy is a measure of the randomness in the information being processed. ID3 follows the rule — A branch with an entropy of zero is a leaf node, and A branch with entropy more than zero needs further splitting.

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

where S $\rightarrow$ Current state, and Pi $\rightarrow$ Probability of an event i of state S or Percentage of class i in a node of state S.

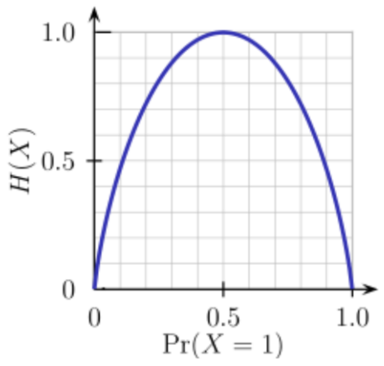


Fig. 9: Entropy graph.

b) Information Gain

Information gain is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values.

$$InformationGain = Entropy(bf) - \sum_{j=1}^{k} Entropy(j, at)$$

where *"bf"* is the dataset before the split, K is the number of subsets generated by the split, and *(j, at)* is subset j after the split.

c) Gini Index

Gini index is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions and easy to implement, whereas information gain favors smaller partitions with distinct values. Higher value of Gini index implies higher inequality, higher heterogeneity.

$$Gini = 1 - \sum_{i=1}^{c} (p_i)^2$$

d) Gain Ratio

Gain ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split. It corrects information gain by taking the intrinsic information of a split into account.

$$\begin{aligned} GainRatio &= \frac{InformationGain}{SplitInfo} \\ &= \frac{Entropy(before) - \sum_{j=1}^{k} Entropy(j, after)}{\sum_{j=1}^{k} log_2 w_j} \end{aligned} \tag{3}$$

4) Pros and cons

Decision Tree has some advantages and disadvantages

Advantages:

- Presentation is easy to understand (comprehensible).
- Can classify both categorical and numerical outcomes, but the attribute generated must be categorical.
- Can deal with noisy data.
- Quickly translated to a set of principle for production.

Disadvantages:

- It has long training time.
- Too many layers of decision tree make it extremely complex sometimes.
- It has a problem of over fitting.
- For more class labels, the computational complexity of the decision tree may increase.

5) Hyperparameters

In Decision Tree algorithms, Scikit-learn provides some parameters which support to build Decision Tree model.

a) ***criterion: "gini", "entropy", "log_loss", default="gini"***

The function *criterion* acts as a strategy to select the features used in the division of node. It measures the split quality at each node. There are three options which are "gini", "entropy" and "log_loss". "gini" criterion is commonly used for the Gini impurity and "log_loss", "entropy" both for the Shannon information gain. Computationally, entropy is more complex since it makes use of logarithms and consequently, the calculation of the Gini Index will be faster.

b) ***max_depth: int, default=None***

In most tree algorithms, height, or depth is an indispensable parameter. For Decision Trees, the *max_deep* parameter is defined by the Scikit-learn library as an integer representing the maximum depth of the tree at which the leaf node will be. In the process of applying the problem, depending on the purpose of use and depending on the data set of the problem, we will fine-tune this parameter differently, if choosing a low value will help the model predict faster but less accurate, but if you choose a high value, it will easily lead to overfitting and slow down.

c) ***min_samples_split: int or float, default=2***

The minimum sample size required to continue the division for the decision node. Used to avoid the size of the leaf node being too small to minimize overfitting.

d) ***min_samples_leaf: int or float, default=1***

Similar to *min_samples_split*, the *min_samples_leaf* parameter is the minimum number of leaf nodes required for a parent node to split when the tree has reached its final depth.

e) ***max_features: int, float or "auto", "sqrt", "log2", default=None***

The number of variables selected to find the best splitter at each split. The higher the *max_feature* value, the more accurate it is, but in return it takes more time.

f) ***max_leaf_nodes: int, default=None***

The maximum number of leaf nodes in a decision tree. Usually set up when control overfitting.

g) ***min_impurity_decrease: float, default=0.0***

The tree will not generate child nodes if the impurities are more than the threshold. With the node generation threshold being a real number. The *min_impurity_decrease* parameter is used to strike a balance between overfitting and precision. Low values bring high accuracy, the risk of overfitting also increases and vice versa.

h) ***ccp_alpha: non-negative float, default=0.0***

Subtrees of lower complexity are dropped. Used to strike a balance between overfitting and precision. Low values bring high accuracy, the risk of overfitting also increases and vice versa.

6) Tuning hyperparameters

The hyperparameter tuning methods relate to how we sample possible model architecture candidates from the space of possible hyperparameter values. This is often referred to as "searching" the hyperparameter space for the optimum values. We need to explore the hyperparameter space in hopes of locating the hyperparameter values which lead to the maximum score. There are many methods for tuning hyperparameters such as: grid search, random search, Bayesian optimization…

In this project, specifically Decision Tree model, we need to build a model which can solve the problem mentioned above with high accuracy and this model is not overfitting with training data. In order to do this, some hyperparameters to consider are:

- What criterion should I use?
- What should be the maximum allowable depth for each decision tree?
- How many sample sizes should be stop splitting?
- How many leaf nodes should be stop splitting?

To find the hyperparameters for issue above, we use Grid Search to discover the "best" hyperparameters which support the built model and avoid overfitting. With this technique, we simply build a model for each possible combination of all the hyperparameter values provided, evaluating each model, and selecting the architecture which produces the best results.

We will use some hyperparameters of decision tree model provided by Scikit-learn and set up following:

- 'criterion' = ('gini','entropy')
- 'max_depth' = np.arange(3,20)
- 'min_samples_split' = np.arange(3,20)
- 'min_samples_leaf' = np.arange(3,20)
- 'max_leaf_node' = [4,8,16,32]

After Grid Search, we have the best hyperparameters are 'criterion': 'gini', 'max_depth': 5, 'max_leaf_nodes': 8, 'min_samples_leaf': 14, 'min_samples_split': 3.

7) Protocol

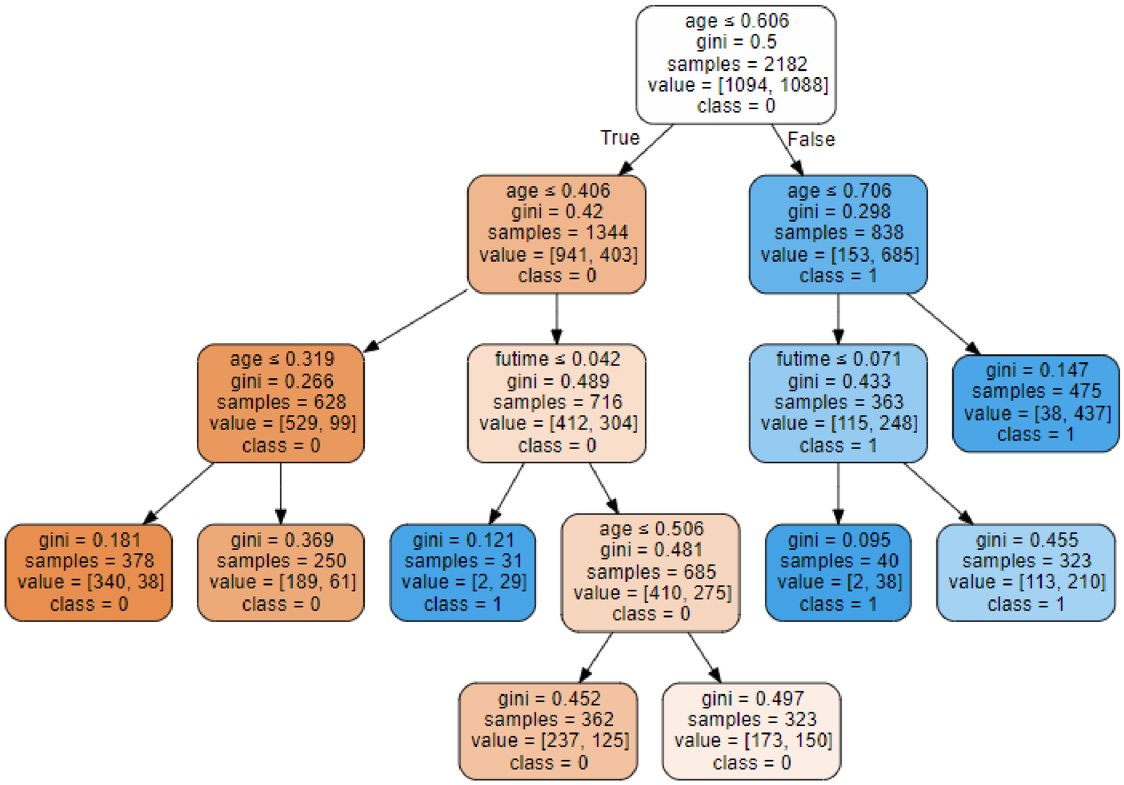


Fig. 10: Decision Tree for NAFLD dataset.

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contains possible values for the best attributes.
- Step-4: Generate the decision tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

IV. Experiments

A. Preprocessing

Experiments were conducted to better understand the operation mechanism of the two main algorithms: K-nearest neighbors and Decision Tree. As start above, we use NAFLD dataset with mission predicting high risk patients of non-alcoholic fatty liver disease. All two algorithms were implemented on Python frameworks. Before classification, we conducted data preprocessing for next steps. In this dataset, we only use 'age', 'male', 'weight', 'height', 'bmi' and 'futime' columns as training data and using the 'status' column as label data. We perform preprocessing through the following three tasks: Balancing the data, handling missing values (NaN) and normalizing the data.

1) Handling NaN values

In the dataset, there are features in which the missing values are the majority of the samples, such as 'height', 'weight' and 'bmi'. So, we have utilized the KNNImputer library from scikit-learn as an approach to handle that missing values (NaN).

- The KNNImputer library is designed to impute missing values in observations by identifying the nearest neighbors based on the Euclidean distance matrix. It fills in the missing values by considering the values of the nearest neighbors.
- Using KNNImputer can be beneficial in cases where missing values are randomly distributed across the dataset. By leveraging the relationships between the existing data points, KNNImputer estimates and replaces the missing values with values that are considered similar based on the neighboring observations.
- By employing the KNNImputer library, we can effectively handle NaN values in a dataset and ensure that the missing data is appropriately imputed for subsequent analysis or modeling tasks.

2) Balancing the data

A remarkable point in the NAFLD dataset, we found that the amount of data is imbalance. This can be demonstrated by the label data plot in Fig. 1 (g). So, we will resample data before training model.

The reason for addressing imbalanced datasets is to prevent the model from exhibiting bias towards the majority class, which typically has a larger number of samples. There are various approaches to handling imbalanced data, for example:

- Adding more data: Increasing the number of samples in the minority class can help improve its representation in the dataset.
- Changing performance metrics: Instead of relying solely on accuracy, using alternative metrics such as precision, recall, or F1 score can provide a more comprehensive evaluation of the model's performance.
- Penalized models: Applying penalties to the algorithm during training to prioritize the minority class can help mitigate the imbalance.
- Changing algorithms: Exploring different algorithms that are more robust to imbalanced data, such as ensemble methods or gradient boosting, can yield better results.
- Resampling: This technique involves either undersampling the majority class or oversampling the minority class to create a more balanced dataset. Undersampling reduces the number of observations from the majority class, while

In this particular case, we have chosen to use undersampling.

- Undersampling involves reducing the number of instances in the majority class to match the number of instances in the minority class. This approach quickly balances the dataset without requiring complex sampling algorithms. However, a drawback is that the sample size will be significantly reduced.
- The reason for selecting undersampling in this scenario is due to the sensitive nature of the medical field. It is crucial to have accurate and reliable data for making correct predictions about a patient's condition. Oversampling can introduce artificial data, which may raise ethical concerns and compromise accuracy. Therefore, undersampling is considered a more appropriate method, even though it may result in the loss of some important information.

Here is a chart describing the label data after it has been resampled:

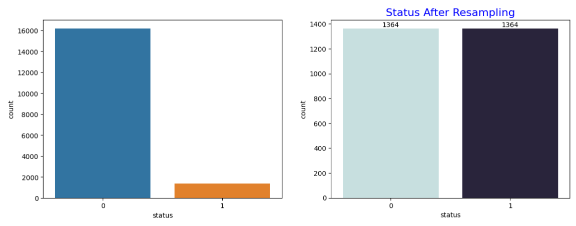


Fig. 11: Before and after resampling label data

3) Normalizing the data

We observed that the majority of features in the dataset are numeric features, so we decided to normalize them. Data normalization is a technique utilized to standardize input data, aiming to prevent certain columns with larger values from exerting a disproportionate influence on the model's predictions while potentially overlooking columns with smaller values.

There are two widely employed methods for data normalization:

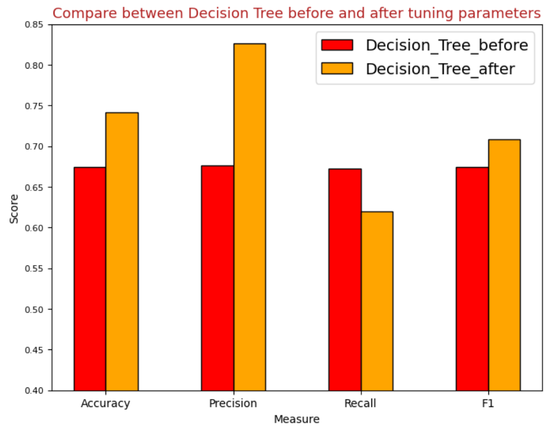
- Min-Max normalization: This technique scales the values of a feature to a range between 0 and 1. This is done by subtracting the minimum value of the feature from each value, and then dividing by the range of the feature.
- Data standardization: also known as feature scaling, is a technique in machine learning that transforms numerical features to have a mean of 0 and a standard deviation of 1. It ensures that features are on a common scale and prevents bias based on magnitude. Standardization facilitates gradient descent optimization, aids interpretability of coefficients, and should be applied consistently to training and test data.

In the context of this dataset, the chosen approach is Min-Max normalization, specifically utilizing the Min-Max normalization method. The objective is to ensure that all the training data in each column is transformed to fall within the range of [0, 1]. This normalization procedure helps mitigate any potential bias arising from columns with larger values, enabling all columns to contribute equally to the model's predictions.

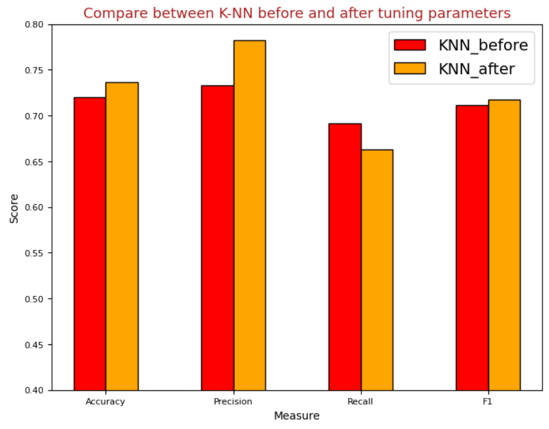
B. Evaluation

To evaluate the efficiency of the classifiers, we used the confusion matrix on the test datasets that was known to result in advance. To calculate the performance of the classifiers, we used three common measures: precision (P), recall (R), F1—Score (F1). For comparison results to be compatible with hyperparameter tuning results, we will use k fold cross validation with k=5 and tuning hyperparameter with Grid Search.

1) Comparison of before and after tuning hyperparameters



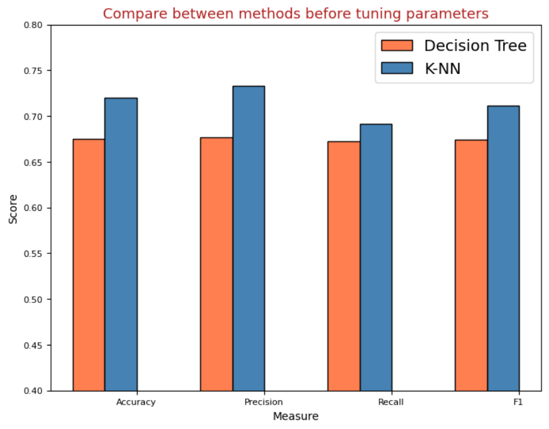
(a) Tuned Decision Tree.



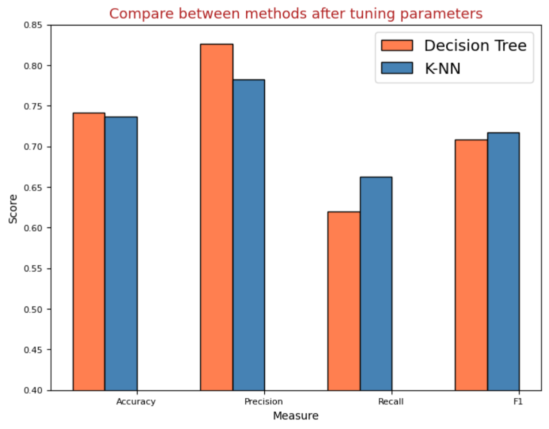
(b) Tuned K-NN.

Fig. 12: Plots showing before and after tuning hyperparameters in each model

2) Comparison of tuning two methods



(a) Before tuning



(b) After tuning

Fig. 13: Plots showing before and after tuning of Decision Tree and K-NN models.

TABLE II: Evaluation of classification algorithms(%)

Algorithms	**Acc**	**P**	**R**	**F1**
K-NN	71.4	75.2	64.9	69.6
K-NN using K-fold	72	73.3	69.1	71.1
Tuned K-NN	73.6	78.2	66.3	71.8
Decision Tree	67	68.5	64.5	66.4
Decision Tree using K-fold	67.5	67.7	67.2	67.4
Tuned Decision Tree	74.2	82.6	61.9	70.8

Table II and figure 12, 13 compare the results of both algorithms. As can be seen, experimental results show that Decision Tree performs classification better than K-Nearest Neighbors since accuracy and precision values of Decision Tree is higher than that of K-Nearest Neighbors (with hyperparameter tuning). Overall, the results of both algorithms increase. This shows the efficiency in tuning hyperparameters for our classification model, since our model prioritized precision. After experiment, we summarize some experiences following table III.

TABLE III: Comparison algorithms

Parameter	**KNN**	**Decision Tree**
Deterministic/ Non-deterministic	Non-deterministic	Deterministic
Effectiveness on	Small data	Large data
Speed	Slower for large data	Faster
Dataset	It can't deal with noisy data	It can deal with noisy data
Accuracy	High accuracy	

V. Conclusion

In conclusion, from analysis on comparison among classification algorithms (Decision Tree, K-NN) after tuning, it shows that all K-Nearest Neighbors algorithm are the most accurate in terms of the accuracy score. About the precision score. While the Decision Tree has the highest precision percentage point, K-NN has the highest recall and F1 score. In this dataset, if we prioritize recall, it wouldn't be appropriate because it tends to falsely label others as deceased, which can lead to unnecessary physical and psychological distress. The well-being and mental health of individuals are crucial when it comes to treating diseases. Therefore, to be more suitable, we will rely on accuracy and precision instead. As a result, we choose the tuned Decision Tree as the key algorithm to project the seriously infected patients.

References

[1] Antony Christopher. 2021. K-Nearest Neighbours

[2] John Clements. 2021. K-Nearest Neighbors (K-NN) Explained

[3] Renu Khandelwal. 2018. K-Nearest Neighbors(KNN)

[4] Priyanka and D. Kumar, "Decision tree classifier: a detailed survey," International Journal of Information and Decision Sciences, vol. 12, no. 3, pp. 246–269, 2020.

[5] RekhaMolala, "Entropy, Information gain and Gini Index; the crux of a Decision Tree," Medium, Mar. 23, 2020. https://blog.clairvoyantsoft.com/entropy-information-gain-and-giniindex-the-crux-of-a-decision-tree-99d0cdc699f4 (accessed Dec. 28, 2020).

[6] BhaveshPatankar and Dr. Vijay Chavda, "A Comparative Study of Decision Tree, Naive Bayesian and k-nn Classifiers in Data Mining", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 4, Issue 12, December 2014

[7] Zhang X, Jiang S. A Splitting Criteria Based on Similarity in Decision

[8] Pandey M, Sharma VK. A decision tree algorithm pertaining to the student performance analysis and prediction. International Journal of Computer Applications. 2013 Jan 1;61(13).

[9] Jayakameswaraiah M, Ramakrishna S. Implementation of an Improved ID3 Decision Tree Algorithm in Data Mining System. International Journal of Computer Science and EngineeringVolume-2, Issue-3 E-ISSN. 2014.

[10] Priyanka and D. Kumar, "Decision tree classifier: a detailed survey," International Journal of Information and Decision Sciences, vol. 12, no. 3, pp. 246–269, 2020

[11] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," Machine learning, vol. 40, no. 3, pp. 203–228, 2000.

[12] Y. Bengio, O. Delalleau, and C. Simard, "DECISION TREES DO NOT GENERALIZE TO NEW VARIATIONS," COMPUTATIONAL INTELLIGENCE, p. 19

[13] I. Ramadhan, P. Sukarno, and M. A. Nugroho, "Comparative Analysis of K-Nearest Neighbor and Decision Tree in Detecting Distributed Denial of Service," in 2020 8th International Conference on Information and Communication Technology (ICoICT), Yogyakarta, Indonesia, Jun. 2020, pp. 1–4, doi: 10.1109/ICoICT49345.2020.9166380.

[14] S.Archana and Dr. K.Elangovan, "Survey of Classification Techniques in Data Mining", International Journal of Computer Science and Mobile Applications, Vol. 2 Issue. 2, February 2014.

[15] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician, 46(3), 175-185.

[16] Quinlan, J. R. (1996). Bagging, boosting, and C4. 5. In AAAI/IAAI (pp. 725-730).

[17] Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. Data mining and knowledge discovery, 2(4), 345-389.

[18] Dasarathy, B. V. (1991). Nearest neighbor (NN) norms: NN pattern classification techniques (Vol. 24). Los Alamitos, CA: IEEE Computer Society Press.