Predicting high risk of death patients due to COVID-19 using classification algorithms

1st Bui Quoc Thinh
Computer Science
Univeristy of Information Technology - VNUHCM
Ho Chi Minh City, Viet Nam
20520934@gm.uit.edu.vn

2nd Bui Viet Dat
Computer Science
Univeristy of Information Technology - VNUHCM
Ho Chi Minh City, Viet Nam
20521162@gm.uit.edu.vn

3rd Le Tan Loc
Computer Science
Univeristy of Information Technology - VNUHCM
Ho Chi Minh City, Viet Nam
20521546@gm.uit.edu.vn

***Abstract*—Knowing who is going to be a patient at high risk of death is crucial in designing the right treatment for each individual. Implementing classification algorithms into COVID-19 disease data, we can determine whether a person is at high risk or not, which express all problems of the patient, since changes necessary will be applied to improve the quality of healthcare. This study focused on the machine learning (Naive Bayes, K-nearest neighbors and Decision Tree) on the COVID-19 dataset provided by the Mexican government. The final results were compared and evaluated to find the most effective model based on different evaluation criteria. The final results were compared and evaluated to find the most effective model based on different evaluation criteria. The experimental results show that K-nearest neighbors algorithm outperformed than two other algorithms in terms of the accuracy, precision and F1-score measurement, with 91.27%, 88.21% and 91.58% on the death classification task. However, Decision Tree has the highest recall with 95.25%. With the use of this application, the hospital can propose an appropriate method to improve the quality of healthcare in the future.**

I. Introduction

Coronavirus disease (COVID-19) is a viral infection caused by a recently identified coronavirus. The majority of patients infected with the COVID-19 virus will develop mild to moderate respiratory disease and will recover without the need for special care. People over the age of 65, as well as those with underlying medical conditions such as cardiovascular disease, diabetes, chronic respiratory disease, and cancer, are more prone to acquire severe illness. One of the major issues that healthcare practitioners have encountered during the epidemic is a lack of medical resources and a sufficient plan to distribute them. In these difficult times, being able to predict what kind of resource an individual might require at the time of being tested positive, or even before that, will be extremely beneficial to authorities because they will be able to procure and arrange for the resources required to save that patient's life. The primary purpose of this research is to create a machine learning model that can predict if a COVID-19 patient is at high risk or not based on current symptoms, status, and medical history. We are going to use three classification algorithms including Naive Bayes, K-nearest neighbors and Decision Tree for this research. Bui Quoc Thinh will be responsible for Naive Bayes algorithm, Bui Viet Dat will do K-nearest neighbors algorithm and Le Tan Loc will take responsibility for Decision Tree algorithm.

II. DATA

The dataset was given by the Mexican government. The reason we chose this dataset is that COVID-19 is still out there, especially in China, and many people are still unvaccinated. This dataset comprises a massive amount of anonymized patient-related information, including pre-conditions. The raw dataset includes 21 distinct characteristics and 1,048,576 distinct patients. 1 indicates "yes" and 2 means "no" in the Boolean features. The values 97 and 99 are missing data.

- sex: female or male.
- age: of the patient.
- classification: COVID-19 test findings. Values 1-3 mean that the patient was diagnosed with COVID-19 in different degrees. 4 or higher means that the patient is not a carrier of COVID-19 or that the test is inconclusive.
- patient type: hospitalized or not hospitalized.
- pneumonia: whether the patient already has air sacs inflammation or not.
- pregnancy: whether the patient is pregnant or not.
- diabetes: whether the patient has diabetes or not.
- copd: Indicates whether the patient has Chronic obstructive pulmonary disease or not.
- asthma: whether the patient has asthma or not.
- inmsupr: whether the patient is immunosuppressed or not.
- hypertension: whether the patient has hypertension or not.
- cardiovascular: whether the patient has heart or blood vessels related disease.
- renal chronic: whether the patient has chronic renal disease or not.
- other disease: whether the patient has other disease or not.
- obesity: whether the patient is obese or not.
- tobacco: whether the patient is a tobacco user.
- usmr: Indicates whether the patient treated medical units of the first, second or third level.
- medical unit: type of institution of the National Health System that provided the care.

- intubed: whether the patient was connected to the ventilator.
- icu: Indicates whether the patient has been admitted to an Intensive Care Unit.
- death: indicates whether the patient died or recovered.

III. Methods

A classifier is a machine learning model that is used to discriminate different objects based on certain features.

A. Naive Bayes

1) Definition: Naive Bayes classifiers are a type of basic "probabilistic classifier" in statistics that are based on using Bayes' theorem with strong independence assumptions between the features. They are among the most basic Bayesian network models, but when combined with kernel density estimation, they may attain great levels of accuracy. Naive Bayes classifiers are extremely scalable, with parameters that are proportional to the number of variables (features/predictors) in a learning task. Maximum-likelihood training may be done in linear time by evaluating a closed-form expression, as opposed to the costly iterative approximation employed for many other types of classifiers. Naive Bayes models are known by several names in the statistics field, including simple Bayes and independence Bayes. All of these titles allude to the employment of Bayes' theorem in the decision rule of the classifier, however naive Bayes is not (necessarily) a Bayesian approach.

2) Bayes' Theorem: Before moving to the formula for Naive Bayes, it is important to know about Bayes' theorem.
Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Fig. 1. Bayes' Theorem.

Where X and y are events and P(X) $\neq$ 0.
Basically, we are trying to find probability of event y, given the event X is true. Event X is also termed as evidence.

- P(y) is the priori of y (the prior probability, i.e., Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event X).
- P(X) is the prior probability of predictor of class.
- P(y|X) is a posteriori probability of B, i.e., probability of event after evidence is seen.
- P(X|y) is the likelihood, which is the probability of predictor of given class.

On the other hand, we can say that using Bayes theorem, we can find the probability of y happening, given that X has occurred. Here, X is the evidence and y is the hypothesis. The assumption made here is that the predictors/features are independent. That is, the presence of one particular feature does not affect the other. Hence, it is called naive.

3) Naive Assumption: It is time to put a naive assumption to the Bayes' theorem, which is, independence among the features. So now, we split evidence into the independent parts. If any two events y and X are independent, then,

$$P(y, X) = P(y)P(X) \tag{1}$$

By substituting for X and expanding using the chain rule, we get,

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

which can be expressed as:

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

We can obtain the values for each by looking at the dataset and substitute them into the equation. For all entries in the dataset, the denominator does not change, it remains static. Therefore, the denominator can be removed, and a proportionality can be introduced.

$$P(y|x_1, ..., x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

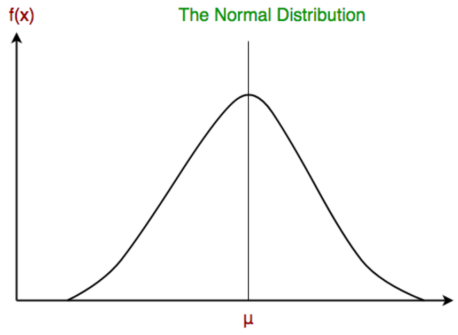
We need to create a classifier model. For this, we find the probability of given set of inputs for all possible values of the class variable y and pick up the output with maximum probability. This can be expressed mathematically as:

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

So, finally, we are left with the task of calculating P(y) and P(xi|y).
Please note that P(y) is also called class probability and P(xi|y) is called conditional probability.
The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi|y).

4) Gaussian Naive Bayes: The method that we discussed above is applicable for discrete data. In case of continuous data, we need to make some assumptions regarding the distribution of values of each feature. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi|y).
Now, we discuss one of such classifiers here. In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell-shaped curve which is symmetric about the mean of the feature values, as shown below:

The likelihood of the features is assumed to be Gaussian; hence, conditional probability is given by:



$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

5) Multinomial Naive Bayes: Feature vectors represent the frequencies with which certain events have been generated by a multinomial distribution. This is the event model typically used for document classification, whether a document belongs to the category of sports, politics, technology etc.
The features/predictors used by the classifier are the frequency of the words present in the document.

6) Bernoulli Naive Bayes: In the multivariate Bernoulli event model, features are independent booleans (binary variables) describing inputs.
Like the multinomial model, this model is popular for document classification tasks, where binary term occurrence features are used rather than term frequencies.
The parameters that we use to predict the class variable take up only values yes or no, for example, if a word occurs in the text or not.

7) Pros and Cons: Naive Bayes has some advantages and disadvantages
Advantages:

- Naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering.
- They require a small amount of training data to estimate the necessary parameters.
- Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods.
- It performs well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed.

Disadvantages:

- The requirement of predictors to be independent. In most of the real-life cases, the predictors are dependent, this hinders the performance of the classifier.
- If a categorical variable has a category (in the test data set), which was not observed in the training data set, then the model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency".
- Naïve Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.

8) Parameters: In Naive Bayes algorithms, Scikit-learn provides some parameters which support to build Naive Bayes model.
For Gaussian Naive Bayes:

a) ***priors: array-like of shape (n_classes,), default=None****:* The function *priors* represents the prior probabilities of the classes. If we specify this parameter while fitting the data, then the prior probabilities will not be justified according to the data.

b) ***var_smoothing: float, default=1e-9****:* This *var_smoothing* parameter gives the portion of the largest variance of the features that is added to variance in order to stabilize calculation.

For Multinomial Naive Bayes:

c) ***alpha: float or array-like of shape (n_features,), default=1.0****:* Additive (Laplace/Lidstone) smoothing parameter (set alpha=0 and force_alpha=True, for no smoothing).

d) ***force_alpha: bool, default=False****:* If False and alpha is less than 1e-10, it will set alpha to 1e-10. If True, alpha will remain unchanged. This may cause numerical errors if alpha is too close to 0.

e) ***fit_prior: bool, default=True****:* Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

f) ***class_prior: array-like of shape (n_classes,), default=None****:* Prior probabilities of the classes. If specified, the priors are not adjusted according to the data.

For Bernoulli Naive Bayes:

g) ***alpha: float or array-like of shape (n_features,), default=1.0****:* Additive (Laplace/Lidstone) smoothing parameter (set alpha=0 and force_alpha=True, for no smoothing).

h) ***force_alpha: bool, default=False****:* If False and alpha is less than 1e-10, it will set alpha to 1e-10. If True, alpha will remain unchanged. This may cause numerical errors if alpha is too close to 0.

i) ***binarize: float or None, default=0.0****:* Threshold for binarizing (mapping to booleans) of sample features. If None, input is presumed to already consist of binary vectors.

j) ***fit_prior: bool, default=True****:* Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

k) ***class_prior: array-like of shape (n_classes,), default=None****:* Prior probabilities of the classes. If specified, the priors are not adjusted according to the data.

9) Tuning hyperparameters: The hyperparameter tuning approaches are concerned with how we choose potential model architecture candidates from the universe of possible hyperparameter values. This is known as "searching" the hyperparameter space for the best values. We must search the hyperparameter space for the hyperparameter values that result in the highest score. There are several methods for adjusting hyperparameters, including grid search, random search, Bayesian optimization, and so on.

In this project, we need to build a model, specifically a Naive Bayes model, that can solve the problem mentioned above with high accuracy while not overfitting with training data. Some hyperparameters to consider in order to accomplish this are:

- var_smoothing: float, default=1e-9 (for Gaussian Naive Bayes)
- alpha: float or array-like of shape (n_features,), default=1.0 (for Bernoulli Naive Bayes and Multinomial Naive Bayes)

GridSearch is used to locate the "best" hyperparameters that assist the development of the model and prevent overfitting, in order to find the hyperparameters for the aforementioned problem. With this method, we just create a model for every conceivable combination of the supplied hyperparameter values, evaluate each model, and choose the design that yields the best results.

The following will be built up using various hyperparameters of the Scikit-learn Naive Bayes model:

For Gaussian Naive Bayes:

- 'var_smoothing': np.logspace(0, -9, num=100)

For Bernoulli Naive Bayes and Multinomial Naive Bayes:

- 'alpha': np.linspace(0, 100, num=200)

After Grid Search, we have the best hyperparameters for Gaussian Naive Bayes is 'var_smoothing': 0.1 and for Bernoulli Naive Bayes is 'alpha': 5.025125628140703

10) Protocol: The Naive Bayes classifier works as follows: 1. Let COVID-19 be the training dataset associated with class labels. Each tuple is represented by n-dimensional element vector.

$$X = (x_1, x_2, x_3, ..., x_n)$$

2. Consider that there are m classes y1, y2, y3..., ym. Suppose that we want to classify an unknown tuple X, then the classifier will predict that X belongs to the class with higher posterior probability, conditioned on X. i.e., the Naive Bayesian classifier assigns an unknown tuple X to the class yi if and only if P(yi|X) > P(yj|X) For $1 \leqslant j \leqslant m$, and $i \neq j$, above posterior probabilities are computed using Bayes Theorem.

B. 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.

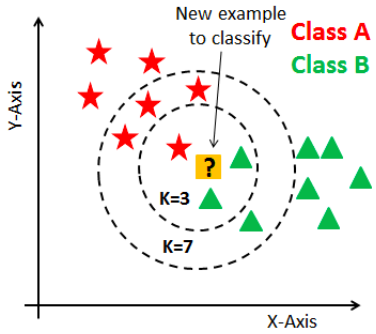


Fig. 2. Example of K-NN.

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.

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

In (2), 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.

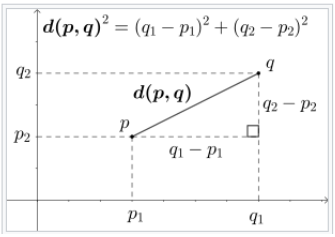


Fig. 3. Example of Euclidean Distance.

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n} |q_i - p_i|} \tag{3}$$

In (3), q and p are two different points which have n dimensions

Fig. 4. Example of Manhattan distance and Euclidean distance.

In Fig. 3, the blue line denotes the Manhattan distance between two points, whereas the red 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(nm), 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 30 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': 'euclidean'
- 'n_neighbors': 20
- 'weights': 'uniform'

8) Protocol:

- Step 1: Choose a value for K. K should be an odd number, as we only have two classes, 'Yes' or 'No'.

- 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.

C. 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.

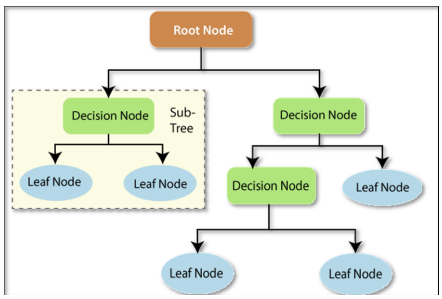


Fig. 5. Example of Decision Tree.

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.

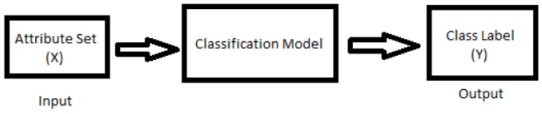


Fig. 6. Flowchart of Classification Tree.

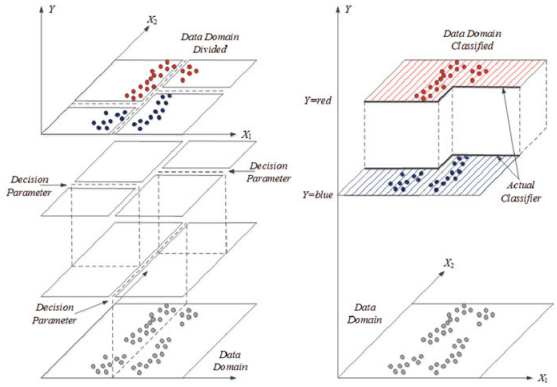


Fig. 7. The Classification Tree is illustrated in 3D using two classes with domain division properties. Response variable Y has two discrete values, red or blue.

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.

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.

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.

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

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.

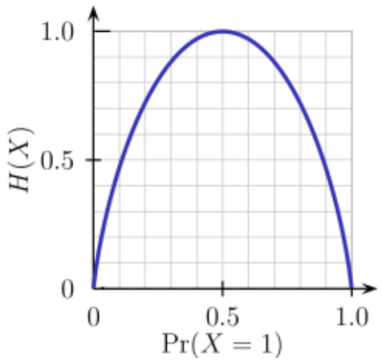


Fig. 8. Entropy graph.

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.

$$GainRatio = \frac{InformationGain}{SplitInfo} = \frac{Entropy(before) - \sum_{j=1}^{k} Entropy(j, after)}{\sum_{j=1}^{k} log_2 w_j} \quad (4)$$

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(3,10)
- 'min_samples_split' = np.arange(3,10,1)
- 'min_samples_leaf' = np.arange(3,10)
- 'max_leaf_node' = [8,16,32]

After Grid Search, we have the best hyperparameters are 'criterion': 'gini', 'max_depth': 8, 'max_leaf_nodes': 32, 'min_samples_leaf': 3, 'min_samples_split': 3.

7) Protocol:

- 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.

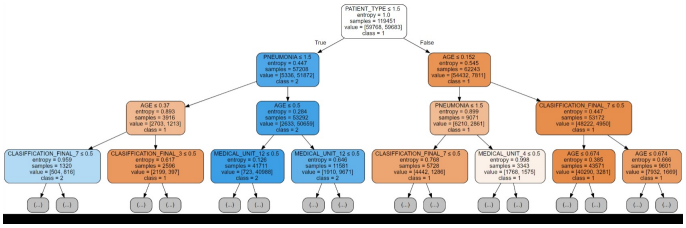


Fig. 9. Decision Tree for COVID-19 Dataset.

IV. Experimental Settings

A. Preprocessing

Experiments were conducted to better understand the operation mechanism of the three main algorithms: Naive Bayes, K-nearest neighbors and Decision Tree. As start above, we use COVID-19 dataset with mission predicting high risk patients of Covid-19. All 3 algorithms were implemented on Python frameworks. Before classification, we conducted data preprocessing for next steps. This process including remove missing values, normalize data and drop irrelevant features.

In the dataset, there are features in which the missing values are the majority of the samples, such as 'ICU' and 'Intubed', so we decide to remove those two columns. As for the 'PREGNANT' column, after plotting, we see that there is a number of 'men' values that are missing. Therefore, we

decide to change those into 'no' as men obviously cannot be pregnant. After all, we remove the missing values in the other columns.

We then plot the heatmap denoting the correlations between given features to see the relationships between the 'DEATH' columns and the others. The features that have the value smaller than 0 will be removed. As a result, 'SEX', 'PREGNANT', 'COPD', 'ASTHMA', 'INMSUPR', 'OTHER_DISEASE', 'CARDIOVASCULAR', 'OBESITY', 'TOBACCO' columns are removed.

Additionally, we see that the dataset have the feature "AGE" which is a numeric feature, so we will use StandardScaler in order to normalize the values with the mean equals 0 and scales the data to unit variance.

A remarkable point in the COVID-19 dataset, we found that the amount of data is imbalance, we will resample data before training model.

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.

V. Experimental Results

TABLE II
EVALUATION OF CLASSIFICATION ALGORITHMS(%)

Algorithms	**Acc**	**P**	**R**	**F1**
Gaussian Naive Bayes	73.1	77.4	75.0	74.4
Tuned Gaussian Naive Bayes	89.6	89.2	90.2	89.7
Bernoulli Gaussian Naive Bayes	64.8	58.3	55.6	56.3
Tuned Bernoulli Gaussian Naive Bayes	76.8	73.1	84.5	78.4
K-NN	83.6	81.4	86.9	83.9
Tuned K-NN	91.3	88.2	95.3	91.6
Decision Tree	85.3	86.5	83.5	84.7
Tuned Decision Tree	91.1	87.3	96.2	91.5

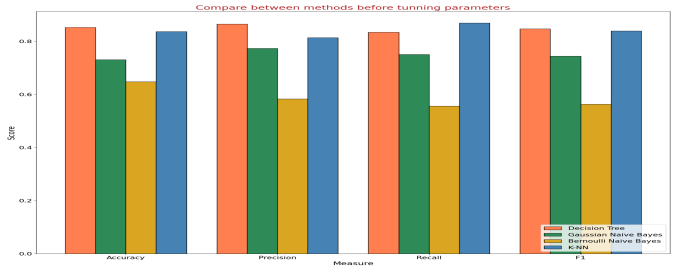


Fig. 10. Before tuning.

Table II and figure 10, 11 compare the results of both algorithms. As can be seen, experimental results show that K-Nearest Neighbor and Decision Tree performs classification

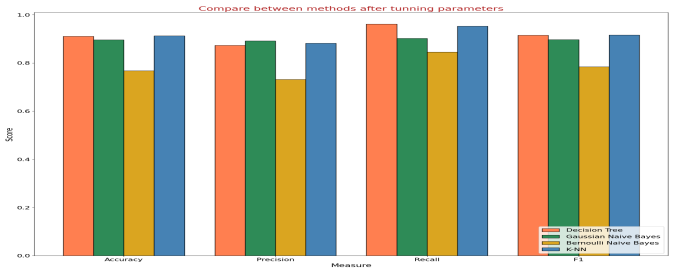


Fig. 11. After tuning.

better than Naive Bayes since every single evaluation metric of K-Nearest Neighbor and Decision Tree are higher than that of Naive Bayes under both scenarios (with hyperparameter tuning and without hyperparameter tuning). Overall, the results of both algorithms increase, noticeably recall values have a dramatic rise. This shows the efficiency in tuning hyperparameters for our classification model, since our model prioritized recall.

After experiment, we summarize some experiences following table III

TABLE III
COMPARISON ALGORITHMS

Parameter	**KNN**	**Naive Bayes**	**Decision Tree**
Deterministic/ Non-deterministic	Non-deterministic	Non-deterministic	Deterministic
Effectiveness on	Small data	Huge data	Large data
Speed	Slower for large data,	Faster than KNN	Faster
Dataset	It can't deal with noisy data	It can deal with noisy data	It can deal with noisy data
Accuracy	Provides high accuracy	For obtaining good results it requires a very large number of records	High accuracy

VI. Conclusion

In conclusion, from analysis on comparison among classification algorithms (Decision tree, K-NN, Naive Bayes) after tuning, it shows that all K-Nearest Neighbors algorithm are the most accurate in terms of the accuracy score. About the precision score, the Gaussian Naive Bayes is the best algorithm. Also, while the Decision Tree has the highest recall percentage point, K-NN has the highest F1 score. In this project, in order to predict patients at the high risk of death due to Covid-19 accurately, we indeed focus on the recall score as. As a result, we choose the tuned Decision Tree as the key algorithm to project the seriously infected patients.

References

[1] Naive Bayes classifier - Wikipedia. http://www.overleaf.com
[2] 1.9. Naive Bayes — scikit-learn 1.2.0 documentation. https://scikit-learn.org/stable/modules/naive_bayes.html

[3] Exercise 6: Naive Bayes - Machine Learning - Andrew Ng.
[4] Text Classification and Naive Bayes - Stanford. http://stanford.edu/ jurafsky/slp3/slides/7_NB.pdf
[5] Naive Bayes Classifier - Rohith Gandhi, Towards Data Science. https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
[6] Data Mining - Naive Bayes (NB) - DataCadamia. https://datacadamia.com/data_mining/naive_bayes
[7] Naive Bayes Classifiers - GeeksforGeeks. https://www.geeksforgeeks.org/naive-bayes-classifiers/
[8] Antony Christopher. 2021. K-Nearest Neighbours
[9] John Clements. 2021. K-Nearest Neighbors (K-NN) Explained
[10] Renu Khandelwal. 2018. K-Nearest Neighbors(KNN)
[11] 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.
[12] 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).
[13] 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
[14] Zhang X, Jiang S. A Splitting Criteria Based on Similarity in Decision
[15] 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).
[16] 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.
[17] 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
[18] 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.
[19] Y. Bengio, O. Delalleau, and C. Simard, "DECISION TREES DO NOT GENERALIZE TO NEW VARIATIONS," COMPUTATIONAL INTELLIGENCE, p. 19
[20] 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.
[21] 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.