Coronary Artery Disease Prediction Using Data Mining Techniques

Mohammad Husain
B160589CS

Rebaka Tejaswi
B160552CS

Rahul Kumar
B160756CS

Sushant Pradhan
B160598CS

***Abstract*—Coronary artery disease (CAD) is the most common type of heart disease which is the leading cause of death in the United States in both men and women.It happens when the arteries that supply blood to heart muscle become hardened and narrowed due to the deposit of cholesterol and other material, called "plaque", on their inner walls, called atherosclerosis. As it grows,the heart muscle can't get the blood or oxygen it needs which lead to chest pain (angina) shortness of breath or a heart attack. Over time, CAD can also weaken the heart muscle and contribute to heart failure and Arrhythmias(changes in the normal beating rhythm of the heart).**

Angioplasty and Heart bypass surgery are two types of surgical treatments for coronary artery disease. Surgery, such as angioplasty or heart bypass surgery, has potential risks. These include heart attack, stroke, or death.

CAD causes more deaths and disability and incurs greater economic costs than any other illness in the developed world is likely to become the most common cause of death worldwide by 2020.

Undoubtedly this alarming figure needs great attention.The remarkable advances in the field of machine learning have led to faster identification of numerous diseases.In our project we aim to use various data mining models to predict Coronary Artery Disease.We will also be analyzing the accuracy and efficiency of each model and proposing the best model amongst them.

I. Introduction

Coronary Artery Disease also known as 'Ischemic Heart Disease(IHD)' or 'Atherosclerotic Heart Disease' or 'Acute Coronary Syndrome' is one of the most common type of heart disease. It is a common term for the buildup of plaque in the heart's arteries that could lead to heart attack. As the arteries become blocked over time,one may experience Angina,shortness of breath and heart attack.Coronary artery disease may take years to develop and hence it's less possible to notice any symptoms until the disease has progressed significantly.Over time, CAD can also weaken the heart muscles and contribute to heart failure and Arrhythmias(changes in the normal beating rhythm of the heart).

With coronary artery disease, plaque first grows within the walls of the coronary arteries until the blood flow to the hearts muscle is limited.This is also called ischemia. It may be chronic, narrowing of the coronary artery over time and limiting of the blood supply to part of the muscle. Or it can be acute, resulting from a sudden rupture of a plaque and formation of a thrombus or blood clot.

The traditional risk factors for coronary artery disease are high LDL (Low-density lipo-protein) cholesterol, low HDL(High-denisty lipo-protein) cholesterol, high blood pressure, family history, diabetes, smoking, being post-menopausal for women and being older than 45 for men, according to Fisher ,who is former editor of the American Heart Association journal,ATVB. Obesity may also be a risk factor.

"Coronary artery disease begins in childhood, so that by the teenage years, there is evidence that plaques that will stay with us for life are formed in most people," said Edward A Fisher.

Coronary artery disease is preventable, typical warning signs include chest pain, shortness of breath, palpitations and even fatigue.Early detection of this disease may help in preventing the further risks and serious problems.

II. Problem Definition

Analyzing the data set using the various prediction models to find out the most suitable model in terms of accuracy and efficiency.

A. *DataSet*

Dataset is taken from UCI and consists of 303 Tuples, each having 13 features and 1 target attribute.

1) Age - age in years
2) Sex - sex (1 = male; 0 = female)
3) Cp - chest pain type
    - Value 1: typical angina

- Value 2: atypical angina
- Value 3: non-anginal pain
- Value 4: asymptomatic

4) Trestbps - resting blood pressure (in mm Hg on admission to the hospital)
5) Chol- serum cholesterol in mg/dl
6) Fbs - (fasting blood sugar ¿ 120 mg/dl) (1 = true; 0 = false)
7) Restecg - resting electrocardiographic results
   - Value 0: normal
   - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of ¿ 0.05 mV)
   - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8) Thalach - maximum heart rate achieved
9) Exang - exercise induced angina (1 = yes; 0 = no)
10) Oldpeak - ST depression induced by exercise relative to rest
11) Slope - slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: down sloping
12) Ca - number of major vessels (0-3) colored by fluoroscopy
13) Thal - 3 = normal; 6 = fixed defect; 7 = reversable defect
14) Target - goal attribute
    - Value 0: less than 50% diameter narrowing
    - Value 1: greater than 50% diameter narrowing

We have split the data into training and test sets in the ration 80:20, similarly as in previous model.

III. Literature Survey

Here we will be discussing about the existing models for prediction of coronary Artery Disease, such as Decision Tree, Random forest,Logistic Regression,K-NN (K Nearest Neighbour), Naive Bayesian Classifier, Support Vector Machine (SVM) and Multilayer perceptron.

A. *DECISION TREE*

A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an

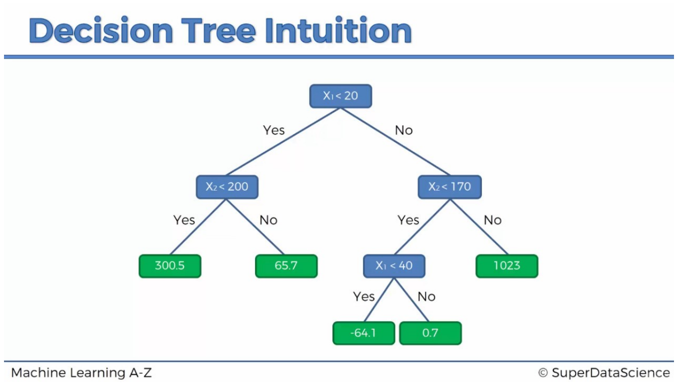


Fig. 1: Decision Tree Intuition 1

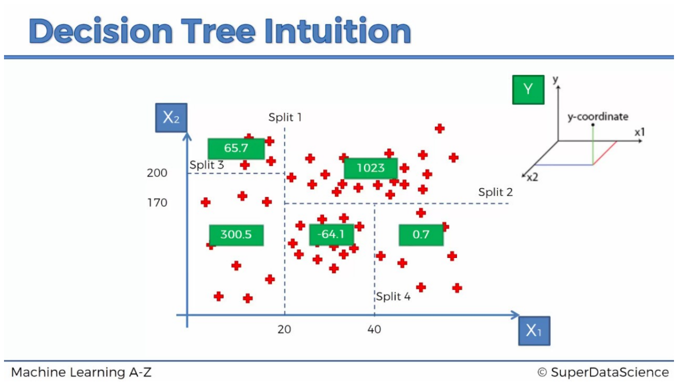


Fig. 2: Decision Tree Intuition 2

attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree,testing the attribute specified by this node,then moving down the tree branch corresponding to the value of the attribute .This process is then repeated for the sub-tree rooted at the new node.

The decision tree learning algorithm recursively learns the tree as follows:
1.Assign all training instances to the root of the tree. Set current node to root node.
2.For each attribute
a.Partition all data instances at the node by the value of the attribute.
b.Compute the information gain ratio from the partitioning.
3.Identify feature that results in the greatest information gain ratio. Set this feature to be the

splitting criterion at the current node.
-If the best information gain ratio is 0, tag the current node as a leaf and return.
4.Partition all instances according to attribute value of the best feature.
5.Denote each partition as a child node of the current node.
6.For each child node:
a.If the child node is pure (has instances from only one class) tag it as a leaf and return.
b.If not set the child node as the current node and recurse to step 2.

Algorithm 1 Decision Tree: Training

```
function GENTREE(D, attributeList)
N = createNode()
if $\forall tuple \in D, tuple.getClass() = C$ then
    N.makeLeafNode()
    N.setLabel(C)
    return N
end
if isEmpty(attributeList) then
    N.makeLeafNode()
    N.setLabel(D.getMajorityClassLabel())
    return N
end
(attribute, selectionCriteria) = attributeSelectionHeuristic(D, attributeList)
N.setLabel(attribute)
attributeList = attributeList - attribute
foreach (critera, childPointer) $\in selectionCriteria$ do
    $D_i$ = applyCriteria(D, criteria)
    if isEmpty($D_i$) then
        childPointer = createNode()
        childPointer.makeLeafNode()
        childPointer.setLabel(D.getMajorityClassLabel())
    else
        childPointer = GenTree($D_i$, attributeList)
    end
end
return N

end function
    Result: Decision Tree
```

Once a decision tree is learned, it can be used to evaluate new instances to determine their class. The instance is passed down the tree, from the root, until it arrives at a leaf. The class assigned to the instance is the class for the leaf.

Algorithm 2 Decision Tree: Classification

```
function CLASSIFY(tree, tuple)
if tree.isLeafNode() then
    return tree.getLabel()
end
attributeValue = tuple[tree.getLabel()]
foreach (criteria, childPointer) $\in tree.getChildren()$ do
    if isCriteriaSatisfied(attributeValue, criteria) then
        return Classify(childPointer, tuple)
    end
end
end function
    Result: Predicted Class Label
```

]1:CLASSIFICATION OF DECISION TREE

Once a decision tree is learned, it can be used to evaluate new instances to determine their class. The instance is passed down the tree, from the root, until it arrives at a leaf. The class assigned to the instance is the class for the leaf.

Algorithm 3 Decision Tree: Classification

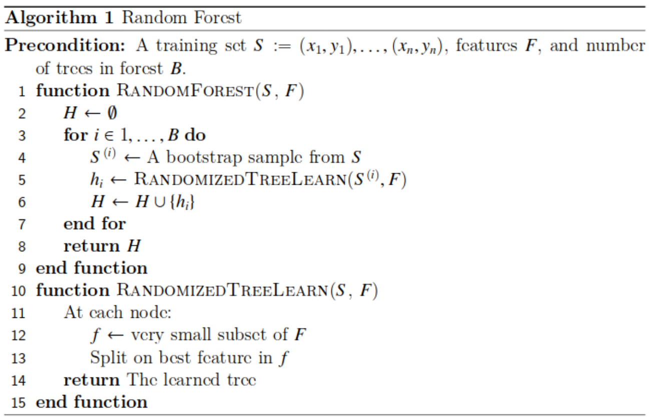
```
function CLASSIFY(tree, tuple)
if tree.isLeafNode() then
    return tree.getLabel()
end
attributeValue = tuple[tree.getLabel()]
foreach (criteria, childPointer) $\in tree.getChildren()$ do
    if isCriteriaSatisfied(attributeValue, criteria) then
        return Classify(childPointer, tuple)
    end
end
end function
    Result: Predicted Class Label
```

B. RANDOM FOREST

Each tree is constructed using the following algorithm:
1.Let the number of training cases be N, and the number of variables in the classifier be M.
2.We are told the number m of input variables to be used to determine the decision at a node of the tree; m should be much less than M.
3.Choose a training set for this tree by choosing n times with replacement from all N available training cases (i.e. take a bootstrap sample). Use the rest of the cases to estimate the error of the

tree, by predicting their classes.
4.For each node of the tree, randomly choose m variables on which to base the decision at that node. Calculate the best split based on these m variables in the training set.
5.Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier).

For prediction a new sample is pushed down the tree. It is assigned the label of the training sample in the terminal node it ends up in. This procedure is iterated over all trees in the ensemble, and the average vote of all trees is reported as random forest prediction.

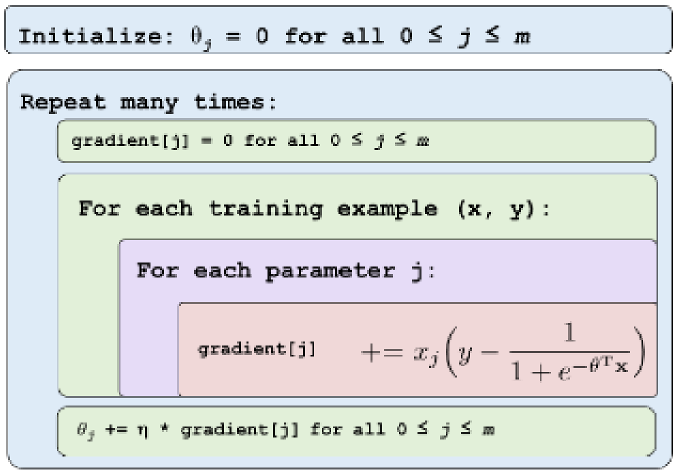


```
Algorithm 1 Random Forest
Precondition: A training set $S := (x_1, y_1), \dots, (x_n, y_n)$, features $F$, and number
    of trees in forest $B$.
function RandomForest($S$, $F$)
    $H \leftarrow \emptyset$
    for $i \in 1, \dots, B$ do
        $S^{(i)} \leftarrow$ A bootstrap sample from $S$
        $h_i \leftarrow$ RandomizedTreeLearn($S^{(i)}, F$)
        $H \leftarrow H \cup \{h_i\}$
    end for
    return $H$
end function
function RandomizedTreeLearn($S$, $F$)
    At each node:
        $f \leftarrow$ very small subset of $F$
        Split on best feature in $f$
    return The learned tree
end function
```

figureRandom Forest Algorithm

C. *LOGISTIC REGRESSION*

Logistic regression is the most famous machine learning algorithm after linear regression. In a lot of ways, linear regression and logistic regression are similar. But, the biggest difference lies in what they are used for. Linear regression algorithms are used to predict/forecast values but logistic regression is used for classification tasks. If you are shaky on the concepts of linear regression, check this out. There are many classification tasks done routinely by people. For example, classifying whether an email is a spam or not, classifying whether a tumour is malignant or benign, classifying whether a website is fraudulent or not, etc. These are typical examples where machine learning algorithms can make our lives a lot easier. A really simple, rudimentary and useful algorithm for classification is the logistic regression algorithm. Now, let's take a deeper look into logistic regression.



```
Initialize: $\theta_j = 0$ for all $0 \le j \le m$
Repeat many times:
    gradient[j] = 0 for all $0 \le j \le m$
    For each training example (x, y):
        For each parameter j:
            gradient[j] $\mathrel{+}= x_j\left(y - \frac{1}{1+e^{-\theta^T x}}\right)$
    $\theta_j$ += $\eta$ * gradient[j] for all $0 \le j \le m$
```

figure:Logistic Regression Algorithm

1) *SIGMOID FUNCTION (LOGISTIC FUNCTION):*

Logistic regression algorithm also uses a linear equation with independent predictors to predict a value. The predicted value can be anywhere between negative infinity to positive infinity. We need the output of the algorithm to be class variable, i.e 0-no, 1-yes. Therefore, we are squashing the output of the linear equation into a range of [0,1]. To squash the predicted value between 0 and 1, we use the sigmoid function.

$$z = \theta_0 + \theta_1 \cdot x_1 + \theta \cdot x_2 + \cdots$$

Fig. 3. Linear Equation and Sigmoid Function

$$h = g(z) = \frac{1}{1+e^{-z}}$$

Fig. 4. Squashed output-h

We take the output(z) of the linear equation and give to the function g(x) which returns a squashed value h, the value h will lie in the range of 0 to 1. To understand how sigmoid function squashes the values within the range, let's visualize the graph of the sigmoid function.
As you can see from the graph, the sigmoid function becomes asymptote to y=1 for positive values of x and becomes asymptote to y=0 for negative values of x.

2) *COST FUNCTION:* Since we are trying to predict class values, we cannot use the same cost function used in linear

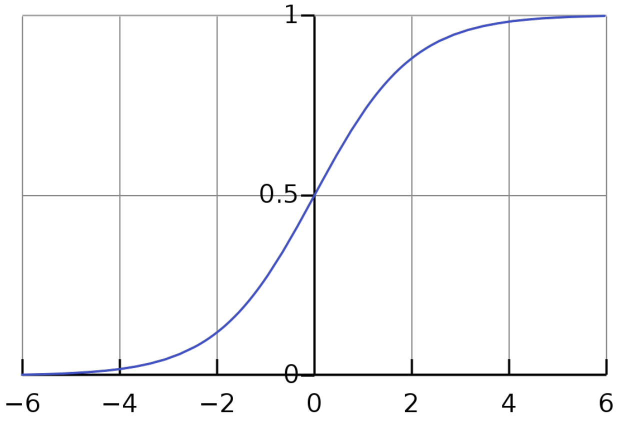


Fig. 5. Sigmoid Function graph

regression algorithm. Therefore, we use a logarithmic loss function to calculate the cost for misclassifying.

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Fig. 6. Cost function

The above cost function can be rewritten as below since calculating gradients from the above equation is difficult.

$$-\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Fig. 7. Cost function

*3) **CALCULATING GRADIENTS**:* We take partial derivatives of the cost function with respect to each parameter(theta_0, theta_1, ...) to obtain the gradients. with the help of these gradients, we can update the values of theta_0, theta_1, ... To understand the equations below you would need some calculus.

D. K NEAREST NEIGHBOURS

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply ”remember” all of its training data (possibly transformed into a fast indexing structure such as a Ball Tree or KD Tree).

$$J = \frac{-1}{m} \cdot [\sum_{i=1}^{m} y_i \cdot \log h_i + (1 - y_i) \cdot \log 1 - h_i]$$

$$\frac{\partial J}{\partial \theta_n} = \frac{-1}{m} \cdot [\sum_{i=1}^{m} \frac{y_i}{h_i} \cdot h_i^2 \cdot x_n \cdot \frac{1 - h_i}{h_i} + \frac{1 - y_i}{1 - h_i} \cdot -h_i^2 \cdot x_n \cdot \frac{1 - h_i}{h_i}]$$

$$\frac{\partial J}{\partial \theta_n} = \frac{-1}{m} \cdot [\sum_{i=1}^{m} x_n \cdot (1 - h_i) \cdot y_i - x_n \cdot h_i \cdot (1 - y_i)]$$

$$\frac{\partial J}{\partial \theta_n} = \frac{1}{m} \cdot x_i \cdot [\sum_{i=1}^{m} h_i - y_i]$$

Fig. 8. Gradients

Despite its simplicity, nearest neighbors has been successful in a large number of classification and regression problems, including handwritten digits and satellite image scenes. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

The basic nearest neighbors classification uses uniform weights: that is, the value assigned to a query point is computed from a simple majority vote of the nearest neighbors. Under some circumstances, it is better to weight the neighbors such that nearer neighbors contribute more to the fit. This can be accomplished through the weights keyword. The default value, weights = ’uniform’, assigns uniform weights to each neighbor. weights = ’distance’ assigns weights proportional to the inverse of the distance from the query point. Alternatively, a user-defined function of the distance can be supplied to compute the weights.

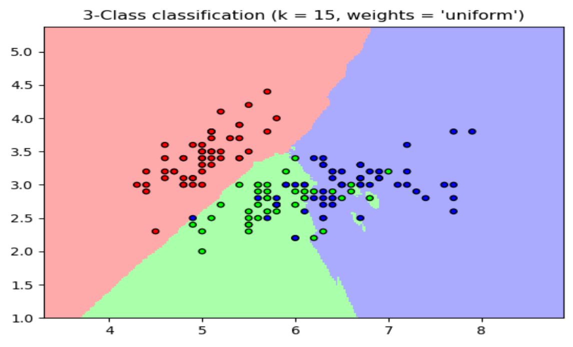


Fig. 9. 3-Class classification (k = 15,weights = ’uniform’)

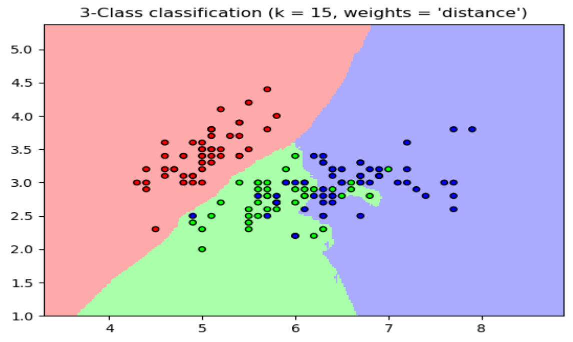


Fig. 10. 3-Class classification (k = 15,weights = ’distance’)

E. NAIVE BAYES

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the naive assumption of conditional independence between every pair of features given the value of the class variable. Bayes’ theorem states the following relationship, given class variable y and dependent feature vector

$$x_1 through x_n,: P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)} \tag{1}$$

Using the naive conditional independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y), \tag{2}$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)} \tag{3}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

P(y $\mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$
$\Downarrow$
$hat05Ey = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$ (4)

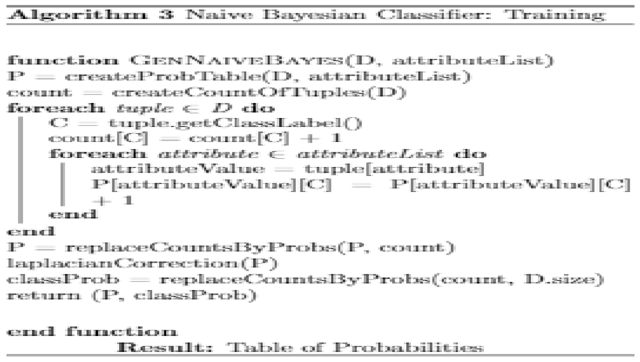
and we can use ”Maximum A Posteriori” (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

In spite of their apparently over-simplified assumptions, 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. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from *predict_proba* are not to be taken too seriously.



Algorithm 3 Naive Bayesian Classifier: Training

```
function GenNaiveBayes(D, attributeList)
P = createProbTable(D, attributeList)
count = createCountOfTuples(D)
foreach tuple ∈ D do
    C = tuple.getClassLabel()
    count[C] = count[C] + 1
    foreach attribute ∈ attributeList do
        attributeValue = tuple[attribute]
        P[attributeValue][C] = P[attributeValue][C]
        + 1
    end
end
P = replaceCountsByProbs(P, count)
laplacianCorrection(P)
classProb = replaceCountsByProbs(count, D.size)
return (P, classProb)

end function
        Result: Table of Probabilities
```

Algorithm 4 Naive Bayesian Classifier: Training

```
GenNaiveBayesD, attributeList
P = createProbTable(D, attributeList)
count = createCountOfTuples(D)
tuple ∈ D C = tuple.getClassLabel()
count[C] = count[C] + 1
attribute ∈ attributeList attributeValue = tuple[attribute]
P[attributeValue][C] = P[attributeValue][C] + 1 P =
replaceCountsByProbs(P, count)
laplacianCorrection(P)
classProb = replaceCountsByProbs(count, D.size)
return (P, classProb)

Table of Probabilities
```

Algorithm 5 Naive Bayesian Classifier: Classification

```
Classify(P, classProb), tuple, attributeList
classLabels = P.getAllClassLabels()
max = 0
classLabel ∈ classLabels p = 1
attribute ∈ attributeList attributeValue = tuple[attribute]
p = p * P[attributeValue][classLabel] p = p *
classProb[classLabel]
p > max max = p
predictedClass = classLabel return predictedClass

Predicted Class Label
```

Algorithm 4 Naive Bayesian Classifier: Classification

```
function CLASSIFY((P, classProb), tuple, attributeList)
classLabels = P.getAllClassLabels()
max = 0
foreach $classLabel \in classLabels$ do
    p = 1
    foreach $attribute \in attributeList$ do
        attributeValue = tuple[attribute]
        p = p * P[attributeValue][classLabel]
    end
    p = p * classProb[classLabel]
    if $p > max$ then
        max = p
        predictedClass = classLabel
    end
end
return predictedClass

end function
        Result: Predicted Class Label
```

F. *MULTI-LAYER PERCEPTRON*

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function

$$f(\cdot) : R\hat{m} \rightarrow R\hat{o}$$

by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features

$$X = x_1, x_2, ..., x_m$$

and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Figure 1 shows a one hidden layer MLP with scalar output.

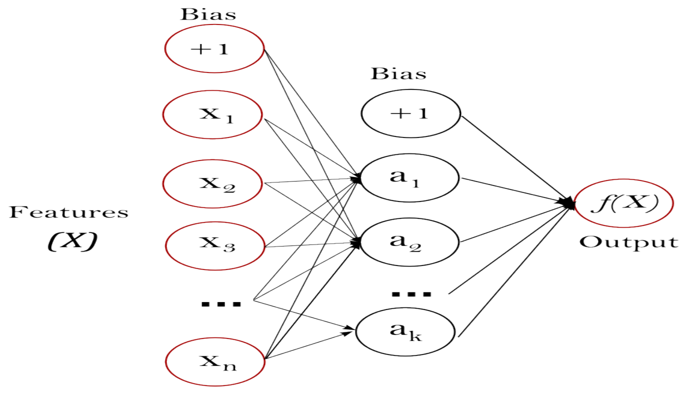


Fig. 11. One hidden layer MLP

The leftmost layer, known as the input layer, consists of a set of neurons $x_i|x_1, x_2, ..., x_m$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + ... + w_mx_m$, followed by a non-linear activation function $g(\cdot) : R \rightarrow R$ - like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

The module contains the public attributes coefs_ and intercepts_. coefs_ is a list of weight matrices, where weight matrix at index i represents the weights between layer i and layer i+1. intercepts_ is a list of bias vectors, where the vector at index i represents the bias values added to layer i+1.

The advantages of Multi-layer Perceptron are:

1.Capability to learn non-linear models.

2.Capability to learn models in real-time (on-line learning) using partial_fit.

The disadvantages of Multi-layer Perceptron (MLP) include:

1.MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.
2.MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.
3.MLP is sensitive to feature scaling.

Algorithms

MLP trains using Stochastic Gradient Descent, Adam, or L-BFGS. Stochastic Gradient Descent (SGD) updates parameters using the gradient of the loss function with respect to a parameter that needs adaptation, i.e.

$$w \leftarrow w - \eta(\alpha \frac{\partial R(w)}{\partial w} + \frac{\partial Loss}{\partial w})$$

where η is the learning rate which controls the step-size in the parameter space search. Loss is the loss function used for the network.

Adam is similar to SGD in a sense that it is a stochastic optimizer, but it can automatically adjust the amount to update parameters based on adaptive estimates of lower-order moments.With SGD or Adam, training supports online and mini-batch learning.

L-BFGS is a solver that approximates the Hessian matrix which represents the second-order partial derivative of a function. Further it approximates the inverse of the Hessian matrix to perform parameter updates. The implementation uses the Scipy version of L-BFGS.

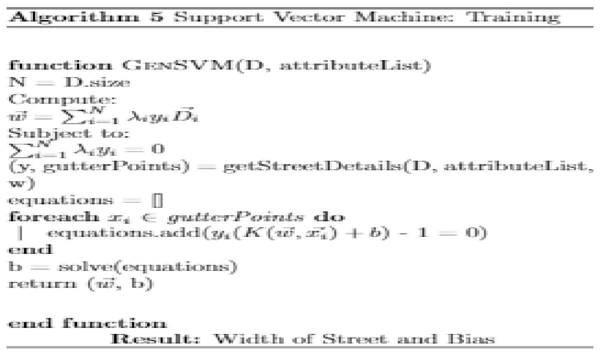
If the selected solver is L-BFGS, training does not support online nor mini-batch learning.

Complexity

Suppose there are n training samples, m features, k hidden layers, each containing h neurons - for simplicity, and o output neurons. The time complexity of backpropagation is $O(n \cdot m \cdot h^k \cdot o \cdot i)$, where i is the number of iterations. Since backpropagation has a high time complexity, it is advisable to start with smaller number of hidden neurons and few hidden layers for training.

G. *SUPPORT VECTOR MACHINE*

Support vector machine (SVM) is a method for the classification of both linear and non linear data. In case the data is non linear, it uses a non linear map- ping to transform the original training data into a higher dimension.With an appropriate non linear mapping to map lower dimensional data to a sufficiently higher dimension, data from two classes can always be separated by a hyper-plane. The SVM finds this hyper-plane using support vectors (es- sential training tuples) and margins (defined by the support vectors).



Algorithm 5 Support Vector Machine: Training

```
function GENSVM(D, attributeList)
N = D.size
Compute:
$\vec{w} = \sum_{i=1}^{N} \lambda_i y_i \vec{D_i}$
Subject to:
$\sum_{i=1}^{N} \lambda_i y_i = 0$
(y, gutterPoints) = getStreetDetails(D, attributeList, w)
equations = []
foreach $x_i \in gutterPoints$ do
    equations.add($y_i(K(\vec{w}, \vec{x_i}) + b)$ - 1 = 0)
end
b = solve(equations)
return ($\vec{w}$, b)

end function
        Result: Width of Street and Bias
```

Algorithm 6 Support Vector Machine: Training

```
GenSVMD, attributeList
N = D.size
Compute:
$w = \sum_{i=1}^{N} \lambda_i y_i D_i$
Subject to:
$\sum_{i=1}^{N} \lambda_i y_i = 0$
(y, gutterPoints) = getStreetDetails(D, attributeList, w)
equations = []
$x_i \in$ gutterPoints equations.add($y_i(K(w, x_i) + b)$ - 1 = 0)  b = solve(equations)
return ($w$, b)
```

Width of Street and Bias

SVMs are set of related supervised learning methods used for classification and regression [2]. They belong to a family of generalized linear classification. A special property of SVM is , SVM simultaneously minimize the empirical classification error and maximize the geometric margin. So SVM called Maximum Margin Classifiers. SVM is based on the Structural risk Minimization (SRM). SVM map input vector to a higher dimensional space where a maximal separating hyper-plane is constructed. Two parallel hyper-planes are constructed on each side of the hyper-plane that separate the data. The separating hyper-plane is the hyper-plane that maximize the distance between the two parallel hyper-planes. An assumption is made that the larger the margin or distance between these parallel hyper-planes the better the generalization error of the classifier will be We consider data points of the form
$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)..., (x_n, y_n)$.
Where $y_n = 1/-1$, a constant denoting the class to which that point x_n belongs. n = number of sample. Each x_n is p-dimensional real vector. The scaling is important to guard against variable (attributes) with larger variance. We can view this Training data , by means of the dividing (or separating) hyper-plane , which takes

$$w.x + b = o - - - - - - (1)$$

Where b is scalar and w is p-dimensional Vector. The vector w points perpendicular to the separating hyper-plane . Adding the offset parameter b allows us to increase the margin. Absent of b, the hyper-plane is forced to pass through the origin , restricting the solution. As we are interesting in the maximum margin , we are interested SVM and the parallel hyper-planes. Parallel hyper-planes can be described by equation
$w.x + b = 1$
$w.x + b = -1$
If the training data are linearly separable, we can select these hyper-planes so that there are no points between them and then try to maximize their distance. By geometry, We find the distance between the hyper-plane is $2/|w|$. So we want to minimize w. To excite data points, we need to ensure that for all I either

$$w.x_i - b \geq 1 \;\; or \;\; w.x_i - b \leq -1$$

This can be written as

$$y_i(w.x_i - b) \geq 1 \;\; , \;\; 1 \leq i \leq n \qquad ....(2)$$

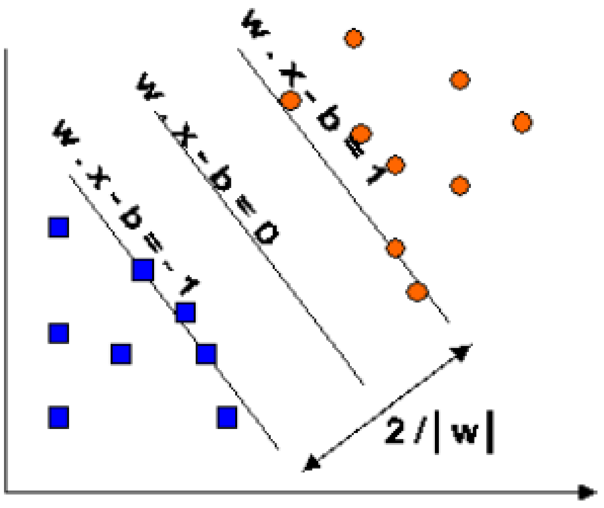


Fig. 12. Figure.1 Maximum margin hyper-planes for a SVM trained with samples from two classes

Algorithm 6 Support Vector Machine: Classification

```
function CLASSIFY(($\vec{w}$, b), $\vec{tuple}$)
if $K(\vec{w}, \vec{tuple}) + b \geq 1$ then
  | return getClassLabel1()
end
if $K(\vec{w}, \vec{tuple}) + b \leq -1$ then
  | return getClassLabel2()
end
end function
    Result: Predicted Class Label
```

Algorithm 7 Support Vector Machine: Classification

Classify(w, b), $tuple$
K(w, $tuple$) + b $\geq$ 1 return getClassLabel1() K(w, $tuple$) + b $\leq$ -1 return getClassLabel2()
Predicted Class Label

Samples along the hyper-planes are called Support Vectors (SVs). A separating hyper-plane with the largest margin defined by M = 2 / w that is specifies support vectors means training data points closets to it. Which satisfy?

$$y_j [w^T . x_j + b] = 1 \quad , i = l \qquad \text{-----(3)}$$

Optimal Canonical hyper-plane (OCH) is a canonical hyper-plane having a maximum margin. For all the data, OCH should satisfy the following constraints

$$y_i[w^T . x_i + b] \geq 1 \quad ; \ i = 1,2\ldots l \qquad \text{------(4)}$$

Where l is Number of Training data point. In order to find the optimal separating hyper-plane having a maximul margin, A learning macine should minimize w*w subject to the inequality constraints

$$y_i [w^T . x_i + b] \geq 1 \quad ; \ i = 1,2.......l$$

This optimization problem solved by the saddle points of the Lagranges Function

$$L_P = L_{(w,b,\alpha)} = 1/2 \| w \| 2 - \sum_{i=1}^{l} \alpha_i \ (y_i (w^T x_i + b) - 1)$$

$$= 1/2 \, w^T w - \sum_{i=1}^{l} \alpha_i \ (y_i(w^T x_i + b) - 1) \ \text{---(5)}$$

Where i is a Lagranges multiplier .The search for an optimal saddle points $(w0, b0, 0)$ is necessary because Lagranges must be minimized with respect to w and b and has to be maximized with respect to nonnegative $i(i0)$. This problem can be solved either in primal form (which is the form of w b) or in a dual form (which is the form of i).Equation number (4) and (5) are convex and KKT conditions, which are necessary and sufficient conditions for a maximum of equation (4). Partially differentiate equation (5) with respect to saddle points $(w0, b0, 0)$.

$$\partial L / \partial w_0 = 0$$

$$\text{i .e} \qquad w_0 = \sum_{i=1}^{l} \alpha_i \ y_i \ \ x_i \qquad \text{-----------(6)}$$

$$\text{And} \qquad \partial L / \partial b_0 = 0$$

$$\text{i .e} \qquad \sum_{i=1}^{l} \alpha_i \ y_i = 0 \qquad \text{-----------(7)}$$

Substituting equation (6) and (7) in equation (5). We change the primal form into dual form.

$$L_d(\alpha) = \sum \alpha_i \ - 1/2 \sum_{i=1}^{l} \alpha_i \, \alpha_j \ y_i \, y_j \ x_i^T \, x_j \ \text{-------(8)}$$

In order to find the optimal hyper-plane, a dual lagrangian (L d) has to be maximized with respect to nonnegative i (i .e. i must be in the nonnegative quadrant) and with respect to the equality constraints as follow

$$\alpha_i \ \geq 0 \qquad , \ i = 1,2......l$$

$$\sum_{i=1}^{l} \alpha_i \ y_i = 0$$

Note that the dual Lagrangian $L_d(\alpha)$ is expressed in terms of training data and depends only on the scalar products of input patterns $(x_i T x_j)$.More detailed information on SVM can be found in Reference no.[1][2].

l be We consider data points of the form This can be written as

$$y_i(w.x_i b) \geq 1 \ \ , \ \ 1 \leq i \leq n - - - - - - (2)$$

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)..., (x_n, y_n)$$

. Figure.1 Maximum margin hyper-planes for a SVM trained with samples from two classes Where $y_n \ = \ 1/-1$, a constant denoting the class to which that point x_n belongs. n = number of sample. Each x_n is p-dimensional real vector. The scaling is important to guard against variable (attributes) with larger varience. We can view this Training data , by means of the dividing (or seperating) hyper-plane , which takes $w.x + b = 0$ Samples along the hyper-planes are called Support Vectors (SVs). A separating hyper-plane with the largest margin defined by $M = 2/|w|$ that is specifies support vectors means training data points closets to it. Which satisfy? —– (1) $y_j[w.T.x_j + b] = 1$, i =1 Where b is scalar and w is p-dimensional Vector. The vector w points perpendicular to the separating hyper-plane . Adding the offset parameter b allows us to increase the margin. Absence of b, the hyper-plane is forced to pass through the origin , restricting the solution.

References

[1] Coronary Artery Disease , $https : //medlineplus.gov/coronaryarterydisease.html$

[2] Coronary Artery Disease , $https : //familydoctor.org/condition/coronary - artery - disease - cad/?adfree = true$

[3] Can coronary artery disease (CAD) be prevented?, $https : //www.sharecare.com/health/circulatory - system - health/can - coronary - artery - disease - cad - be - prevented$

[4] Statistics About CAD, $https : //www.atrainceu.com/course - module - short - view/1711371 - 102_coronary - artery - disease - cad - module - 02$

[5] $https : //en.wikipedia.org/wiki/Coronary_artery_disease$

[6] $https : //www.heart.org/en/health - topics/consumer - healthcare/what - is - cardiovascular - disease/coronary - artery - disease$

[7] $KaggleHeartDiseaseUCIInternet : https : //www.kaggle.com/ronitf/heart - disease - uci, [Mar.16, 2019]$

[8] Mayoclinic Coronary artery disease - Symptoms and causes Internet:$https : //www.mayoclinic.org/diseases - conditions/coronary - artery - disease/symptoms - causes/syc - 20350613, [Mar.25, 2019]$

[9] Wikipedia- Multilayer perceptron Internet:$https : //en.wikipedia.org/wiki/Multilayer_perceptron, [Apr.3, 2019]$

Unused “captionsetup[1] on input line AtEndDocument