**SUMMER\_PROJECT - TEAM 10**

**DIABETES CLASSIFICATION DOCUMENTATION**

**MODULE IMPLEMENTED :**

I implemented the module **Dataset Splitting and Model Building**.

**DATA SPLITTING INTUITION**

* **Data Splitting** involves dividing the data for **training** and **testing** purpose .The machine learning model learns from the training data and then classifies the testing data accurately.
* For eg the **Decision tree classifier** algorithm, builds the **decision tree** with the help of the training data ,similarly the **Naïve Bayes algorithm** creates the **multilevel dictionary** which contains the **probability of each classes and feature** in it using training data
* The testing data is used to test the trained or validated model which gives an accuracy using evaluation metrics.
* In **Decision Tree classifier** the testing data can be used on the tree that was built with the help of training data and classifiy accordingly,Similarly in **Naïve Bayes** with help of the **multilevel dictionary** that consist **class and feature probability** it can used to classify the class with gives the highest probability when testing data is applied on it.

**DATA SPLITTING ALGORITHM**

* We split the given dataset into **training data** and **testing data** with help of the **skearn** library under **model\_selection** function **train\_test\_split(input\_features , outcome)** is used
* We can specify the size of the training and testing data using **test\_size** parameter.By default it is Training data : 75 % and Testing data : 25%.
* We specified **random state** paramter indicates that same testing and trainind data is acquired when we run it multiple times.
* It return four list **X\_train , X\_test , Y\_train , Y\_test,**

**X\_train** – training dataset with input / independent variables

**X\_test** – testing dataset with input/independent varaibles

**Y\_train** – training dataset with output / dependent varaible

**Y\_test** – Once we predict or classify the X\_test we compare with this to check the performance of the model

**DATA STRUCTURE FOR SPLITTING :**

1 ) List 2 )DataFrame in pandas

Text

Description automatically generated

**MODEL BUILDING:**

**1 ) DECISION TREE INTUITION:**

The **decision tree classifier** comes under **supervised machine learning** model that can be used **for prediction and classification**(Binary or multiclass classification) problems.The model predicts the output by the following one of the path/branch of the tree based on the condition it holds true. A decision tree is **built top-down** from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous).

From the outcome after a split we get the number of values that belong to the respective classes.Keeping this as parent node it can be splitted by any one of the features again and this goes on untill base consition are fulfilled.We basically need to split the parent node by a given feature such that after splitting the tree it must give a higher accuracy when compared with other features

**Deceiding the feature to split on :**

* Gini index
* Entropy
* Information gain

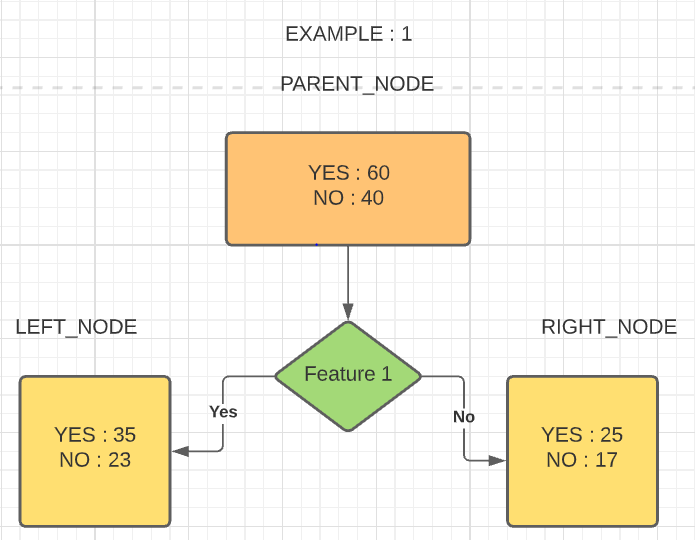
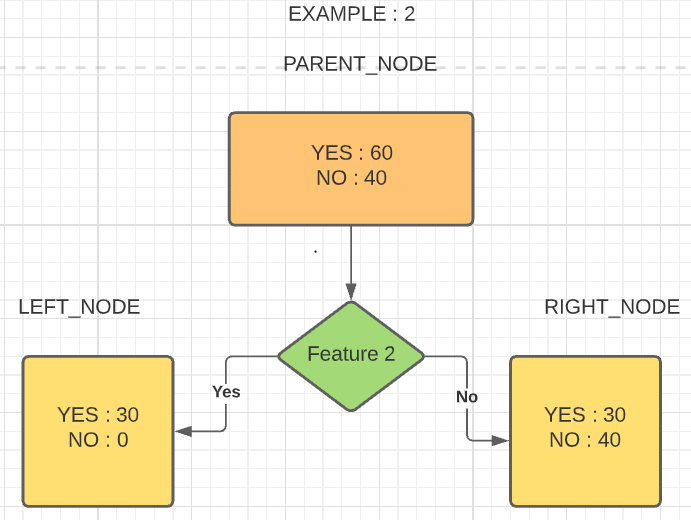
These tree metrics decide which is the best feature to split tree

**TREE METRICS :**

* **Gini Index:**
* The gini index calculates the amount of **probability** of a specific feature that is classified as **incorrectly .**

The gini index formula is given Gini Index = **1 – )2**

* Pi - Probability of ith class =
* The summation goes on for c number of classes
* The gini index value ranges from **0 to 1**
* Consider a problem in which the output is to classify the input variable to any one of the classes Yes , No.Initially the outcome values are as shown in parent node

**EXAMPLE : 1**

* The gini index of the left node **=** **1 – ((35/58)2  + (23/58)2) = 0.49**
* The gini index of the right node = **1 – ((25/42)2 + (17/42)2) = 0.49**

**EXAMPLE :2**

* The gini index of the left node **=** **1 – ((30/30)2  + (0/30)2) = 0**
* The gini index of the right node = **1 – ((30/70)2 + (40/70)2) = 0.48**

The weighted average of both the node is given by

+

The average in Example 1 is **= (58 / 100) \*0.49 + (42/100) \* 0.49 = 0.48**

The average in Example 2 is **= (30 / 100) \*0 + (70/100) \* 0.48 = 0.33**

* The node that only contains the domination of exactly one feature is called as **pure node**
* The gini index of **the pure node** is always 1
* The gini index is **0.5** when values are distributed **equally** to all classes in a node
* The feature which **less gini index is best** one to split
* By comparing the gini index by splitting the tree with feature 1 and feature 2 ,splitting the tree with feature 2 is better as the impurity is less when compared to feature 1
* **ENTROPY:**
* The entropy gives the measure of **uncertainity , disorder , impurity** in a node
* The entropy is given by the formula **Entropy = \* log(pi))**
* Pi - Probability of ith class =
* The summation goes on for c number of classes
* Conside the same above examples
* **Example 1**
* The entropy of the left node is = - **35/58 \* log(35/58) – 23/58 \* log(23/58)**

**= 0.96**

* The entropy of the right node is = -**25/42 \* log(25/42) - 17/42 \* log(42)**

= **0.97**

* **Example 2**
* The entropy of the left node is = - **30/30 \* log(30/30) – 0/30 \* log(0/30)**

**= 0**

* The entropy of the right node is = -**30/70 \* log(30/70) - 40/70 \*log(40/70)**

= **0.98**

* The entropy of a **pure node** is **0**
* The entropy is **1** when the features are **equally divided**
* The weighted average of both the node is given by

+

The average entropy in Example 1 is = (58 / 100) \*0.96 + (42/100) \* 0.97 = **0.96**

The average entropy in Example 2 is = (30 / 100) \*0 + (70/100) \* 0.98 = **0.68**

* **INFORMATION GAIN :**
* The information gain is based on the **decrease in entropy** after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the **highest** information gain (i.e., the most homogeneous branches).
* **Maximising (Entropy\_before\_split – Entropy\_After\_split)** is information gain as we are going towards more pure or homogeneous nodes. We take weighted average of the entropies of two nodes to get the resultant entropy at the level
* The information gain **= E(old) – E(new)**

Consider the above two examples :

The Entropy before the split is = **-60/100 \* log2(60/100) – 40/100 \* log2(40/100)**

**= 0.97**

The information gain in Example 1 is = **0.97 – 0.96 = 0.01**

The information gain in Example 2 is = **0.97 – 0.68 = 0.29**

* Thus splitting the tree with feature is the best split as we gain more informatio when compared with splitting with feature 1

**DECISION TREE ALGORITHM : (Building the tree internal implementation)**

**STEP 1** : Start with the parent node that contains the class count from outcome column

**STEP 2** : Split the node with for each feature

**STEP 3** : Check the feature with maximum information gain / minimum gini index and split the node accordingly into 2 other node in which one node contains the count of each classses that hold true whereas the other node contains the class count that holds false

**STEP 4**: Recursively call for both nodes

**STEP 5** : The base conditions are :

* Stop and return when the node is pure
* Stop and return when there is no featue left out to split further

**STEP 6 :** For each data point in the testing dataset it go goes through branches by the tree built by the above algorithm by splliting via each feature according to the condition and predict the class by taking the majority from the leaf nodes

**ALGORITHM (as in code) :**

**STEP 1:** Import the library DecisionTreeClassifier from sklearn.tree

**STEP 2 :** Create an instance of the class DecisionTreeClassifier as decisionTree

**STEP 3:** Train the model using the fit function which accepts two parameters X\_train , Y\_train.

**STEP 4 :** Classify the data using the predict function which accepts a X\_test as paramter and store in y\_pred\_decision\_tree to compare it with Y\_test

**DATA STRUCTURE (Internally Used) :**

1. N-ary Tree
2. DataFrame
3. numpy array
4. List

**DATA STRUCTURE (used in code) :**

1. DataFrame in pandas
2. List

Text

Description automatically generated

**2)NAÏVE BAYES INTUITION :**

**Naïve Bayes classifier** is **probabilistic** based machine learning algorithm which works based on **Bayes theorem**

Bayes theorem states that **probability of dependent event** is given by

**P(A | B)** =

* **P(A | B)** – is a **conditional probability** that the **event A** occurring given that **event B** has already occurred
* **P(B | A)** - is a **conditional probability** that the **event B** occurring given that **event A** has already occurred
* **P(B)** , **P(A)** are the probabilities of **event A , B** occurring independently each other

In general ,suppose there are **n features** independent variable(features) denoted by **X** and the outcome varaible denoted by **Y** and there are **k** different classification then,

**P(Y = ai | X = x)** =

Where **ai** refers to the ith class and i ranges from 1 to k.We must find the **maximum probability** amoung k classes for each dataset points and classify accordingly.

**P(Y = ai) =**

To find **P(X = x)** is why it called as **Naïve** Bayes

The Naïve Bayes is **Naïve** because it makes a strong assumption that all the **features are independent** of each other

The probability of independent events say **P(A ∩ B) = P(A) \*P(B)**

Thus we apply the same to the term **P(X = x)** it becomes as

**P(X = x) = (P(X = xi1 | Y = aj) \* P(x = xi2 | Y = aj) ……\* P(x = xiN | Y = aj)) \* P(Y = aj)**

**Xi**denotes **the ith data point** and **Xni**denotes the **nth feature** in ith data point in the dataset

We can remove the denominator as it a **constant value** for all the i data points in the dataset

**P(y = ai | X = x)** =

where j refers to the j th classification class and ranges form 1 to k classes

**P(X = xi | Y = aj)** =

**NAÏVE BAYES ALGORITHM (internal implementation) :**

**STEP 1 :** We create **Multilevel dictionary** in which the first level store all type of

**classes (c1 , c2 …… ck) as keys**

**STEP 2:** As values for above keys we store another dictionary where the keys are

**Features(x1,x2 ….. xn)**

**STEP 3 :** As values for above keys are another dictionary where it store the **distinct values** as keys and their **frequencies** as values as depicted in below diagram

Diagram, schematic

Description automatically generated

**STEP 4 :** Each data point in the testing dataset we go through each classes and find out the probability of each feature in the class cj and pick the class which shows the highest probability amoung them and classify xj accordingly.

**ALGORITHM (as in code):**

**STEP 1:** Import the library GaussianNB from sklearn.naive\_bayes

**STEP 2 :** Create an instance of the class GaussianNB as naïve\_bayes

**STEP 3:** Train the model using the fit function which accepts two parameters X\_train , Y\_train

**STEP 4 :** Classify the data using the predict function which accepts a X\_test as paramter and store in y\_pred\_naive\_bayes to compare it with Y\_test

**DATA STRUCTURE (Used internally)**

1) Multilevel Dictionary

2) numpy array

**DATA STRUCTURE (used in code) :**

1)DataFrame in pandas

2) numpy array

Graphical user interface, text, website

Description automatically generated

**REFERENCES :**

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

<https://scikit-learn.org/stable/modules/tree.html>

<https://scikit-learn.org/stable/modules/naive_bayes.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html>

<https://scikitlearn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>