## DATA SCIENCE PROJECT

# REPORT ON
# BANKNOTE AUTHENTICATION

# Introduction

## TOPIC:

## BankNote Authentication

Banknotes are currencies used by any nation to carry-out financial activities and are every countries asset which every nation wants it (bank-note) to be genuine. Lot of miscreants induces fake notes into the market which resemble exactly the original note. Hence, there is a need for an efficient authentication system which predicts accurately whether the given note is genuine or not.

## DATA SET:

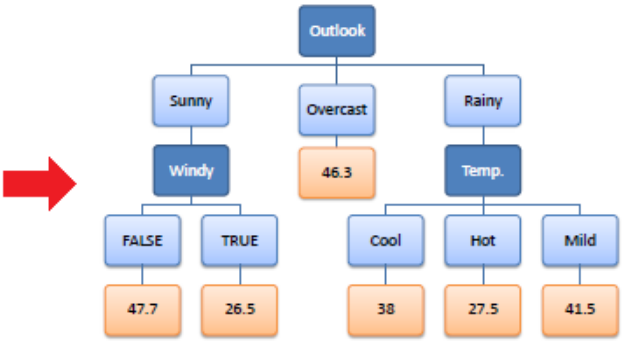
| variance | skewness | curtosis | entropy | class |
|---|---|---|---|---|
| 3.6216 | 8.6661 | -2.8073 | -0.44699 | 0 |
| 4.5459 | 8.1674 | -2.4586 | -1.4621 | 0 |
| 3.866 | -2.6383 | 1.9242 | 0.10645 | 0 |
| 3.4566 | 9.5228 | -4.0112 | -3.5944 | 0 |
| 0.32924 | -4.4552 | 4.5718 | -0.9888 | 0 |
| 4.3684 | 9.6718 | -3.9606 | -3.1625 | 0 |

## Algorithms used

# Decision tree regression

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

| Predictors | | | | Target |
|---|---|---|---|---|
| Outlook | Temp. | Humidity | Windy | Hours Played |
| Rainy | Hot | High | False | 25 |
| Rainy | Hot | High | True | 30 |
| Overcast | Hot | High | False | 48 |
| Sunny | Mild | High | False | 46 |
| Sunny | Cool | Normal | False | 62 |
| Sunny | Cool | Normal | True | 23 |
| Overcast | Cool | Normal | True | 43 |
| Rainy | Mild | High | False | 36 |
| Rainy | Cool | Normal | False | 38 |
| Sunny | Mild | Normal | False | 48 |
| Rainy | Mild | Normal | True | 48 |
| Overcast | Mild | High | True | 52 |
| Overcast | Hot | Normal | False | 44 |
| Sunny | Mild | High | True | 30 |



# Decision tree algorithm

The core algorithm for building decision trees called ID3 by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with Standard Deviation Reduction.

### Standard Deviation

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). We use standard deviation to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous its standard deviation is zero.

- Standard deviation for **one** attribute:

| Hours Played |
|---|
| 25 |
| 30 |
| 46 |
| 45 |
| 52 |
| 23 |
| 43 |
| 35 |
| 38 |
| 46 |
| 48 |
| 52 |
| 44 |
| 30 |

$$\textbf{\textit{Count}} = n = 14$$

$$\textbf{\textit{Average}} = \bar{x} = \frac{\sum x}{n} = 39.8$$

$$\textbf{\textit{Standard Deviation}} = S = \sqrt{\frac{\sum (x - \bar{x})^2}{n}} = 9.32$$

$$\textbf{\textit{Coeffeicient of Variation}} = CV = \frac{S}{\bar{x}} * 100\% = 23\%$$

- **Standard deviation for two attributes (target and predictor):**

$$S(T, X) = \sum_{c \in X} P(c) S(c)$$

| | | Hours Played (StDev) | Count |
|---|---|---|---|
| Outlook | Overcast | 3.49 | 4 |
| | Rainy | 7.78 | 5 |
| | Sunny | 10.87 | 5 |
| | | | 14 |

**S**(Hours, Outlook) = **P**(Sunny)*S(Sunny) + **P**(Overcast)*S(Overcast) + **P**(Rainy)*S(Rainy)

= (4/14)*3.49 + (5/14)*7.78 + (5/14)*10.87

= 7.66

## Standard Deviation Reduction

The standard deviation reduction is based on the decrease in standard deviation after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest standard deviation reduction (i.e., the most homogeneous branches).

**Step 1:**The standard deviation of the target is calculated.

Standard deviation (Hours Played) = 9.32

**Step 2:** The dataset is then split on the different attributes. The standard deviation for each branch is calculated. The resulting standard deviation is subtracted from the standard deviation before the split. The result is the standard deviation reduction.

| | | Hours Played (StDev) |
|---|---|---|
| Outlook | Overcast | 3.49 |
| | Rainy | 7.78 |
| | Sunny | 10.87 |
| SDR=1.66 | | |

| | | Hours Played (StDev) |
|---|---|---|
| Temp. | Cool | 10.51 |
| | Hot | 8.95 |
| | Mild | 7.65 |
| SDR=0.17 | | |

| | | Hours Played (StDev) |
|---|---|---|
| Humidity | High | 9.36 |
| | Normal | 8.37 |
| SDR=0.28 | | |

| | | Hours Played (StDev) |
|---|---|---|
| Windy | False | 7.87 |
| | True | 10.59 |
| SDR=0.29 | | |

$$SDR(T,X) = S(T) - S(T,X)$$

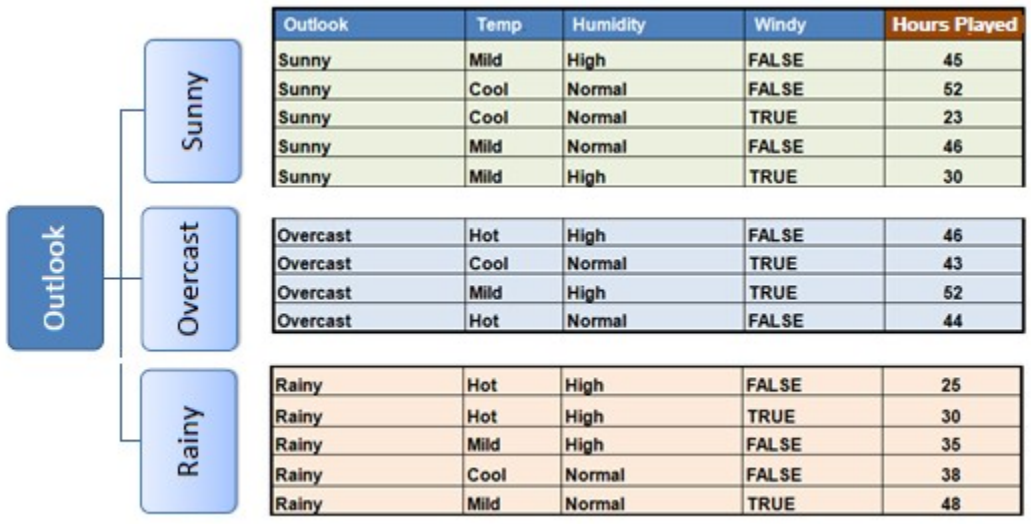
**SDR**(Hours , Outlook) = **S**(Hours ) – **S**(Hours, Outlook)

= 9.32 – 7.66 = 1.66

**Step 3:** The attribute with the largest standard deviation reduction is chosen for the decision node.

| ★ | | Hours Played (StDev) |
|---|---|---|
| Outlook | Overcast | 3.49 |
| | Rainy | 7.78 |
| | Sunny | 10.87 |
| SDR=1.66 | | |

**Step 4a:** The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches, until all data is processed.




| Outlook | Temp | Humidity | Windy | Hours Played |
|---|---|---|---|---|
| Sunny | Mild | High | FALSE | 45 |
| Sunny | Cool | Normal | FALSE | 52 |
| Sunny | Cool | Normal | TRUE | 23 |
| Sunny | Mild | Normal | FALSE | 46 |
| Sunny | Mild | High | TRUE | 30 |

| | | | | |
|---|---|---|---|---|
| Overcast | Hot | High | FALSE | 46 |
| Overcast | Cool | Normal | TRUE | 43 |
| Overcast | Mild | High | TRUE | 52 |
| Overcast | Hot | Normal | FALSE | 44 |

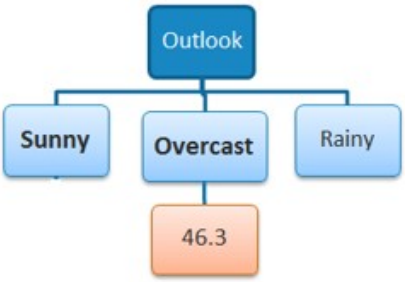
| | | | | |
|---|---|---|---|---|
| Rainy | Hot | High | FALSE | 25 |
| Rainy | Hot | High | TRUE | 30 |
| Rainy | Mild | High | FALSE | 35 |
| Rainy | Cool | Normal | FALSE | 38 |
| Rainy | Mild | Normal | TRUE | 48 |

In practice, we need some termination criteria. For example, when coefficient of deviation (CV) for a branch becomes smaller than a certain threshold (e.g., 10%) and/or when too few instances (n) remain in the branch (e.g., 3).

**Step 4b:** "Overcast" subset does not need any further splitting because its CV (8%) is less than the threshold (10%). The related leaf node gets the average of the "Overcast" subset.

Outlook - Overcast

| | | Hours Played (StDev) | Hours Played (AVG) | Hours Played (CV) | Count |
|---|---|---|---|---|---|
| Outlook | Overcast | 3.49 | 46.3 | 8% | 4 |
| | Rainy | 7.78 | 35.2 | 22% | 5 |
| | Sunny | 10.87 | 39.2 | 28% | 5 |




**Step 4c:** However, the "Sunny" branch has an CV (28%) more than the threshold (10%) which needs further splitting. We select "Windy" as the best best node after "Outlook" because it has the largest SDR.

Outlook - Sunny

| Temp | Humidity | Windy | Hours Played |
|---|---|---|---|
| Mild | High | FALSE | 45 |
| Cool | Normal | FALSE | 52 |
| Cool | Normal | TRUE | 23 |
| Mild | Normal | FALSE | 46 |
| Mild | High | TRUE | 30 |
| | | | S = 10.87 |
| | | | AVG = 39.2 |
| | | | CV = 28% |

| | | Hours Played (StDev) | Cou |
|---|---|---|---|
| Temp | Cool | 14.50 | 2 |
| | Mild | 7.32 | 3 |

SDR = 10.87-((2/5)*14.5 + (3/5)*7.32) = **0.678**

| | | Hours Played (StDev) | Cou |
|---|---|---|---|
| Humidity | High | 7.50 | 2 |
| | Normal | 12.50 | 3 |

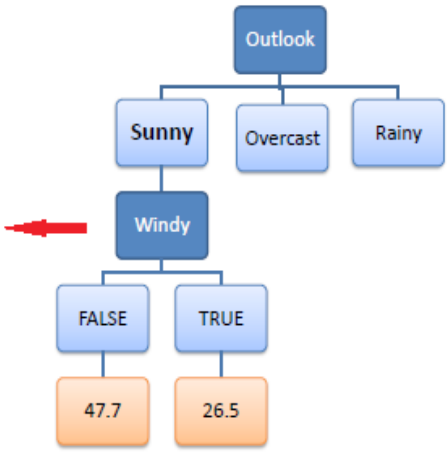
SDR = 10.87-((2/5)*7.5 + (3/5)*12.5) = **0.370**

| | | Hours Played (StDev) | Cou |
|---|---|---|---|
| Windy | False | 3.09 | 3 |
| | True | 3.50 | 2 |

SDR = 10.87-((3/5)*3.09 + (2/5)*3.5) = **7.62**

Because the number of data points for both branches (FALSE and TRUE) is equal or less than 3 we stop further branching and assign the average of each branch to the related leaf node.

| Temp | Humidity | Windy | Hours Played |
|---|---|---|---|
| Mild | High | FALSE | 45 |
| Cool | Normal | FALSE | 52 |
| Mild | Normal | FALSE | 46 |
| Cool | Normal | TRUE | 23 |
| Mild | High | TRUE | 30 |




**Step 4d:** Moreover, the "rainy" branch has an CV (22%) which is more than the threshold (10%). This branch needs further splitting. We select "Windy" as the best node because it has the largest SDR.

## Outlook - Rainy

| Temp | Humidity | Windy | Hours Played |
|---|---|---|---|
| Hot | High | FALSE | **25** |
| Hot | High | TRUE | **30** |
| Mild | High | FALSE | **35** |
| Cool | Normal | FALSE | **38** |
| Mild | Normal | TRUE | **48** |
| | | | **S** = 7.78 |
| | | | **AVG** = 35.2 |
| | | | **CV** = 22% |

| | | Hours Played (StDev) | Count |
|---|---|---|---|
| **Temp** | **Cool** | 0 | 1 |
| | **Hot** | 2.5 | 2 |
| | **Mild** | 6.5 | 2 |

SDR = 7.78 - ((1/5)*0+(2/5)*2.5 + (2/5)*6.5) = **4.18**

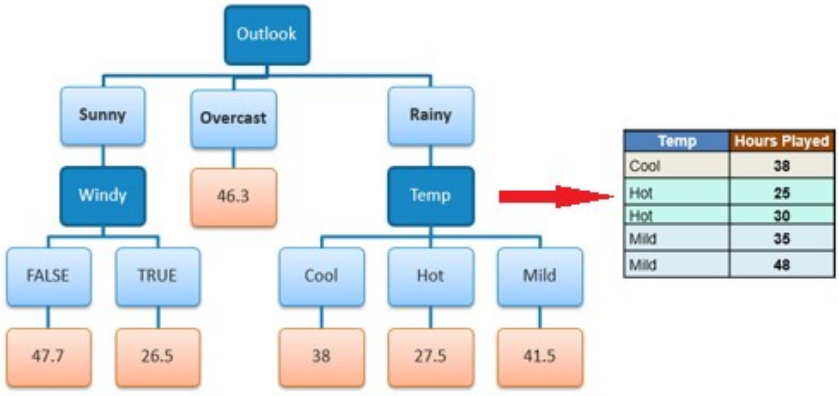
| | | Hours Played (StDev) | Count |
|---|---|---|---|
| **Humidity** | **High** | 4.1 | 3 |
| | **Normal** | 5.0 | 2 |

SDR = 7.78 - ((3/5)*4.1 + (2/5)*5.0) = **3.32**

| | | Hours Played (StDev) | Count |
|---|---|---|---|
| **Windy** | **False** | 5.6 | 3 |
| | **True** | 9.0 | 2 |

SDR = 7.78 - ((3/5)*5.6 + (2/5)*9.0) = **0.82**

Because the number of data points for all three branches (Cool, Hot and Mild) is equal or less than 3 we stop further branching and assign the average of each branch to the related leaf node.




When the number of instances is more than one at a leaf node we calculate the average as the final value for the target

## Decision tree types

Decision trees used in data mining are of two main types:

- Classification tree analysis is when the predicted outcome is the class (discrete) to which the data belongs.
- Regression tree analysis is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

The term Classification And Regression Tree (CART) analysis is an umbrella term used to refer to both of the above procedures, first introduced by Breiman et al. in 1984. Trees used for regression and trees used for classification have some similarities - but also some differences, such as the procedure used to determine where to split.

Some techniques, often called *ensemble* methods, construct more than one decision tree:

- Boosted trees Incrementally building an ensemble by training each new instance to emphasize the training instances previously mis-modeled. A typical example is AdaBoost. These can be used for regression-type and classification-type problems.
- Bootstrap aggregated (or bagged) decision trees, an early ensemble method, builds multiple decision trees by repeatedly resampling training data with replacement, and voting the trees for a consensus prediction.
    - A random forest classifier is a specific type of bootstrap aggregating
- Rotation forest – in which every decision tree is trained by first applying principal component analysis (PCA) on a random subset of the input features.

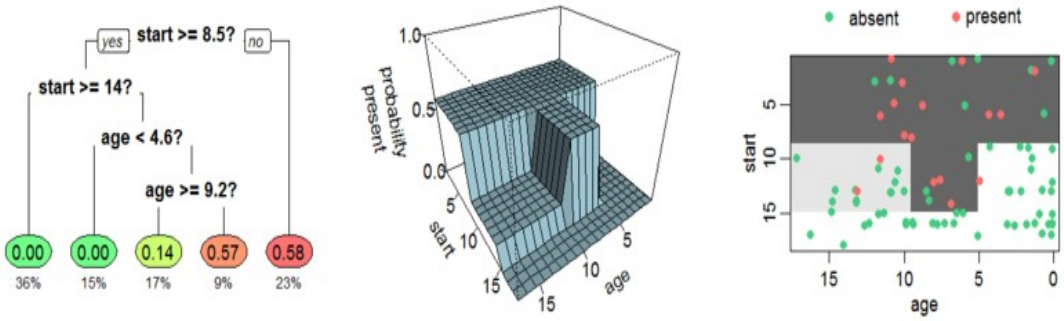
A special case of a decision tree is a decision list, which is a one-sided decision tree, so that every internal node has exactly 1 leaf node and exactly 1 internal node as a child (except for the bottommost node, whose only child is a single leaf node). While less expressive, decision lists are arguably easier to understand than general decision trees

due to their added sparsity, permit non-greedy learning methods and monotonic constraints to be imposed.

**Decision tree learning** is the construction of a decision tree from class-labeled training tuples. A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node.

There are many specific decision-tree algorithms. Notable ones include:

- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)[4]
- Chi-square automatic interaction detection (CHAID). Performs multi-level splits when computing classification trees.[12]
- MARS: extends decision trees to handle numerical data better.
- Conditional Inference Trees. Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning




**An example tree which estimates the probability of kyphosis after surgery, given the age of the patient and the vertebra at which surgery was started. The same tree is shown in three different ways. Left The colored leaves show the probability of kyphosis after surgery, and percentage of patients in the leaf. Middle The tree as a perspective plot. Right Aerial view of the middle plot. The probability of kyphosis after surgery is higher in the darker areas**

# Naive Bayes algorithm

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

## 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:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where A and B are events and P(B) ? 0.

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence.
- P(A) is the priori of A (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 B).
- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

## Gaussian naive Bayes

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. For example, suppose the training data contains a continuous attribute, $x$ We first segment the data by the class, and then compute the mean and variance of $x$ in each class. Let $\mu_k$ be the mean of the values in $x$ associated with class $C_k$, and let $\sigma_k^2$ be the Bessel corrected variance of the values in $x$ associated with class $C_k$. Suppose we have collected some observation value $v$ Then, the probability distribution of $v$ given a class $p(x = v \mid C_k)$ can be computed by plugging $v$ into the equation for a normal distribution parameterized by $\mu_k$ and $\sigma_k^2$

That is,

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

## Types of Naive Bayes Classifier:

### Multinomial Naive Bayes:

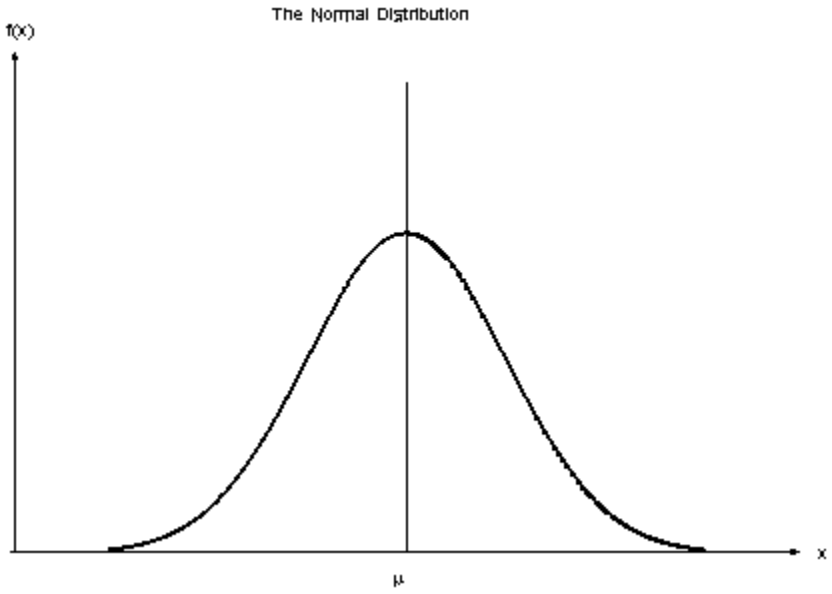
This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

### Bernoulli Naive Bayes:

This is similar to the multinomial naive bayes but the predictors are boolean variables. 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.

### Gaussian Naive Bayes:

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.



# Implementation of decision tree regression

## IMPORTING REQUIRED LIBRARIES

Before we can import our dataset and perform analysis, we need to import a few libraries. The following script is used to import libraries:

```
[ ]: import pandas as pd
     import numpy as np
     import seaborn as sns
```

## LOADING THE DATASET

Once we import the libraries, the next step is to load the dataset into our application. To do so, we used the “read_csv()” function of the Pandas library, which reads dataset that is in the CSV format

The following script loads the dataset into “banknote_dataset” data frame:

```
[3]: banknote_datadset = pd.read_csv('https://raw.githubusercontent.com/Kuntal-G/Machine-Learning/master/R-mach
```

## DATA ANALYSIS

To see how the dataset actually looks, we can use the “head()” function of the Pandas data frame:

The "head()" function returns the first five rows of the dataset as shown below:

```
[ ]: banknote_datadset.head()
```

| | variance | skew | curtosis | entropy | class |
|---|---|---|---|---|---|
| **0** | 3.62160 | 8.6661 | -2.8073 | -0.44699 | 0 |
| **1** | 4.54590 | 8.1674 | -2.4586 | -1.46210 | 0 |
| **2** | 3.86600 | -2.6383 | 1.9242 | 0.10645 | 0 |
| **3** | 3.45660 | 9.5228 | -4.0112 | -3.59440 | 0 |
| **4** | 0.32924 | -4.4552 | 4.5718 | -0.98880 | 0 |

To see the statistical details of the data, the “describe()” function can be used, which returns the mean, count, standard deviation, quartile information and maximum values for each column

```
[ ]: banknote_datadset.describe()
```

| | variance | skew | curtosis | entropy | class |
|---|---|---|---|---|---|
| count | 1372.000000 | 1372.000000 | 1372.000000 | 1372.000000 | 1372.000000 |
| mean | 0.433735 | 1.922353 | 1.397627 | -1.191657 | 0.444606 |
| std | 2.842763 | 5.869047 | 4.310030 | 2.101013 | 0.497103 |
| min | -7.042100 | -13.773100 | -5.286100 | -8.548200 | 0.000000 |
| 25% | -1.773000 | -1.708200 | -1.574975 | -2.413450 | 0.000000 |
| 50% | 0.496180 | 2.319650 | 0.616630 | -0.586650 | 0.000000 |
| 75% | 2.821475 | 6.814625 | 3.179250 | 0.394810 | 1.000000 |
| max | 6.824800 | 12.951600 | 17.927400 | 2.449500 | 1.000000 |

## DATA PREPROCESSING AND TRAINING/TESTING DATASET

```
5]: X = df.drop('class', axis = 1)
    X[0:5]
```

5]:

| | variance | skew | curtosis | entropy |
|---|---|---|---|---|
| 0 | 3.62160 | 8.6661 | -2.8073 | -0.44699 |
| 1 | 4.54590 | 8.1674 | -2.4586 | -1.46210 |
| 2 | 3.86600 | -2.6383 | 1.9242 | 0.10645 |
| 3 | 3.45660 | 9.5228 | -4.0112 | -3.59440 |
| 4 | 0.32924 | -4.4552 | 4.5718 | -0.98880 |

```
4]: Y = df['class']
    Y[0:5]
```

```
4]: 0    0.0
    1    0.0
    2    0.0
    3    0.0
    4    0.0
    Name: class, dtype: object
```

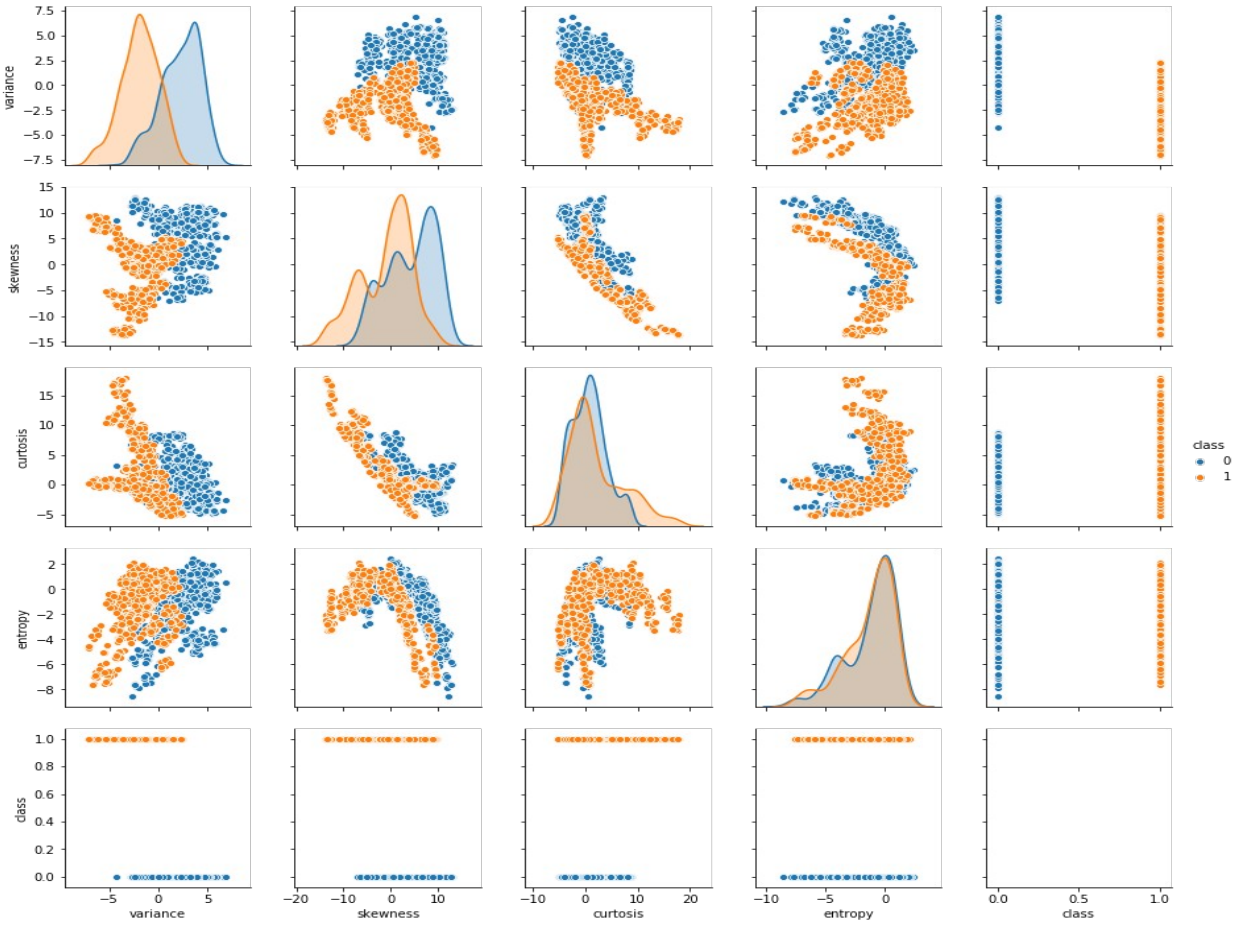
```
0]: x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=3)
```

## BUILDING MODEL AND CALCULATING MODEL'S ACCURACY

```
.0]: bank_tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
     bank_tree

.0]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                            splitter='best')

.2]: bank_tree.fit(x_train, y_train)

.2]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                            splitter='best')

.3]: pred_tree = bank_tree.predict(x_test)
     pred_tree[0:5]

.3]: array(['1.0', '0.0', '0.0', '1.0', '1.0'], dtype=object)

'9]: from sklearn import metrics
     import matplotlib.pyplot as plt
     print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, pred_tree))

     DecisionTrees's Accuracy:  0.9611650485436893
```

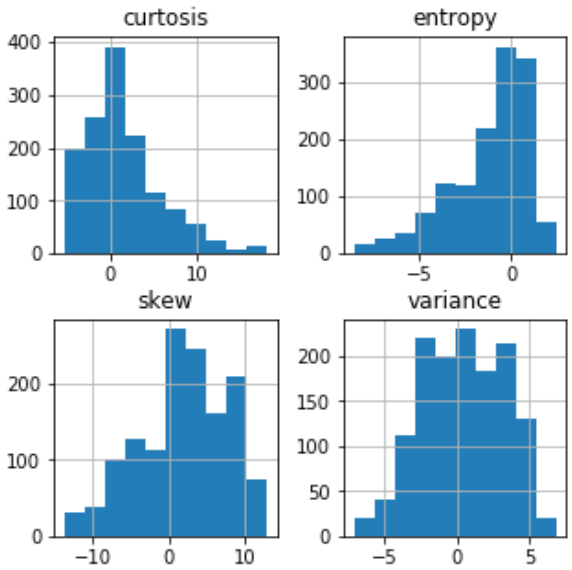
## DATA VISUALIZATION

```
[ ]: sns.pairplot(banknote_datadset)
```



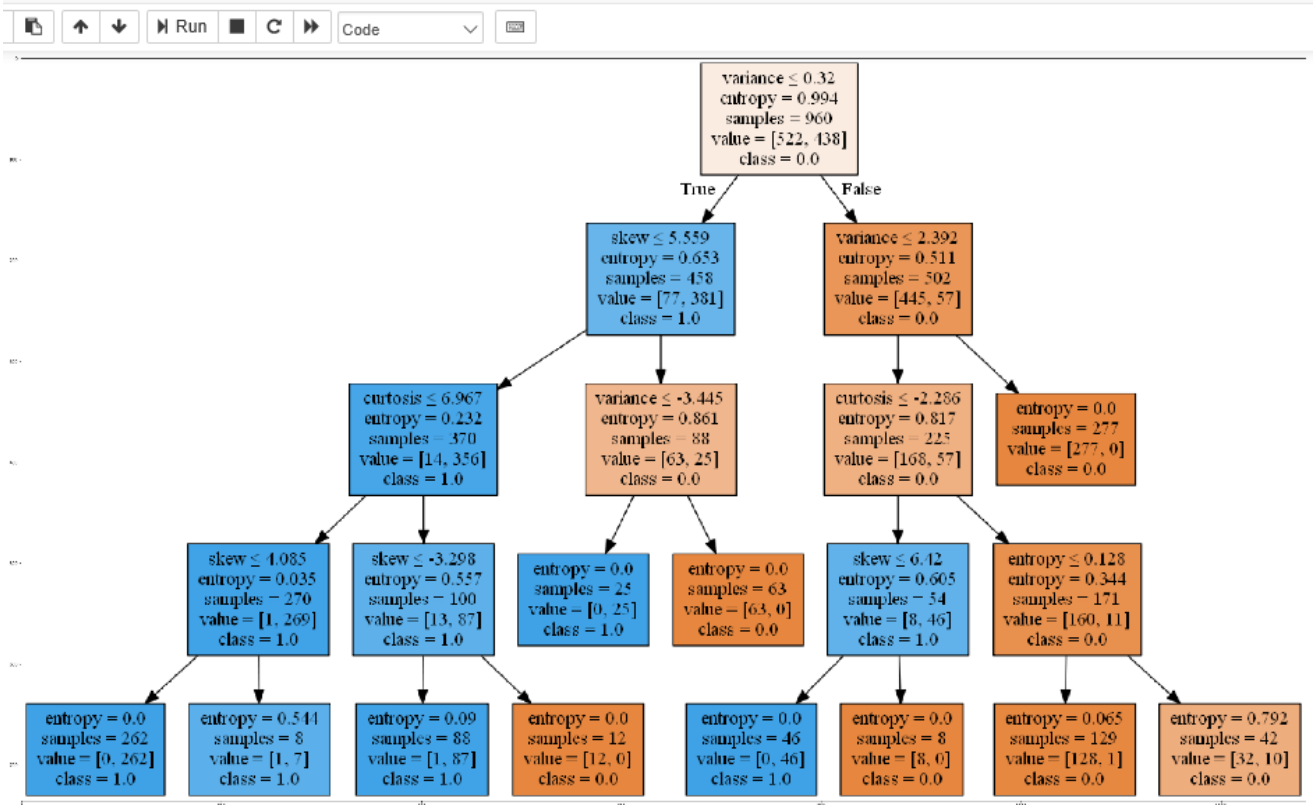

**It is visible from the output that entropy and variance have a slight linear correlation. Similarly, there is an inverse linear correlation between the curtosis and skew. Finally, we can see that the values for curtosis and entropy are slightly higher for real banknotes, while the values for skew and variance are higher for the fake banknotes**

[128]:
```
banknote_dataset.hist(figsize = (5,5))
plt.show()
```




01]:
```
col_names = banknote_dataset.drop('class', axis = 1).columns.tolist()

plt.figure(figsize = (10,3))
i = 0
for col in col_names:
    plt.subplot(1,4,i+1)
    plt.grid(True, alpha =0.5)
    sns.kdeplot(banknote_dataset[col][banknote_dataset['class'] ==0], label = 'Fake note')
    sns.kdeplot(banknote_dataset[col][banknote_dataset['class'] ==1], label = 'Original note')
    plt.title('Class vs ' + col)
    plt.tight_layout()
    i+=1
plt.show()
```

76]:
```
dot_data = StringIO()
filename = "banktree.png"
featureNames = df.columns[0:4]
targetNames = df["class"].unique().tolist()
out=tree.export_graphviz(bank_tree,feature_names=featureNames, out_file=dot_data, class_names= np.unique(y_train), filled=True,
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(50, 60))
plt.imshow(img,interpolation='nearest')
```



# Implementation of Naive bayes algorithm

## IMPORTING REQUIRED LIBRARIES

Before we can import our dataset and perform analysis, we need to import a few libraries. The following script is used to import libraries:

```
[ ]: import importlib
     import matplotlib.pyplot as plt
     import matplotlib.patches as mpatches
     import pandas as pan
     import numpy as np
     import gnb
```

## LOADING THE DATASET

Once we import the libraries, the next step is to load the dataset into our application. To do so, we used the “read_csv()” function of the Pandas library, which reads dataset that is in the CSV format

The following script loads the dataset into “banknote_dataset” data frame:

```
[3]: banknote_datadset = pd.read_csv('https://raw.githubusercontent.com/Kuntal-G/Machine-Learning/master/R-mach
```

## DATA ANALYSIS

To see how the dataset actually looks, we can use the “head()” function of the Pandas data frame:The "head()" function returns the first five rows of the dataset as shown below:

```
[ ]: banknote_datadset.head()
```

| | variance | skew | curtosis | entropy | class |
|---|---|---|---|---|---|
| 0 | 3.62160 | 8.6661 | -2.8073 | -0.44699 | 0 |
| 1 | 4.54590 | 8.1674 | -2.4586 | -1.46210 | 0 |
| 2 | 3.86600 | -2.6383 | 1.9242 | 0.10645 | 0 |
| 3 | 3.45660 | 9.5228 | -4.0112 | -3.59440 | 0 |
| 4 | 0.32924 | -4.4552 | 4.5718 | -0.98880 | 0 |

To see the statistical details of the data, the “describe()” function can be used, which returns the mean, count, standard deviation, quartile information and maximum values for each column

```
[ ]: banknote_datadset.describe()
```

| | variance | skew | curtosis | entropy | class |
|---|---|---|---|---|---|
| count | 1372.000000 | 1372.000000 | 1372.000000 | 1372.000000 | 1372.000000 |
| mean | 0.433735 | 1.922353 | 1.397627 | -1.191657 | 0.444606 |
| std | 2.842763 | 5.869047 | 4.310030 | 2.101013 | 0.497103 |
| min | -7.042100 | -13.773100 | -5.286100 | -8.548200 | 0.000000 |
| 25% | -1.773000 | -1.708200 | -1.574975 | -2.413450 | 0.000000 |
| 50% | 0.496180 | 2.319650 | 0.616630 | -0.586650 | 0.000000 |
| 75% | 2.821475 | 6.814625 | 3.179250 | 0.394810 | 1.000000 |
| max | 6.824800 | 12.951600 | 17.927400 | 2.449500 | 1.000000 |

## DATA PREPROCESSING AND TRAINING/TESTING DATASET

5]:
```
X = df.drop('class', axis = 1)
X[0:5]
```

5]:

| | variance | skew | curtosis | entropy |
|---|---|---|---|---|
| 0 | 3.62160 | 8.6661 | -2.8073 | -0.44699 |
| 1 | 4.54590 | 8.1674 | -2.4586 | -1.46210 |
| 2 | 3.86600 | -2.6383 | 1.9242 | 0.10645 |
| 3 | 3.45660 | 9.5228 | -4.0112 | -3.59440 |
| 4 | 0.32924 | -4.4552 | 4.5718 | -0.98880 |

4]:
```
Y = df['class']
Y[0:5]
```

4]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: class, dtype: object
```

0]:
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=3)
```

## BUILDING MODEL AND CALCULATING MODEL'S ACCURACY

```
In [10]: bank_gnb = GaussianNB()
         bank_gnb.fit(x_train, y_train)
```

```
Out[10]: GaussianNB(priors=None, var_smoothing=1e-09)
```

```
In [11]: y_pred = bank_gnb.predict(x_test)
         y_pred[0:10]
```

```
Out[11]: array([1, 0, 0, 1, 1, 1, 1, 0, 1, 1])
```

```
In [12]: from sklearn import metrics
         print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)*100)
```

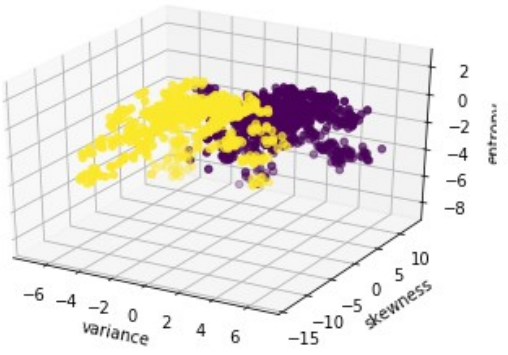
```
Gaussian Naive Bayes model accuracy(in %): 84.70873786407766
```

## DATA VISUALIZATION

```
[ ]: plt.plot(list_ratio,logistic_accuracy,color = 'g')
     plt.plot(list_ratio,gnb_accuracy,color = 'orange' )
     plt.xlabel('training_sample_ratio')
     plt.ylabel('validation accuracy in percentage')
     green_patch = mpatches.Patch(color='green', label='logistic regression')
     orange_patch = mpatches.Patch(color='orange', label='guassian naive bayes')
     plt.legend(handles=[green_patch,orange_patch])
     plt.show()
```

**This graph represents the curve of banknote authentication problem using Naive bayes algorithm and logistic regression**

```
]: fig = plt.figure()
   ax = fig.add_subplot(111, projection='3d')
   xs = df['variance']
   ys = df['skew']
   zs = df['entropy']
   ax.scatter(xs, ys, zs, c=df['class'])

   ax.set_xlabel('variance')
   ax.set_ylabel('skewness')
   ax.set_zlabel('entropy')

   plt.show()
```



In [16]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
xs = df['skew']
ys = df['curtosis']
zs = df['variance']
ax.scatter(xs, ys, zs, c=df['class'])

ax.set_xlabel('skew')
ax.set_ylabel('curtosis')
ax.set_zlabel('variance')

plt.show()
```