Binary Classification Experiment Using a Naive Bayes (NB) Classifier And Simple Neural Network-based (NN) Classifier.

Mohamed E. Eid, Mahmoud Magdy, Mohamed E. Ali, Abdullah M. Omran

Department of Systems and Biomedical Engineering, Faculty of Engineering, Cairo University

K E Y W O R D

Naive Bayes classifier
Outliers
Standardization
Statistical test
Conditional distributions
Prediction
Deep learning
Simple Neural Network

A B S T R A C T

This paper delves into the exploration and cleaning of data as part of a comprehensive analysis of two popular machine learning algorithms, the Naive Bayes (NB) classifier, and Neural Network, for binary classification tasks. A tabular dataset consisting of a mix of quantitative and categorical variables. Rigorous data cleaning techniques are employed to ensure the reliability and integrity of the dataset. The exploration phase involves examining the dataset's features and their distributions through the creation of histograms. This visualization aids in understanding the nature of the data and identifying potential patterns or irregularities. Additionally, a statistical test is conducted to determine if each feature adheres to a normal distribution, which is crucial for certain assumptions underlying the algorithms. Cleaning techniques are applied to address outliers. This process enhances the dataset's quality and reduces the potential for biased or erroneous results.

1. Introduction

Classifying data into two categories is a common task in machine learning, and many algorithms have been developed to do this. This paper explores how two popular classifiers—the Naive Bayes (NB) and Neural Network—perform on binary classification tasks.

The study employs a tabular dataset obtained from the Kaggle platform, chosen specifically for its relevance to binary classification problems. The dataset comprises both quantitative and categorical variables, making it suitable for evaluating the performance of different algorithms across diverse feature types. To ensure the integrity of the data, statistical methods are employed to identify and eliminate outliers that might affect the classification process.

Descriptive statistics are calculated to gain a comprehensive understanding of the dataset, including measures of central tendency and dispersion. Standardization techniques are applied to the features to facilitate fair comparison between the classifiers. The dataset is then divided into training and testing partitions, enabling the assessment of model performance on unseen data.

In order to gain insights into the data, histograms are plotted to visualize the distributions of individual features. Additionally, statistical tests are conducted to examine if the features follow a normal distribution. The conditional distributions of each feature with respect to the target classes are analyzed to uncover potential relationships and dependencies.

The Naive Bayes classifier is implemented from scratch and trained on the training data. The model's accuracy is evaluated by predicting the classifications of the test data. To benchmark the results, the performance of the NB classifier from standard Python packages is also assessed.

Furthermore, a simple Neural Network classifier is developed using gradient descent. The loss plot is examined to gauge the model's convergence, and the accuracy is calculated to measure its performance.

The paper employs the NB classifier, implementing it from scratch and training it on the cleaned training data. Predictions are made on the testing data, and the accuracy of the model is calculated. To establish a benchmark, the results obtained from the self-implemented NB classifier are compared with those derived from standard Python packages.

Additionally, a simple Neural Network classifier is developed using gradient descent. The loss plot, depicting the model's convergence during training, is examined, and the model's accuracy is computed to assess its performance.

Through an extensive exploration and cleaning process, this research aims to ensure data reliability and integrity, providing a solid foundation for the subsequent analysis of the NB classifier and Neural Network. By addressing outliers, assessing feature distributions, and employing standardization techniques, this study enhances the accuracy and effectiveness of the classification models, facilitating informed decision-making in binary classification tasks.

By comparing the results obtained from both classifiers providing valuable insights into their relative effectiveness for binary classification tasks. The findings will aid in determining the most suitable approach based on the characteristics of the dataset, contributing to the advancement of binary classification methodologies.

2 Steps

2.1 Importing data

Pandas is used mainly to read the data from Kaggle, since the data is (.csv) extension, We used the "**read_csv()**"method to import the data in form of a data frame. Any operation performed on the data frame is done through a function from the Pandas library.

This Data frame is used as it offers powerful and flexible advantages: (1) Easy data manipulation: It supports operations such as filtering, sorting, grouping, joining, and aggregating data. These operations can be performed quickly and efficiently, enabling complex data transformations and analysis tasks. (2) Data indexing and slicing: data Frames support powerful indexing and slicing operations. Rows and columns can be accessed using labels or numerical indices, allowing for easy data extraction and sub setting. This feature is particularly useful for selecting specific subsets of data for analysis or visualization. (3) Integration with other libraries: Pandas integrates well with other popular libraries in the Python ecosystem, such as NumPy, Matplotlib, and scikit-learn,

Here are some used methods: **Pd.read_csv()** : read the dataset into a Data frame ,**Df.describe()**: provide statistical information about each column ,Df.info(): provide us with the column names, Non-Null count , columns **Dtype ,Df.drop_duplicates()**: remove any duplicate rows in the **dataframe**., **Data.dropna()**: remove any row contains NA value.

2.2 Data cleaning

To clean the data, we need to remove Nan values, duplicate rows, wrong format data and most importantly the outliers. We didn't find any wrong format values or duplicate rows, so all we need to do is to remove outliers.

We used the interquartile method (**IQR**): (1) Calculate the first quartile (Q1) and third quartile (Q3) of the dataset. The first quartile is the median of the lower half of the data, while the third quartile is the median of the upper half. (2) Compute the interquartile range (IQR) by subtracting Q1 from Q3: IQR = Q3 - Q1. (3) Determine the lower bound as Q1 - (1.5 * IQR) and the upper bound as Q3 + (1.5 * IQR). (4)Identify any data points that fall below the lower bound or above the upper bound. These points are considered outliers.

These identified data points can be either removed from data or replaced with other reasonable values (i.e. the mean value), we chose to remove them using **Data.dropna ()**: remove any row contains NA value.

2.3 Transform Categorical data into Numerical data

To avoid dealing with string values, we found that the name column needs to be transformed, so we Converted the Column Name into Orange = 0, Grapefruit = 1, by sub setting the name column from the dataset and then mapping its values to 0 and 1.

2.4 Calculate a set of descriptive statistics

Pandas data frame columns get stored as NumPy arrays and data frame operations are thin wrappers around NumPy operations. two methods used to calculate these stats: (1) Directly on the column data: which we didn't prefer because the piece of code to calculate it doesn't look good and takes a lot of space (2) Instead, we used Dictionary of (keys = column names & values = a list), we used this **dict** variable to store all the needed stats of the data and access it way more easier than the 1st method.

So by setting each alone and performing these functions (**np.mean (), np.median (), np.var (), np.std ()**) simultaneously on each column, we easily obtain our stats on each given feature.

2.5 Standardization using Z-score method

Using the precalculated stats of each column, we applied the Z-score transformation on each column simultaneously. the variable **columns_to_standardize** is a list containing the names of the columns that need to be standardized. The code then iterates over each column in columns_to_standardize using a for loop. Used formula is

$$Z = \frac{(X - \mu)}{\sigma}$$

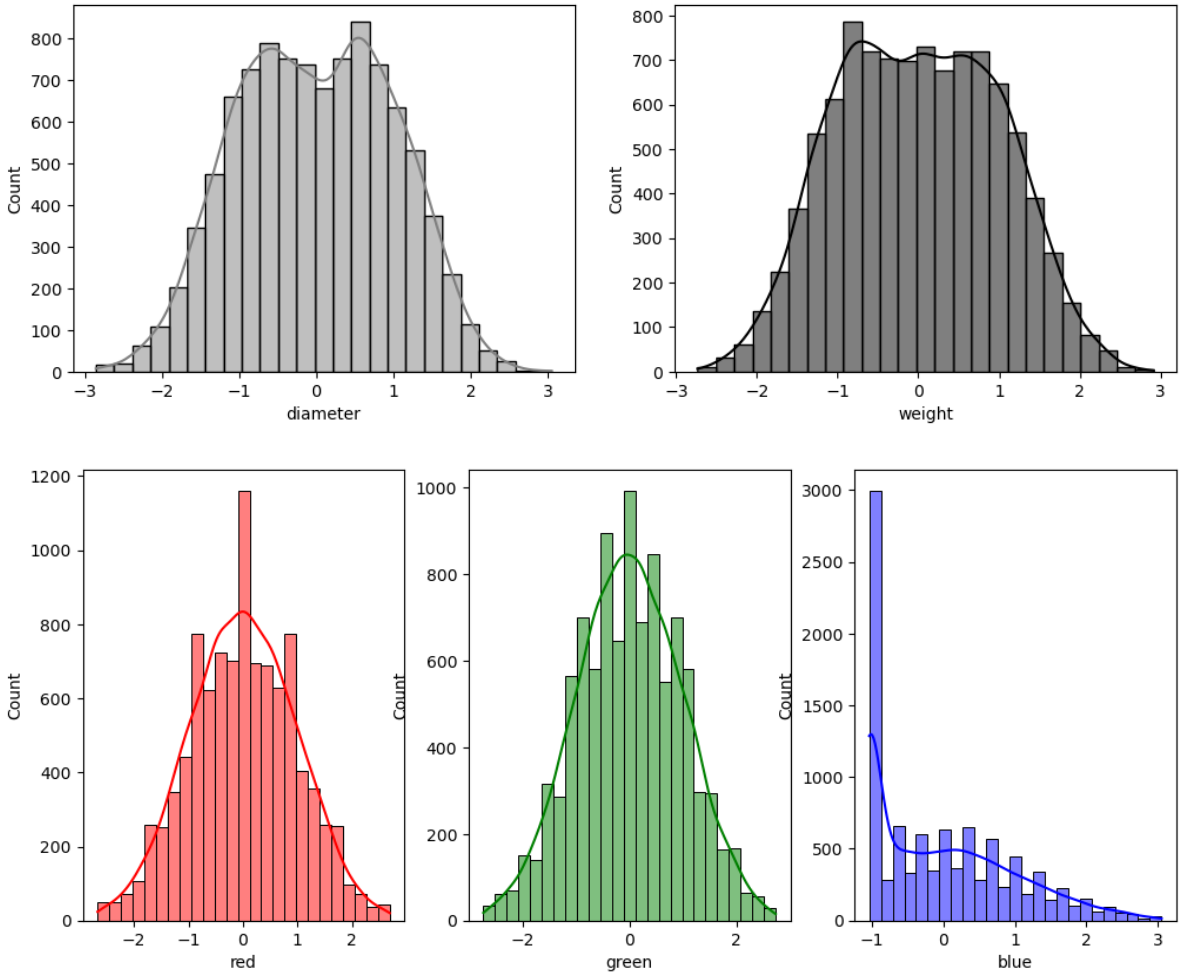


Fig.1: Distribution of data after Standardization

3 Hypothesis testing for Normality

3.1 Shapiro-Wilk test

Using Shapiro-Wilk test, it superimposes a normal curve over the observed distribution. It then computes which percentage of our sample overlaps with it: a similarity percentage. The null hypothesis for the Shapiro-Wilk test is a variable is normally distributed in some population, reject the null hypothesis if $p < 0.05$. The alternative hypothesis for the Shapiro-Wilk test is a variable is not normally distributed.

The main Equation for the Test Statistic is

$$W = \frac{(\sum_{i=1}^{n} a_i \, x_i)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

The variable (a) is the Shapiro-Wilk coefficients, we can get that from Shapiro-Wilk tables if n is less than 50, but for our data n is greater than 50, so we have to calculate (a) using the following steps:

1. Firstly, calculate the quantiles for the Shapiro-Wilk test statistic using this equation: $m_i = \sum_{i=1}^{n+1} \frac{i-0.375}{n+0.25}$
2. Calculate the sum of squared quantiles: $m = \sum_{i=1}^{n} {m^2}_i$
3. Calculate the scaling factor (u): $u = \frac{1}{\sqrt{n}}$
4. Calculate coefficient (an) and (an-1) which are (last 2 elements) using these equations:

$$a_{n-1} = -3.582633u^5 + 5.682633u^4 - 1.752461u^3 - 0.293762u^2 + 0.042981u + m_{n-1}m^{-0.5}$$

$$a_n = -2.706056u^5 + 4.434685u^4 - 2.071190u^3 - 0.147981u^2 + 0.221157u + m_n m^{-0.5}$$

5. Calculate (ε) using this equation: $\varepsilon = \frac{m - 2\,m_n^2 - 2{m^2}_{n-1}}{1 - 2\,a_n^2 - 2{a^2}_{n-1}}$
6. Calculate (a_i) using these equations: $a_1 = -a_n \;,\; a_2 = -a_{n-1} \;,\; a_i = \frac{m_i}{\sqrt{\varepsilon}} \; for\; 2 < i < n-2$

We calculated (a_i) so we can use the original Test Statistic Equation to calculate the p-value and to know if our features are normally distributed or not.

3.2 QQ plots

Q-Q plots are graphical tools used to assess the similarity between a dataset and a specific theoretical distribution, such as the normal distribution. They compare the quantiles of the observed data to the quantiles of the theoretical distribution, providing a visual indication of any deviations or departures from the expected distribution shape, as shown in the following figures:

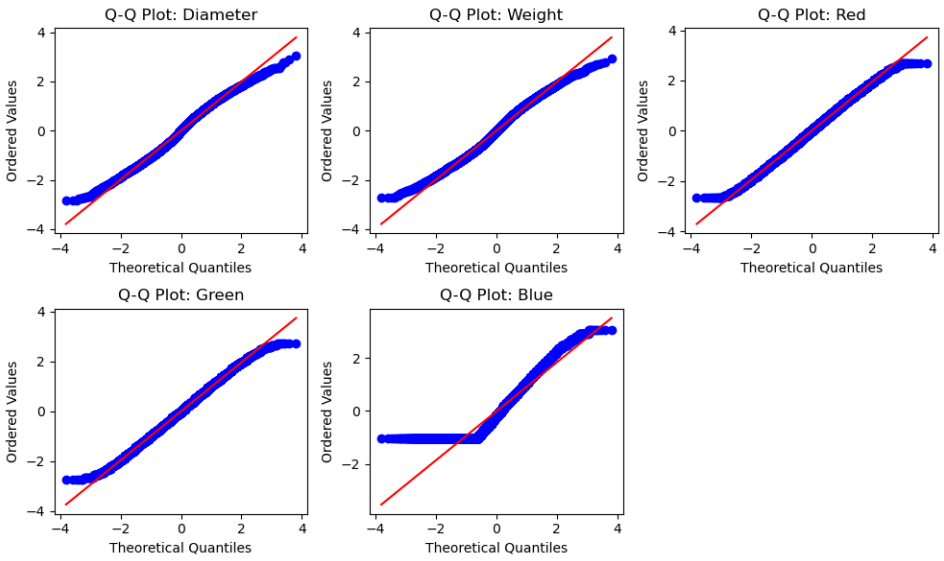


Fig.2: Q-Q plot to compare between distribution of data and normal distribution

4 Conditional distribution

A conditional probability plot displays the relationship between variables while conditioning on a third variable. It shows how the probability of an event or outcome varies based on different values or categories of the conditioning variable. This plot helps visualize the impact of the conditioning variable on the probability of the event occurring, providing insights into patterns or trends within the data.

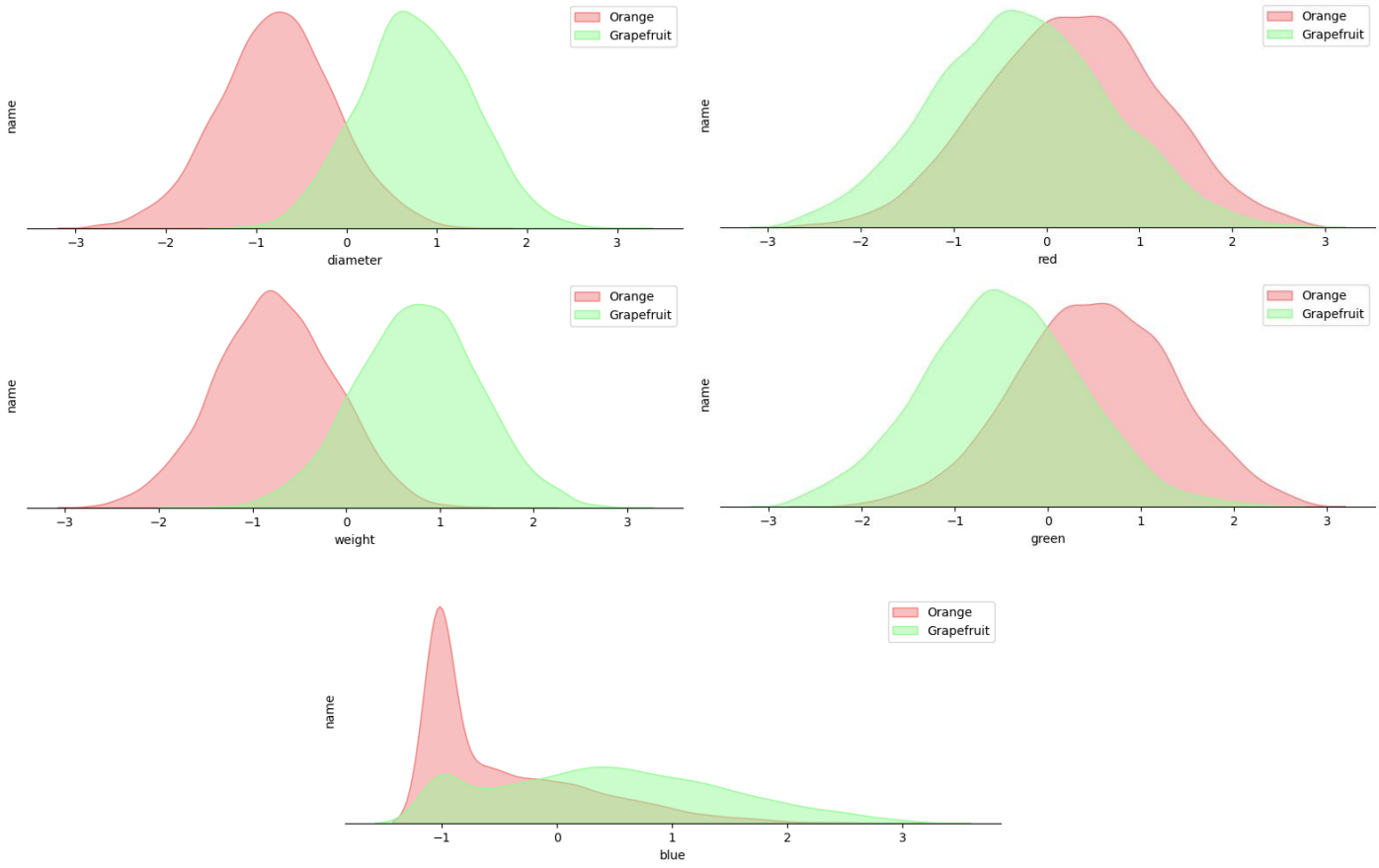


Fig.3: Conditional distribution between 2 classes and each feature

5 Naïve Bayes classifier

5.1 Theory and formula

The Naïve Bayes (NB) classifier is a popular and widely used machine learning algorithm for classification tasks. It is based on Bayes' theorem, which is a fundamental concept in probability theory. The NB classifier assumes that the features are conditionally independent given the class, hence the term "naïve." Despite this simplifying assumption, NB classifiers often perform well and are computationally efficient.

Formally, let's denote the target class as C and a set of features as $x = \{x_1, x_2, \ldots x_n\}$. The NB classifier calculates the probability of each class given the features and selects the class with the highest probability as the predicted class. Mathematically, this can be expressed as:

$$P(C|x) = \frac{P(x|C) * P(C)}{P(x)}$$

Where:

- P(C | x) represents the probability of class C given the features x.
- P(C) is the prior probability of class C.
- P(x | C) is the likelihood of observing the features x given class C.
- P(x) is the evidence probability, which is the probability of observing the features x.

To simplify the computation, the NB classifier makes the assumption of conditional independence among the features given the class. This assumption allows us to factorize the likelihood as:

$$P(\mathrm{x}\,|C) = \mathrm{P}(\mathrm{x}_1 \mid \mathrm{C}) * \mathrm{P}(\mathrm{x}_2 \mid \mathrm{C}) * \ldots * \mathrm{P}(\mathrm{x}_\mathrm{n} \mid \mathrm{C})$$

Based on the training data, the NB classifier estimates the prior probabilities P(C) and the likelihoods $P(x_i \mid C)$ for each feature. These estimates can be obtained using maximum likelihood estimation or other smoothing techniques.

During the classification phase, the NB classifier calculates the posterior probabilities for each class and selects the class with the highest probability as the predicted class:

$$P(\mathrm{x}\,|C)\,\mathrm{P}(\mathrm{C}) = \mathrm{P}(\mathrm{x}_1 \mid \mathrm{C})\ \mathrm{P}(\mathrm{x}_2 \mid \mathrm{C})\ \ldots\ldots \mathrm{P}\,(\mathrm{x}_\mathrm{n} \mid \mathrm{C}) * \mathrm{P}(\mathrm{C})$$

Selecting class with highest probability:

$$argmax_C\ P(\mathrm{x}\,|C) = C = argmax_C\ \mathrm{P}(\mathrm{x}_1 \mid \mathrm{C})\ \mathrm{P}(\mathrm{x}_2 \mid \mathrm{C})\ \ldots\ldots \mathrm{P}\,(\mathrm{x}_\mathrm{n} \mid \mathrm{C}) * \mathrm{P}(\mathrm{C})$$

Applying the log function to the equation to simplify computation

$$C = argmax_C\ \log \mathrm{P}(\mathrm{x}_1 \mid \mathrm{C}) + \log \mathrm{P}(\mathrm{x}_2 \mid \mathrm{C}) + \ldots\ldots + \log \mathrm{P}\,(\mathrm{x}_\mathrm{n} \mid \mathrm{C}) + \log \mathrm{P}(\mathrm{C})$$

Overall, the NB classifier provides a powerful tool for binary classification tasks, leveraging Bayes' theorem and the assumption of conditional independence to make predictions efficiently and effectively.

5.2 Software packages

NumPy (imported as "**np"**): used for various mathematical operations and array manipulations in this code. It provides efficient and convenient ways to work with multi-dimensional arrays and mathematical functions.

The code utilizes the following NumPy functions and attributes

Method	Discerption
np.unique()	Returns the unique elements of an array.
np.zeros()	Creates an array filled with zeros.
np.float64	Specifies the data type of the created arrays.
np.log()	Computes the natural logarithm of an array.
np.sum()	Computes the sum of array elements.
np.exp()	Computes the exponential of an array.
np.argmax()	Returns the indices of the maximum values along an axis.

Scikit-learn (imported as "**sklearn"**): a widely used machine learning library in Python that provides various tools for classification, regression, clustering, and more.

The code utilizes the following scikit-learn functions and classes:

Method	Discerption
GaussianNB ()	Creates an instance of the Gaussian Naive Bayes classifier.
Fit ()	Fits the classifier using the training data X_train and corresponding labels y_train.
Predict ()	Predicts the labels for the test data X_test.
accuracy_score ()	Computes the accuracy of the classifier by comparing the predicted labels with the true labels y_test.
roc_curve ()	Computes the Receiver Operating Characteristic (ROC) curve using the true labels y_test and the predicted labels. It returns the false positive rate (fpr) and true positive rate (tpr).
Auc()	Computes the Area Under the Curve (AUC) value using the false positive rate (fpr) and true positive rate (tpr).

5.3 Model Training and Accuracy

To train model, an instance of the **NaiveBayes** class is created. The fit method is called with the training data (**X_train** and **y_train**) to train the Naive Bayes classifier. After training, the classifier is used to predict the labels for the test data (**X_test**) using the predict method. The predicted labels are stored in the **y_pred** variable.

To evaluate the accuracy of the Naive Bayes classification, the accuracy function is defined. This function takes the true labels (**y_true**) and predicted labels (**y_pred**) as inputs. It calculates the accuracy by comparing the number of correct predictions with the total number of samples, and then converts it to a percentage. Finally, the accuracy of the Naive Bayes classification is printed using the accuracy function with the test labels (**y_test**) and predicted labels (**y_pred**).

The accuracy of the model after training on data is **92.4012%**

5.4 Built in Naïve Bayes classifier

The accuracy of the Naive Bayes (NB) classifier is compared between the self-implemented model and the **GaussianNB** model from the sklearn package. Both models yield an accuracy of approximately **92.4012%.** This finding demonstrates that the self-implemented NB classifier performs on par with the NB classifier provided by the sklearn package, highlighting the reliability and effectiveness of the self-implemented model.

5.5 ROC curve and AUC value

The ROC curve is a graphical representation of the classifier's performance by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds as shown in fig.4.

After plotting the ROC curve, the AUC value is calculated using the "**auc"** function from the sklearn package. It takes the FPR and TPR values as inputs and calculates the area under the ROC curve, AUC value is equal **0.923948**

A higher AUC value indicates better classifier performance, as it represents a larger area under the ROC curve and better discrimination between positive and negative sample

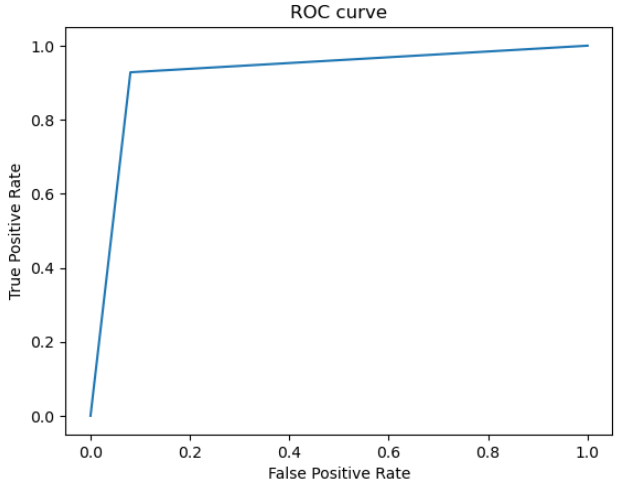
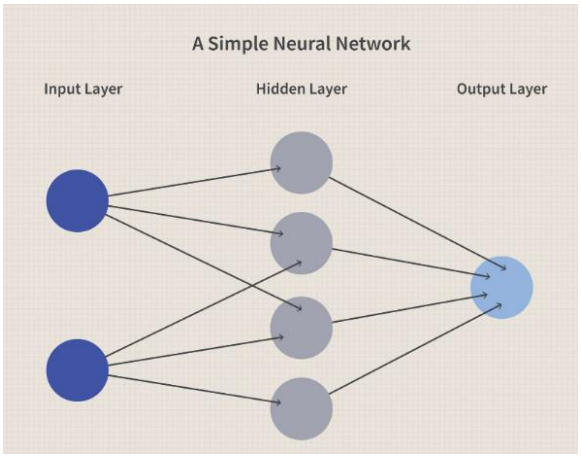


Fig.4: ROC curve of True Positive Rate (TPR) and False Positive Rate (FPR)

6 Simple Neural Network-based (NN) classifier

6.1 introduction into Neural Network

Neural networks, inspired by the structure and functioning of the human brain, consist of interconnected layers of artificial neurons, also known as nodes. These networks learn to extract meaningful patterns and relationships from the input data through a process called training.



During training, a neural network adjusts the weights and biases associated with each connection between neurons to minimize the difference between its predictions and the true labels of the training examples. This process, known as backpropagation, uses gradient descent optimization to iteratively update the network's parameters.

In binary classification tasks, a neural network typically consists of an input layer, one or more hidden layers, and an output layer. The input layer accepts the features as input, and the output layer produces the predicted class probabilities or labels.

By training a neural network on a labeled dataset, it can learn to accurately classify new, unseen examples based on the learned patterns and relationships. The performance of a neural network can be evaluated using metrics such as accuracy

6.2 Data Transformation

Converting data to tensors is a common step in deep learning research and allows for efficient computations on GPUs. The first part of the code converts the training and validation data, represented by **X_train** and **X_valid**, into tensors using the "**tc.tensor()**" function. This suggests that the research is based on a framework utilizes tensors, which is PyTorch.

Label encoding is a technique used to convert categorical labels (**y_train** and **y_valid**) into numerical values. The **LabelEncoder** assigns a unique integer to each distinct label in the dataset. This step is necessary when working with models that require numerical inputs.

One-hot encoding is a process where categorical values are represented as binary vectors, with each vector element corresponding to a specific category. This ensures that each sample can belong to one and only one class.

6.3 Initializing weight and bias tensors

The **tc.manual_seed()** method ensures consistent data generation by setting a seed for the random number generator. This guarantees reproducibility in subsequent experiments.

The weight tensor (w) is initialized using **tc.randn()** with a size of (2, 5), indicating 2 rows and 5 columns. By setting **requires_grad = True**, the tensor's gradients will be tracked during backpropagation for training purposes. The **retain_grad()** method is used to retain the gradients during computation. Similarly, the bias tensor (b) is having the same operations.

6.4 Forward propagation and Activation Function

Forward propagation is a key step in the computational process of neural networks, where input data is processed through the network's layers to produce an output or prediction.

Input Layer: The input layer receives the input data. Each input neuron represents a feature or attribute of the data.

Hidden Layer: The hidden layer consists of neurons that perform computations on the input data. Each neuron in the hidden layer receives inputs from the input layer, applies weights to the inputs, and computes a weighted sum. The weighted sum is then passed through an activation function.

Activation Function: The activation function introduces non-linearity into the neural network, allowing it to model complex relationships in the data. We used the **softmax** function as it is typically used for classification problems. It transforms the output of a neural network into a probability distribution over the classes, where the sum of the probabilities is 1. The output of the **softmax** function is always between 0 and 1. . The output of the activation function becomes the activation value of the neuron and is passed to output.

Output Layer: The output layer receives the activation values from the hidden layer neurons. For more justification, here is the weighted sum of our NN:

Calculate weighted sum:

$$a_0 = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_5 \cdot x_5 + b_1$$

$$a_1 = w_6 \cdot x_1 + w_7 \cdot x_2 + \cdots + w_{10} \cdot x_5 + b_2$$

Where: a_0 and a_1 are the two classes we classify between them, Weights are a (2, 5) matrix and biases are a (1, 2) matrix

6.5 Prediction error "loss or cost"

Prediction error measures the discrepancy between the predicted output of a machine learning model and the actual or target output. It quantifies how well the model is performing in terms of its ability to approximate the desired output.

During training, the goal is to minimize the prediction error by adjusting the model's parameters, such as weights and biases, using optimization techniques like gradient descent. The model iteratively updates its parameters to reduce the prediction error and improve its predictive performance.

We used Binary Cross-Entropy function as a loss function as it is commonly used for binary classification problems. It measures the dissimilarity between the predicted probability and the true binary label.

6.6 Backpropagation and Optimization loop

Backpropagation is a process used to compute the gradients of the model's parameters (weights and biases) with respect to their loss. It involves propagating the error backward through the network, layer by layer, and updating the parameters based on the calculated gradients.

We used Gradient descent as an optimization algorithm which is commonly used to train machine learning models, including neural networks. It is used to minimize a given loss function by iteratively updating the model's parameters in the direction of steepest descent of the loss function.

The optimization loop continues until the model's performance reaches a satisfactory level or the specified stopping criteria are satisfied, such as a maximum number of epochs.

6.7 Convergences and model accuracy

During the training process, the loss value is calculated for each iteration or epoch. These loss values is plotted over time to visualize the progress of the optimization process. The x-axis represents the number of iterations or epochs, and the y-axis represents the loss value. The plot shows how the loss decreases over time, indicating whether the model is converging or if further training is required, as shown in fig.5

Upon completing the training process with the previous neural network architecture, we achieved an accuracy of around 92.94%. This accuracy level is comparable to the accuracy achieved by the model constructed using the Naïve Bayes classifier.

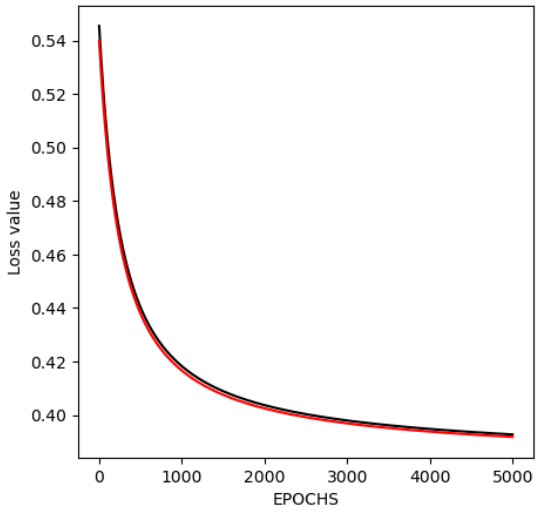


Fig.5: plot of loss curve until convergence

Conclusion

In this paper, we conducted a comprehensive analysis of binary classification using the Naive Bayes (NB) classifier and a simple Neural Network (NN) classifier. We explored various aspects of the classification process, including data exploration and cleaning, model training, evaluation, and comparison.

The Naive Bayes classifier demonstrated its effectiveness in handling categorical and quantitative features, utilizing the principles of probability and conditional independence assumptions. We implemented the NB classifier from scratch, calculated the descriptive statistics and standardized the features. Through histogram plotting and statistical tests, we assessed the distribution characteristics of the features and examined their conditional distributions on each target class.

The implementation of the NB classifier achieved comparable accuracy to the **GaussianNB** model from the sklearn package, highlighting the accuracy and reliability of our self-implemented model. This demonstrated that the NB algorithm can be successfully implemented from scratch using Python.

By analyzing the Receiver Operating Characteristic (ROC) curve and calculating the Area under the Curve (AUC), we obtained insights into the performance and discrimination capability of the classifiers. The comparison of the AUC values further highlighted the efficacy of the Naive Bayes classifier.

Additionally, we explored the utilization of a simple Neural Network classifier for binary classification. The NN classifier, trained using gradient descent or stochastic gradient descent, offered a flexible approach to learn complex non-linear relationships from the data. We plotted the loss function to monitor the convergence of the model during training and evaluated its accuracy on the test data.

In conclusion, this paper showcased the utilization of the Naive Bayes classifier and the Neural Network classifier for binary classification. The NB classifier demonstrated its simplicity, efficiency, and competitive performance, while the Neural Network classifier illustrated its ability to capture complex patterns and achieve accurate predictions. This research contributes to the understanding and application of these classification techniques in solving real-world problems. Further exploration and experimentation with various datasets and advanced techniques can enhance their performance and broaden their applicability in diverse domains.

Contribution

Contributor	Sec.	BN.	Distribution
Abdallah Mohamed Sayed	1	38	Data exploration, Transform Categorical data into Numerical data, Descriptive statistics, Standardization of data.
Mahmoud Magdy	2	21	Boxplot of features, Removing Outliers using interquartile Range, Data visualization, Implement NB classifier from sklearn package.
Mohamed El-Sayed Ali	2	12	Hypothesis testing for Normality, Q-Q plots, Conditional Distribution.
Mohamed El-Sayed Eid	2	13	Implementation of the Naïve Bayes classifier from scratch, Training and Test model, Implementation of Neural Network-based (NN) classifier, Visuals of conditional distribution

References

[1] "Python for Data Analysis" by Wes McKinney - Second Edition

[2] "Exploratory Data Analysis" by John W. Tukey - Reprint Edition

[3] "Practical Statistics for Data Scientists" by Peter Bruce, Andrew Bruce

[4] Scribbr: How to Find Outliers [https://www.scribbr.com/statistics/outliers/]

[5] Shapiro, S.S. & Wilk, M.B. (1965), an analysis of variance for normality (complete samples). Biometrika, Vol. 52.

[6] Royston, J. P. (1992) Approximating the Shapiro-Wilk W-test for non-normality, Statistics and Computing, vol. 2.

[7] "Pattern Classification" by Richard O. Duda, Peter E. Hart, and David G. Stork

[8] "Deep Learning with Python" by François Choll