

Comparative Analysis of Machine Learning Algorithms

Analysis and Evaluation for Classification Task

Monday, July 10th, 2023

Name: Ayush Anand Bourai 2020B5A21659P

Arin Agrawal 2020A1PS0277P



Table of Contents

(i) Acknowledgment	03
1. Abstract	04
2. Introduction	05
2.1 Data Used	05
2.2 Accuracy Or Generality?	05
3. Methodology	06
3.1 Libraries	06
3.2 Process From Scratch	06
3.2.1 Preparing Data and Dealing with Missing Values	07
3.2.2 Helper Functions	08
4. Decision Trees	09
4.1 Interpretability	11
5. Random Forest Algorithm	14
6. Naive Bayes Classifier	16
7. Logistic Regression	18
8. Neural Networks	20
8.1 Neural Networks with "ReLu()"	21
8.2 Neural Networks with "Tanh()"	22
9. Results	23
10. Conclusion	26
11. References	27

Acknowledgement

We would like to express our sincere gratitude and appreciation to all those who have contributed to the completion of this project on machine learning with decision trees, random forest, naive bayes, neural networks and logistic regression. This endeavor would not have been possible without the support, guidance, and collaboration of various individuals and resources.

First and foremost, we extend our deepest appreciation to Prof. N.L. Bhanu Murthy, for their invaluable guidance and expertise. Their knowledge and insights have been instrumental in shaping our understanding of these algorithms and machine learning concepts.

We would also like to acknowledge the open-source community for their invaluable contributions in developing and maintaining libraries and frameworks that have significantly facilitated the implementation and experimentation process. Specifically, we would like to acknowledge the contributions of the developers behind pandas, NumPy etc., for providing a powerful and user-friendly platform for implementing these algorithms.

Lastly, we would like to express our gratitude to our families, friends, and peers for their unwavering support and understanding throughout the project. Their encouragement and motivation have been instrumental in keeping our spirits high during challenging times.

In conclusion, this project has been a collaborative effort that involved the contributions and support of numerous individuals and resources. We are truly grateful to everyone who has played a part, no matter how big or small, in making this project a reality.

Thank you.

Ayush Anand Bourai 2020B5A21659P

Arin Agrawal 2020A1PS0277P

1. Abstract

This report presents a comprehensive analysis and evaluation of decision trees, random forest, naive bayes, neural networks and logistic regression for a classification task.

Decision trees are widely used in machine learning and data mining due to their simplicity, interpretability, and ability to handle both categorical and numerical features.

Logistic regression is a statistical machine learning algorithm used for binary classification tasks, which predicts the probability of an instance belonging to a particular class based on its features, using a logistic function to map the input to a range between 0 and 1.

Neural networks are interconnected layers of artificial neurons that learn from data to recognize patterns and make predictions, inspired by the functioning of biological neurons in the human brain.

Naive Bayes is a probabilistic machine learning algorithm that assumes independence between features and uses Bayes' theorem to predict the probability of an instance belonging to a particular class based on its feature values.

Random Forest is an ensemble machine learning algorithm that combines multiple decision trees to make predictions, where each tree is trained on a random subset of the data and features, and the final prediction is determined by aggregating the predictions of all the trees.

The objective of this study is to assess the classification accuracy and how each algorithm differs from the other, and evaluate their performance on the census-income-dataset containing census information for 48,842 people. It has 14 attributes for each person (age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, and native-country) and a Boolean attribute class classifying the input of the person as belonging to one of two categories >50K, <=50K.

The findings demonstrate that decision trees exhibit competitive performance in terms of accuracy, especially when combined with ensemble methods like Random Forests. Moreover, decision trees offer interpretable models that allow domain experts to understand the decision-making process. However, limitations related to overfitting and handling continuous variables are also discussed for each algorithm.

This report contributes to a comprehensive understanding of these algorithms, their strengths, and their limitations. The results of this study can guide researchers and practitioners in leveraging a decision for improved classification performance and interpretable models in various domains.

2. Introduction

The main objective of the report is to highlight the application of multiple models on the provided dataset. To construct an optimal sized decision tree for the prediction of whether the income of a given person is $>50K$ or $\leq 50K$, using the census-income dataset from the US Census Bureau. The optimal sized decision tree is to be obtained by applying the “Reduced Error Pruning” technique. To construct a Random Forest Classifier and compare its result with the decision tree and models like Logistic Regression, Neural Networks and Naive Bayes Classifier. Also, to discuss the advantages and limitations of these approaches.

2.1. Data Used

The census-income dataset contains census information for 48,842 people. It has 14 attributes for each person (age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, and native-country) and a Boolean attribute class classifying the input of the person as belonging to one of two categories $>50K$, $\leq 50K$. The prediction problem here is to classify whether a person's salary is $>50K$ or $\leq 50K$ given the attribute values.

Shared in the form of 'word' documents it was converted to '.csv' files for easy implementation using some well known libraries in 'python3'.

2.2. Accuracy Or Generality?

The different models were implemented on the dataset provided. Some models worked well on the dataset. But there was no significant model amongst these which would have outperformed the rest. Amongst all the decision tree seems to be the best, not only because of its high accuracy on this dataset but also due to its interpretability. The neural networks didn't really put up a good result in spite of trying different numbers of hidden layers, with different nodes. There wasn't any significant improvement in the process. The logistic regression on the other hand seems to have had some easy implementation and gave a good result. So was the Naive Bayes classification. But there is some interpretation on the decision tree that can be understood.

3. Methodology

To explain in brief the method used was to create helper functions and arrange them appropriately to get a decision tree with the help of a method `decision_tree_algorithm()` which returns the decision tree. The method used didn't use pre existing functions on the decision tree. Everything was made from scratch.

3.1. Libraries

Major libraries used are: Pandas, NumPy, Matplotlib, seaborn, random and pprint.

Pandas: A software library for data manipulation and analysis.

NumPy: For large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Matplotlib: A plotting library for the Python programming language and its numerical mathematics extension NumPy.

seaborn: A widely popular library used for data visualization.

random: A series of functions for generating or manipulating random integers

pprint: Known as "pretty-print" it provides capability to arbitrary Python data structures to be used as input by the interpreter.

3.2. Process From Scratch

Firstly, we prepared the data and took care of missing values. Secondly, we wrote the helper functions and finally we found the decision tree and pruned it to increase its optimality. Similarly, we also do this for the random forest, logistic regression, neural networks and the naive bayes classifier. So we have made the helper functions for all of them and called them one by one by making a model.

3.2.1. Preparing Data and Dealing with Missing Values

For preparing the data, first we converted the census data to a '.csv' file. Then we used the block coding wherein we execute blocks of code, rather than executing an entire file, like the Jupyter Notebook. After converting to '.csv' we renamed the attributes so that we can access them. Then we checked for missing values and suitably replaced them.

Mean, mode, and median are commonly used measures of central tendency that can be beneficial for replacing missing values in a dataset, depending on the nature of the data and the specific context. Here are some scenarios where each measure can be particularly useful:

Mean: The mean is beneficial when dealing with numerical data that is approximately normally distributed or lacks extreme outliers. It represents the average value of the data and can be used to fill in missing values. However, it is sensitive to extreme values, so if the data contains outliers, the mean may not accurately represent the typical value.

Mode: The mode is beneficial for categorical or discrete data, where the most frequent value is sought. It is particularly useful when dealing with missing values in nominal or ordinal variables. Replacing missing values with the mode can help preserve the distribution and frequency of existing categories.

Median: The median is beneficial when the data contains outliers or is skewed. It represents the middle value of a dataset when it is sorted in ascending or descending order. The median is less sensitive to extreme values compared to the mean, making it a suitable measure for replacing missing values in skewed distributions or when outliers are present.

It's worth noting that there is no one-size-fits-all approach, and the choice of using mean, mode, or median depends on the specific characteristics of the data and the research question. Additionally, other techniques, such as regression imputation or multiple imputation, may be more appropriate in certain cases to handle missing data more comprehensively. It is essential to carefully consider the properties of the data and the potential impact of using these measures before deciding on an imputation strategy.

First, we found that whether each attribute given is a "Continuous" or "Discrete"/"Categorical". So, if there are missing values, and they belong to a "continuous" column then we replace the missing values with the median, and if they belong to a "discrete" column then we replace the missing values with "mode" of that respective column. In this way we dealt with the missing values.

Then we also created a method to split our dataset into the required proportion, either according to percentage or number of samples.

A function was also made to check the purity of the data.

```
def determine_type_of_feature(df): #CHECKS cont. or discrete  
  
def missing_values(df):
```

```
def train_test_split(df,test_size):

def check_purity(data):
```

3.2.2. Helper Functions

Besides these, the other functions that helped us make the decision tree and used the concept of Shanon's entropy, $-\sum_{i=1}^n p(x_i) \cdot \log_2(p(x_i))$, to find the attribute with least entropy or the maximum information gain. That attribute was then further used as a node and classified our data.

```
def classify_data(data):

def get_potential_splits(data):

def split_data(data, split_column,split_value):

def calculate_entropy(data):

def calculate_overall_entropy(data_below, data_above):

def determine_best_splits(data, potential_splits):
```

Like this we also made helper functions in Logistic regression. We calculated the activations and the sigmoid and compared the predictions for them. We ran the model for different learning rates and picked the one with the best learning rate and minimum loss amongst all. We found that the learning rate of 1 was best for this dataset.

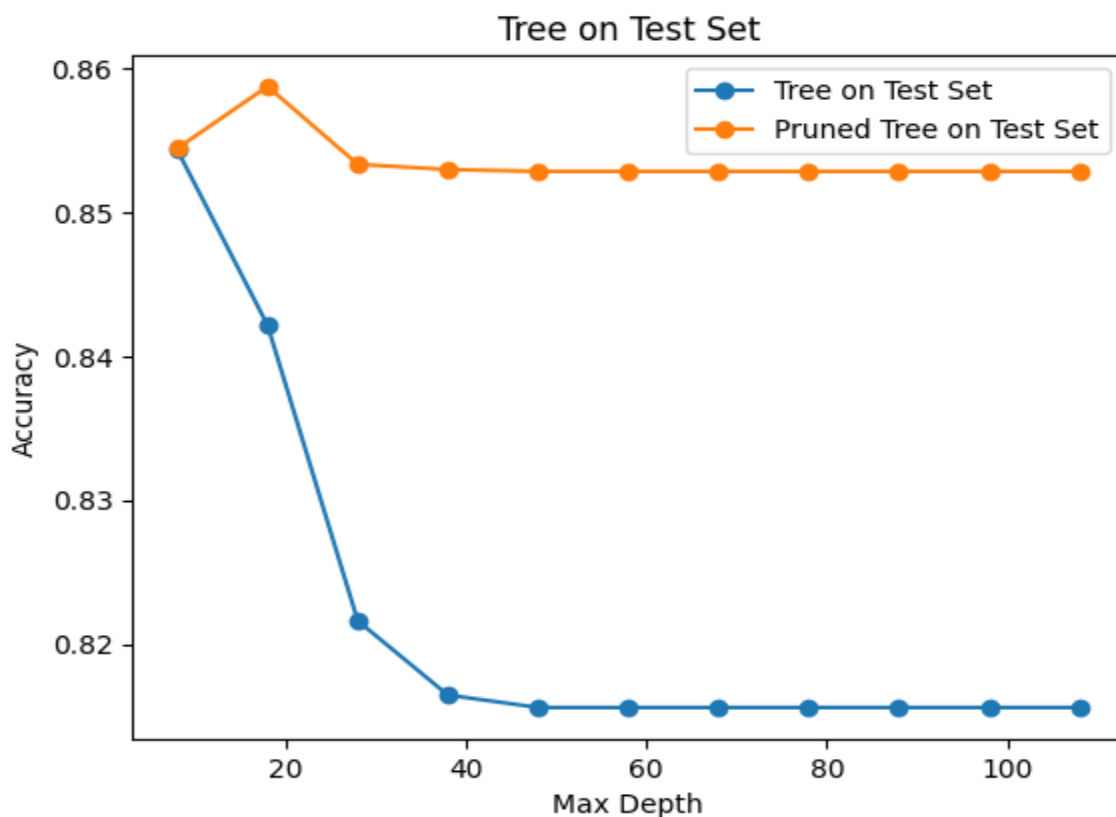
```
learning_rates = [5, 1, 0.9, 0.5 ,0.1 ,0.01, 0.001, 0.0001]
```

Similarly, for Neural Networks and Naive Bayes, a similar procedure was followed. Building functions for the sigmoid, tanh, ReLu, forward propagation, backward propagation, plotting the normal distribution for the set of values and so on.

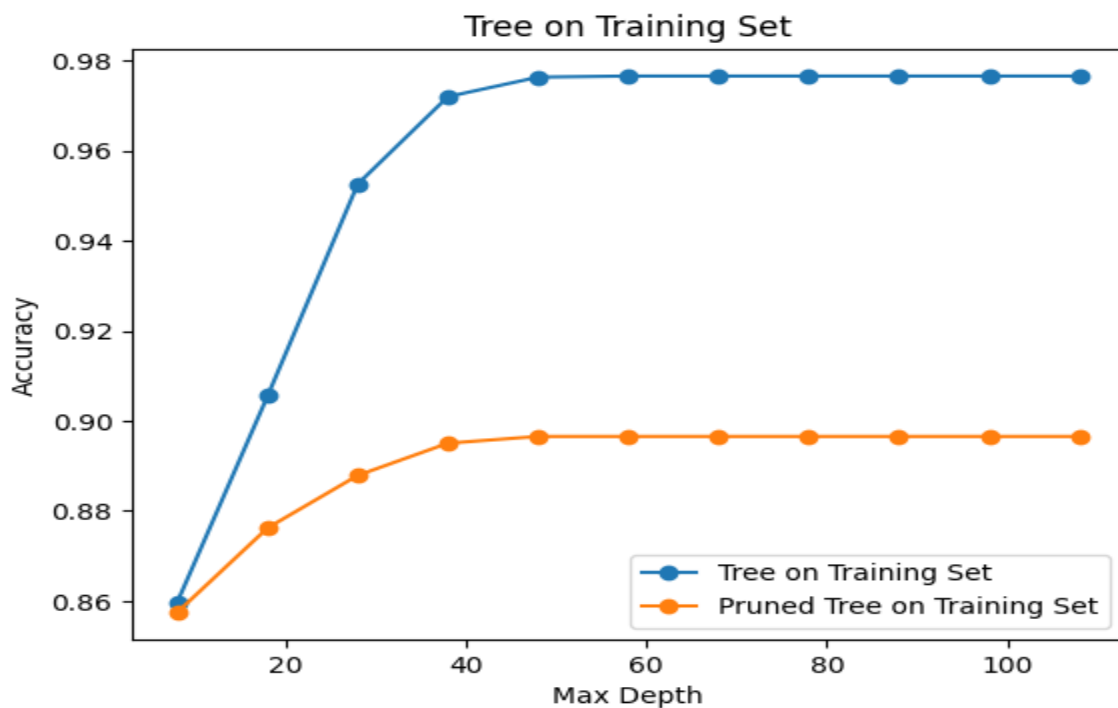
4. Decision Trees

Later on, after finding the pruned tree from the large trained tree, we could see the sharp drop in training accuracy, but there was a rise in the test set accuracy. So, the essence of the algorithm was to catch the generality of the data. Rather than overfitting the data, we need to classify the unseen data with more accuracy. Since, there was a 4% surge in the test data accuracy, it seems that our tree before Reduced Error Pruning was overfitting the training data and performing relatively poorly on the test data. But once the tree was pruned it performed better on the test data. Hence, it performed much better on the unseen data, which is the ultimate goal of all the machine learning algorithms. Since we regularized our tree by pruning it, it was nice to see that pruning the tree gave more correctly classified test samples.

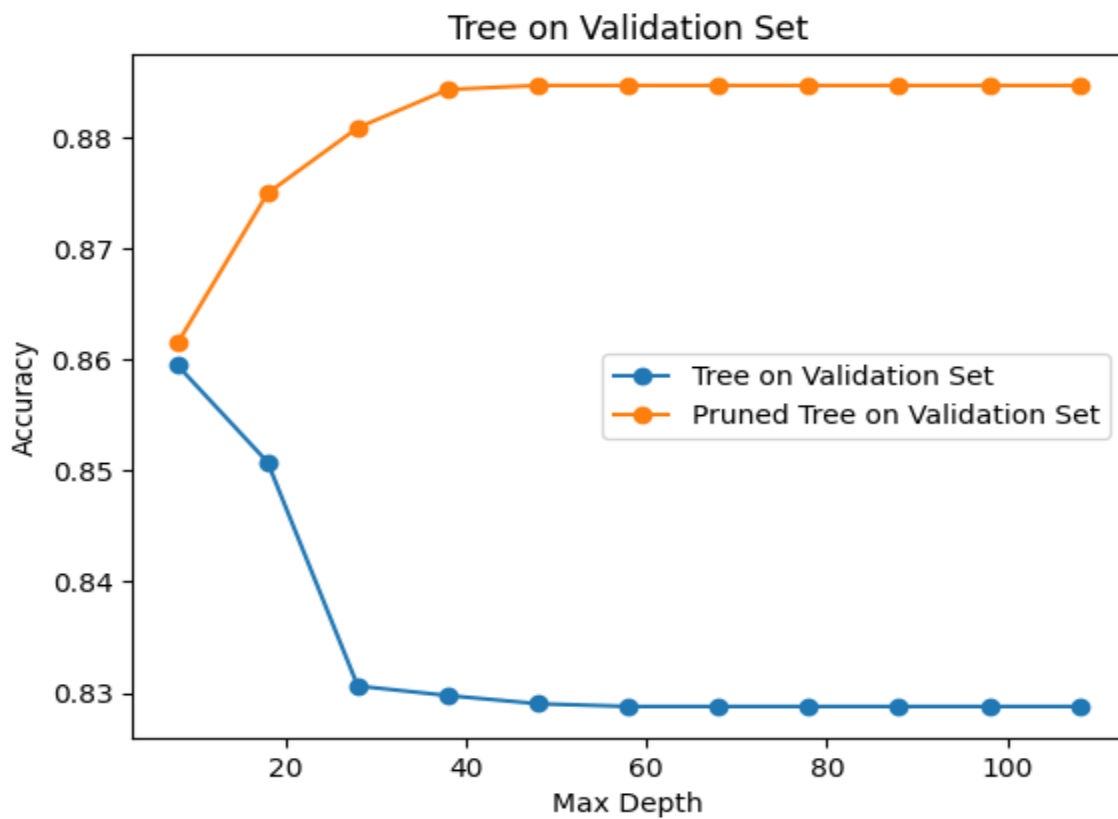
Thus, for the test set, training set and the validation set the following graphs were obtained for unpruned and pruned trees.



This graph shows how the algorithm performs better on pruning and showcases that there is optimum ~0.86. Hence pruning the tree has increased the optimality.



This tree shows how an unpruned tree starts overfitting the data after a point and pruning brings down overfitting, leading to a decrease in training set accuracy.



The validation set shows how the accuracy of the pruned tree has drastically improved leading to more optimal results.

4.1. Interpretability

Thus, we can see here that the decision tree helps us in getting an interpretation of the classified example. We would know that under what conditions the specific sample was classified as >50K or <=50K. The following images explain this. We can know the rules or the questions that can be asked at each node.

This is an example of one the training data. This lists out the attributes it has and these attributes are compared with the question or set of rules at each node to reach the required classification. In this case we can see that the example is right classified as "<=50K".

The tree gives a set of rules or questions that the sample answers at each node and eventually gets classified in either of the two classes. The example is first asked if 'marital_status = Married-civ-spous' and then 'education_num <= 12' and then 'capital_gain <= 5013' and so on. Here it is using '=' for the discrete or categorical attributes and '<=' for the continuous attributes. Here, in continuous data, to find these split points lead the tree to take a lot of time in training. Although the time was reduced a bit by using the NumPy library. But it could've been even shortened by reducing the number of places where we find splits, but that might lead to not finding the optimal splitting

point, which leads to the lowest Shannon's entropy, $-\sum_{i=1}^n p(x_i) \cdot \log_2(p(x_i))$ for the divided parts.

Hence this approach was used to discretize the continuous data, but the only drawback was that it was time consuming.

Next we used the helper functions in a set order over our dataset and got a decision tree. The following snippet represents the code for the function that returns a decision tree that grows and stops in accordance to the lowest entropy factor given by Shanon.

Thus, we were successfully able to train a decision tree according to the given data and classify our training set with an accuracy of 97.66%, 82.88% with the Validation Set and 81.56% with the Test set. But the size of this tree is huge. It contains 6,841 nodes and has a depth of 106. The image shows the result for the tree made from this function.

```
accuracy_train = calculate_accuracy(train_df,tree)
accuracy_val = calculate_accuracy(valid_df,tree)
accuracy_test = calculate_accuracy(test_df,tree)
print("Training Set Accuracy:",accuracy_train)
print("Validation Set Accuracy:",accuracy_val)
print("Test Set Accuracy:",accuracy_test)

[188] ✓ 2.6s Python
... Training Set Accuracy: 0.9765670587512668
Validation Set Accuracy: 0.8287679646235107
Test Set Accuracy: 0.8156019656019656
```

Then after using 3 sets of functions used to prune the tree. The functions were:

```
def make_predictions(df, tree):

def filter_df(df,question):

def pruning_result(tree, df_train, df_val):

def post_pruning(tree , df_train , df_val):
```

After using these functions we got the pruned tree with 1453 nodes and a depth of 84.

The following snippet returns the pruned tree using recursive techniques.

```
def post_pruning(tree , df_train , df_val):

    question = list(tree.keys())[0]

    yes_answer , no_answer = tree[question]

    #base case

    if not isinstance(yes_answer,dict) and not isinstance(no_answer,dict):

        return pruning_result(tree , df_train , df_val)

    else: #Recursive case

        df_train_yes, df_train_no = filter_df(df_train , question)

        df_val_yes, df_val_no = filter_df(df_val , question)

        if isinstance(yes_answer,dict):

            yes_answer = post_pruning(yes_answer,df_train_yes,df_val_yes)

        if isinstance(no_answer,dict):
```

```

no_answer = post_pruning(no_answer,df_train_no,df_val_no)

tree = {question : [yes_answer,no_answer]}

return pruning_result(tree , df_train , df_val)

```

After pruning the tree we got an accuracy of 89.65% on the training set, 88.47% on the Validation set and 85.28% on the Test set.

```

accuracy_train = calculate_accuracy(train_df,pruned_tree)
accuracy_val = calculate_accuracy(valid_df,pruned_tree)
accuracy_test = calculate_accuracy(test_df,pruned_tree)
print("Training Set Accuracy:",accuracy_train)
print("Validation Set Accuracy:",accuracy_val)
print("Test Set Accuracy:",accuracy_test)

```

[195] ✓ 1.8s Python

```

... Training Set Accuracy: 0.896532661773287
Validation Set Accuracy: 0.8846579044343447
Test Set Accuracy: 0.8528255528255528

```

Thus, this observation leads us to our next insight of Interpretability and extracting Generality from the data.

5. Random Forest Algorithm

Feature Engineering

As the dataset contains categorical features, such as "workclass" and "native_countr," it is necessary to transform them into numerical representations before feeding them into the random forest algorithm. Dummy encoding is used to convert the categorical data into numeric form. Pandas' `get_dummies()` function is applied to the "workclass" column, which has multiple unique values (Sategov, Private, and many more), resulting in new columns representing these values. The issue of the dummy trap, where one column is redundant, is addressed by dropping one of the encoded columns. The same process is applied to the "education" feature, which has many unique values (Bachelors, HSgrad, and many more). The resulting DataFrame consists of the transformed features, "workclass" and "education" and includes other features as well.

Concatenation of DataFrames

To incorporate the encoded features into the main DataFrame, the encoded DataFrames are concatenated using the `pd.concat()` function along the column axis. A new DataFrame called `new_data` and `new_data_t` is created for training and testing data resp., which includes all the relevant features for further analysis.

Removing Redundant Columns

The concatenated DataFrame may contain duplicated columns, such as "workclass" and "marital_status," which are repeated from the original DataFrame. These redundant columns are dropped using the `drop()` function with the appropriate column names. The resulting DataFrame, `new_data`, now contains all the necessary features without redundancy.

Prediction and Evaluation

The trained random forest classifier is used to make predictions on the test set (`x_test`). The predictions are stored in the variable `Y_pred`. To evaluate the performance of the random forest classifier, several metrics are employed. The confusion matrix, classification report, and accuracy score are computed using functions from the `sklearn.metrics` module. The confusion matrix provides an overview of the classifier's performance, indicating the number of correctly classified and misclassified instances. The classification report displays metrics such as precision, recall, and F1 score, providing insights into the classifier's accuracy for each class. Finally, the accuracy score represents the overall accuracy of the random forest classifier on the test set.

6. Naive Bayes Classifier

Naive- Bayes classification is a Probabilistic classifier which is based upon the Bayes theorem (with strong independence theorem) which states that:

$$P(c | x) = \frac{P(x | c)P(c)}{P(x)}$$

Likelihood
Class Prior Probability
Posterior Probability
Predictor Prior Probability

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

Here c is the hypothesis and x is the set training data. Our interest is to find the probability of a hypothesis given the training data, which is also referred to as Posterior Probability.

Our aim here is to maximize this probability to get the hypothesis that best fits the data we input as testing data.

As in the second expression we assume that training examples are independent of each other and the probability of training data P(x) is constant as it does not change with different hypotheses.

Now,

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

y is the hypothesis and x represents the training data set

Since the probabilities here would range from 0 to 1 we take logarithms on both sides to simplify the calculation and represent the expression as a sum.

$$y = \underset{y}{argmax} \log(P(x_1 | y)) + \log(P(x_2 | y)) + \dots + \log(P(x_n | y)) + \log(P(y))$$

P(y) - posterior probability (frequency of each hypothesis)

P(x_i/y) - class conditional probabilities which we model from the Gaussian distribution.

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

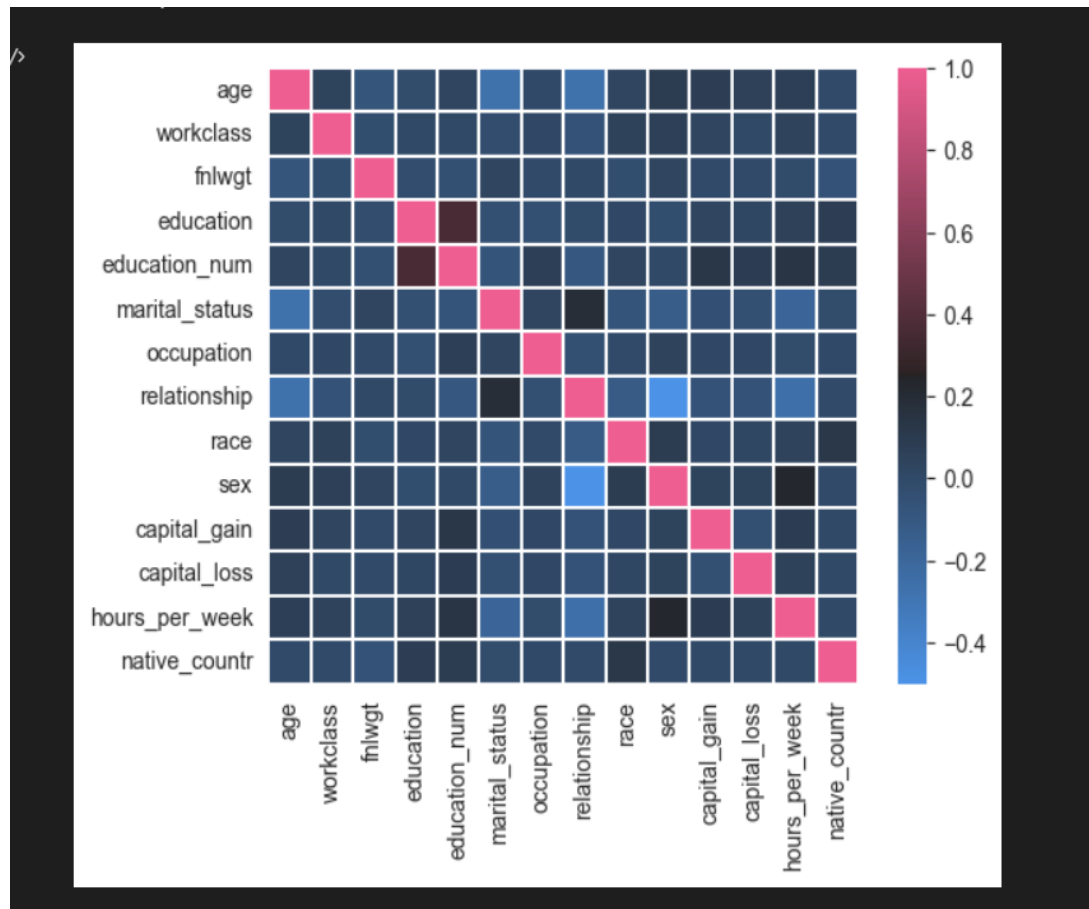
In training we calculate the mean, variance and prior (frequency) of each hypothesis with the training set.

In the prediction step we calculate the posterior probability with

$$y = \underset{y}{argmax} \log(P(x_1 | y)) + \log(P(x_2 | y)) + \dots + \log(P(x_n | y)) + \log(P(y))$$

and the Gaussian distribution formula.

And finally choosing the hypothesis with max posterior probability.



This heatmap plot for the correlation between the attributes showcases that the attributes are more or less, conditionally independent, hence there was no need to specially treat the attribute values.

7. Logistic Regression

Logistic regression is a statistical machine learning algorithm used for binary classification tasks, which predicts the probability of an instance belonging to a particular class based on its features, using a logistic function to map the input to a range between 0 and 1. So here we used the sigmoid function as an activation (could also use 'tanh()') and the 'ReLU' functions. Using the sigmoid function for the linear activations, we trained the model and got an accuracy of 81.1% on the dataset.

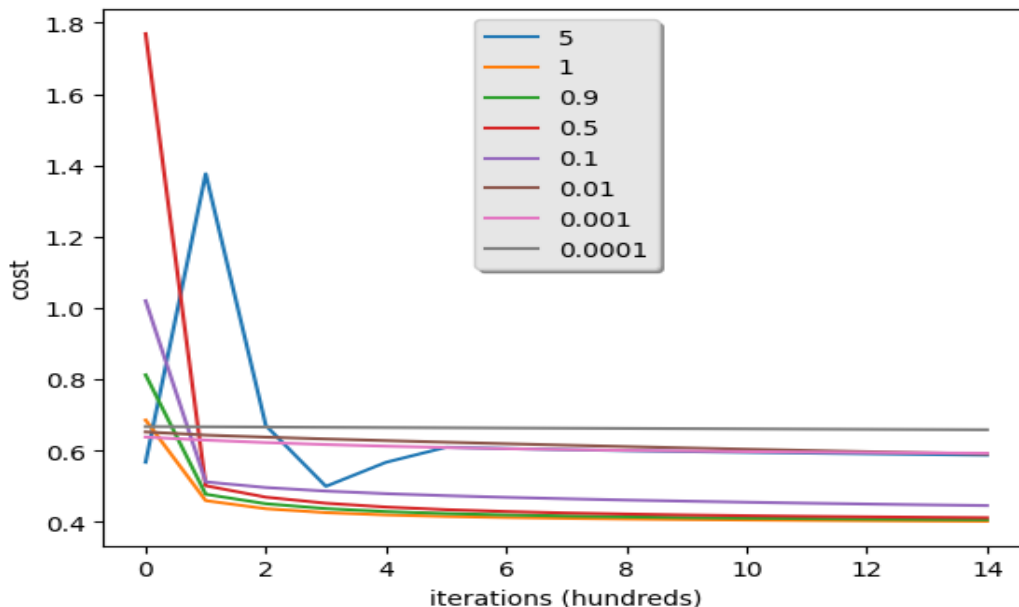
Using the gradient descent algorithm we approached the minima of the cost function and found the optimal minima which corresponds to the values of 'w' and 'b' learning parameters.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

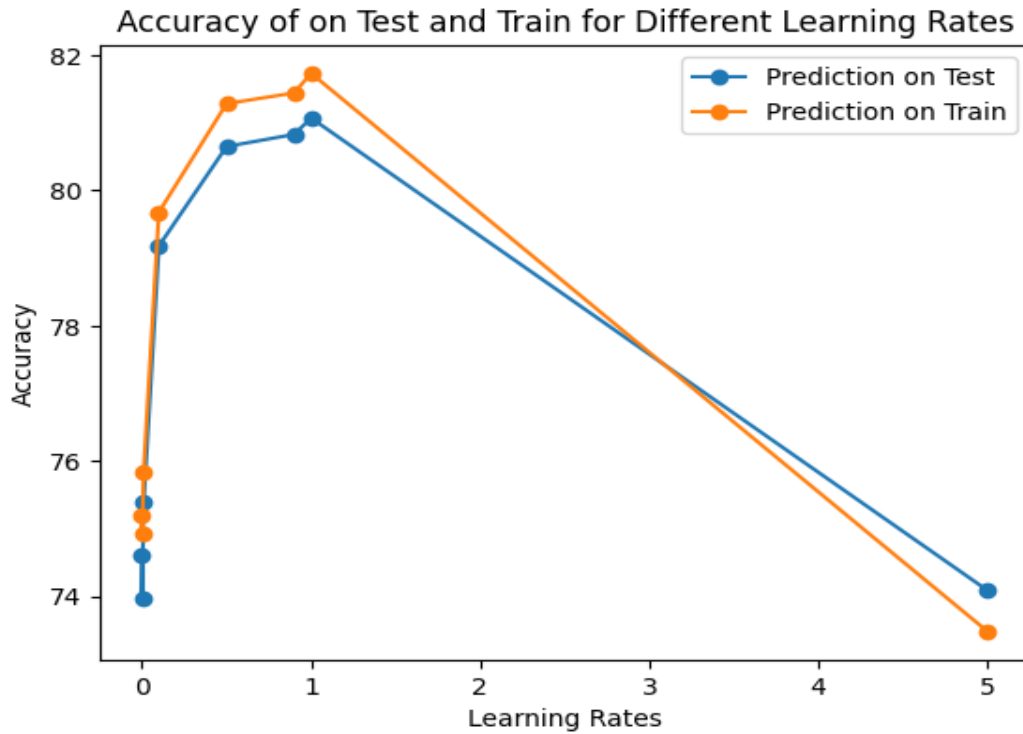
$$J(\theta) = \frac{1}{m} \left[\sum_{i=1}^m -y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right]$$

m = number of samples

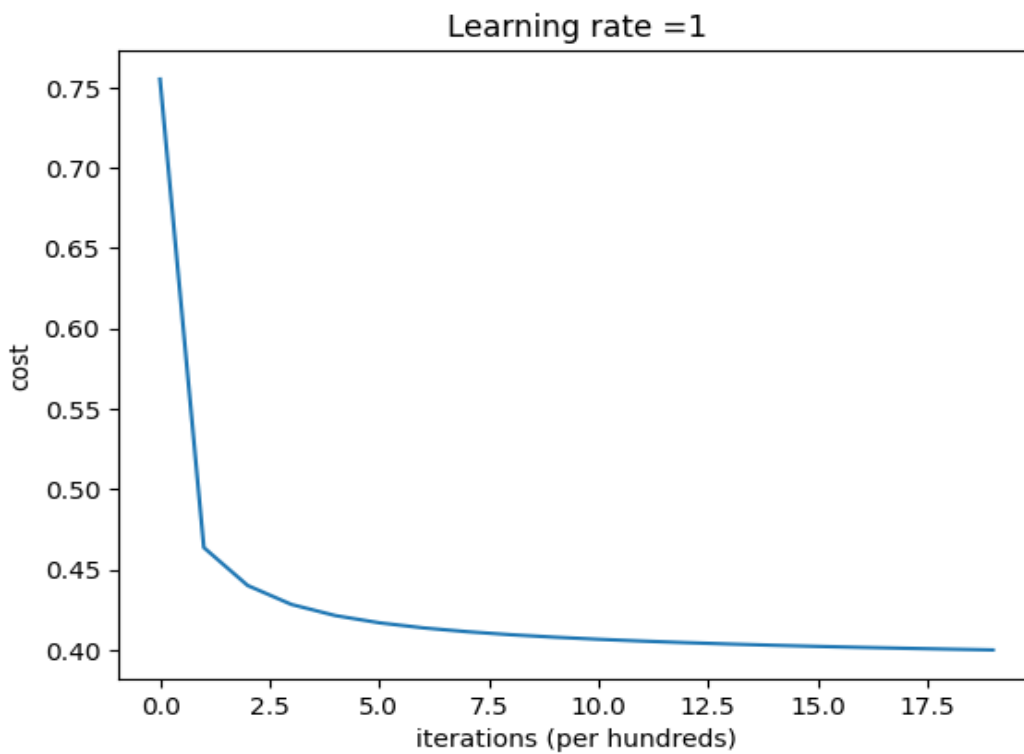
Thus, this loss function was implemented for the logistic regression model. Now to find the best learning parameters for the gradient descent algorithm we tried different learning rates and plotted them with the cost with each model and chose the best one.



This image shows that the learning rate of 0.9 - 1 worked the best for this model. We selected 1 for the final run and saw that it has the best accuracy on the test set too.



This plot shows that for the learning rate of 1 the logistic regression model worked the best on the test set and the training set.



Thus, finally we were able to get an accuracy of 81-82% on the dataset.

8. Neural Networks

Neural networks, also known as artificial neural networks or deep learning models, are a class of machine learning algorithms inspired by the structure and functioning of biological neurons in the human brain. They consist of interconnected nodes, called artificial neurons or "units," organized in layers. Each unit takes input, performs a computation, and produces an output. By stacking multiple layers of units, neural networks can learn complex patterns and relationships in data, enabling tasks such as classification, regression, image recognition, natural language processing, and more. Training a neural network involves adjusting the weights and biases of the units through a process called backpropagation, where the network learns from labeled training examples to improve its predictions.

Here we had to do the classification task for the same problem dataset. Here instead of using the sigmoid function for all the layers we did something different. We had made a program that would account for the number of layers in itself, we just had to pass the dimensions of the architecture, i.e. number of nodes in each layer.

```
layer_single_hidden_dims = [14,7,1]
layer_double_hidden_dims = [14,11,7,1]
layer_triple_hidden_dims = [14,10,6,4,1]
```

We decided to have the nodes in each layer with this pattern. When determining the number of nodes for each hidden layer in the neural network, there was no definitive answer or "best" number that universally applies to all scenarios. The optimal number of nodes depends on various factors, including the complexity of the problem, the amount of available training data, and the desired model capacity. A common approach is to start with a number of nodes between the number of input features and the number of output classes. In other words, we began by setting the number of nodes in each hidden layer to be roughly the average of the input and output dimensions of two consecutive layers.

Since, while using multiple layers, we found that using sigmoid for all hidden layers and even starting all parameters from zero were both hindering the better output. Since, if we would start all parameters 'w' and 'b' from zero then each node would be computing the same matured features. So it would not make sense to have different numbers of nodes, if they are all computing the same features. For using sigmoid for all layers, sometimes the values for multiple hidden layers got smaller, for the computer to keep track. So we used "relu" and "tanh" one time each for the entire process. Suppose, there are 'L' layers, then we applied a "relu" activation function for 'L-1' layers and used a 'sigmoid' activation function for the last layer, as we needed a classification between 0 and 1.

8.1. Neural Networks with “ReLU()”

For a single layered neural network we had 14 attributes. Thus, our nodes in the input layer are 14, in the hidden layer are 7, and in the output layer is 1 since it is a classification problem.

```
layers_dims = layer_single_hidden_dims
parameters_1, costs_1 = l_layer_model(x_train, y_train, layers_dims, learning_rate = 0.9, num_iterations = 10000, print_cost = False)
print(costs_1)
pred_train1 = predict(x_train, y_train, parameters_1)
pred_test1 = predict(x_test, y_test, parameters_1)
acc_train1 = np.sum((pred_train1 == y_train)/y_train.shape[1])
acc_test1 = np.sum((pred_test1 == y_test)/y_test.shape[1])
print("Training accuracy for 1 hidden layer", np.sum((pred_train1 == y_train)/y_train.shape[1]))
print("Test accuracy for 1 hidden layer", np.sum(np.sum((pred_test1 == y_test)/y_test.shape[1])))
```

[88] ✓ 1m 44.5s Python

... Cost after iteration 9999: 0.33203250960031877
[0.6932207658266462, 0.37430592672357174, 0.352936069431986, 0.3463620522961265, 0.34231057521273056, 0.33936936420868274, 0.337366050743036, 0.337366050743036, 0.337366050743036]
Training accuracy for 1 hidden layer 0.8424092409240926
Test accuracy for 1 hidden layer 0.8398064275964758

For the single hidden layer network, we found that when we used “relu” for ‘L-1’ layers and “sigmoid” for the last layer, we got an accuracy of ~84% on the test set.

```
layers_dims = layer_double_hidden_dims
parameters_2, costs_2 = L_layer_model(x_train, y_train, layers_dims, learning_rate = 0.9, num_iterations = 10000, print_cost = False)
print(costs_2)
pred_train2 = predict(x_train, y_train, parameters_2)
pred_test2 = predict(x_test, y_test, parameters_2)
acc_train2 = np.sum((pred_train2 == y_train)/y_train.shape[1])
acc_test2 = np.sum((pred_test2 == y_test)/y_test.shape[1])
print("Training accuracy for 2 hidden layer", np.sum((pred_train2 == y_train)/y_train.shape[1]))
print("Test accuracy for 2 hidden layer", np.sum((pred_test2 == y_test)/y_test.shape[1]))
```

Cost after iteration 9999: 0.3239145338345491
[0.6931468038655293, 0.4294579607668908, 0.3554385014881484, 0.3445696067311969, 0.33906381955693576, 0.33520748252672555, 0.3320417626140836, 0.33000000000000004]
Training accuracy for 2 hidden layer 0.8482459357046819
Test accuracy for 2 hidden layer 0.8458865864251146

Similarly, for two hidden layers, we got an accuracy of ~85% on the test set.

```
layers_dims = layer_triple_hidden_dims
parameters_3, costs_3 = L_layer_model(x_train, y_train, layers_dims, learning_rate = 0.9, num_iterations = 10000, print_cost = False)
print(costs_3)
pred_train3 = predict(x_train, y_train, parameters_3)
pred_test3 = predict(x_test, y_test, parameters_3)
acc_train3 = np.sum((pred_train3 == y_train)/y_train.shape[1])
acc_test3 = np.sum((pred_test3 == y_test)/y_test.shape[1])
print("Training accuracy for 3 hidden layer", np.sum((pred_train3 == y_train)/y_train.shape[1]))
print("Test accuracy for 3 hidden layer", np.sum(np.sum((pred_test3 == y_test)/y_test.shape[1])))
```

✓ 3m 3.6s Python

Cost after iteration 9999: 0.5482897284508244
 [0.6931471716988858, 0.5482897431393837, 0.54828974227102, 0.5482897413083493, 0.5482897402268161, 0.5482897389925453, 0.5482897375662718, 0.5482897363541491, 0.5482897351420274, 0.5482897339299051]
 Training accuracy for 3 hidden layer 0.7624067962351793
 Test accuracy for 3 hidden layer 0.7572899863506637

And finally for three hidden layers the accuracy fell down to ~76%. Thus, on trying different numbers of nodes for these setups we saw, more or less, a similar pattern of falling accuracy as we increased the layers and also fluctuating the learning rate for the gradient descent. Now, we thought of using “tanh” and see if we get a different result keeping the number of iterations constant in all cases.

8.2. Neural Networks with “Tanh()”

Now similarly the same process was done and the activation function used for ‘L-1’ layers was the ‘hyperbolic tan’ and this improved the accuracy a bit, but there was no significant improvement.

```
For Single Hidden Layer

layers_dims = layer_single_hidden_dims
parameters_1, costs_1 = l_layer_model(x_train, y_train, layers_dims, learning_rate = 2, num_iterations = 10000, print_cost = False)
print(costs_1)
pred_train1 = predict(x_train, y_train, parameters_1)
pred_test1 = predict(x_test, y_test, parameters_1)
acc_train1 = np.sum((pred_train1 == y_train)/y_train.shape[1])
acc_test1 = np.sum((pred_test1 == y_test)/y_test.shape[1])
print("Training accuracy for 1 hidden layer", np.sum((pred_train1 == y_train)/y_train.shape[1]))
print("Test accuracy for 1 hidden layer", np.sum((pred_test1 == y_test)/y_test.shape[1]))

Cost after iteration 9999: 0.3284363433771903
[0.6932679125082907, 0.37481146069673676, 0.3533980290894347, 0.34599533741188854, 0.34234857110081396, 0.33912875271388526, 0.3361684770967711, 0.33320875271388526, 0.33020875271388526, 0.32720875271388526]
Training accuracy for 1 hidden layer 0.8469930326365974
Test accuracy for 1 hidden layer 0.8408611490259335
```

Thus, for a single hidden layer with the same criteria as above we get the accuracy of 84.1%.

```
For Two Hidden Layers

layers_dims = layer_double_hidden_dims
parameters_2, costs_2 = l_layer_model(x_train, y_train, layers_dims, learning_rate = 2, num_iterations = 10000, print_cost = False)
print(costs_2)
pred_train2 = predict(x_train, y_train, parameters_2)
pred_test2 = predict(x_test, y_test, parameters_2)
acc_train2 = np.sum((pred_train2 == y_train)/y_train.shape[1])
acc_test2 = np.sum((pred_test2 == y_test)/y_test.shape[1])
print("Training accuracy for 2 hidden layer", np.sum((pred_train2 == y_train)/y_train.shape[1]))
print("Test accuracy for 2 hidden layer", np.sum((pred_test2 == y_test)/y_test.shape[1]))

Cost after iteration 9999: 0.3186993421217179
[0.693146872609742, 0.38198211939783355, 0.35319379920184546, 0.34048618258479113, 0.33361769559342663, 0.320370266756099, 0.3326814683310312, 0.320370266756099, 0.320370266756099, 0.320370266756099]
Training accuracy for 2 hidden layer 0.8524019068573527
Test accuracy for 2 hidden layer 0.845452289365926
```

For the network with two hidden layers we get an accuracy of 84.5%.

```
For Three Hidden Layers

layers_dims = layer_triple_hidden_dims
parameters_3, costs_3 = l_layer_model(x_train, y_train, layers_dims, learning_rate = 2, num_iterations = 10000, print_cost = False)
print(costs_3)
pred_train3 = predict(x_train, y_train, parameters_3)
pred_test3 = predict(x_test, y_test, parameters_3)
acc_train3 = np.sum((pred_train3 == y_train)/y_train.shape[1])
acc_test3 = np.sum((pred_test3 == y_test)/y_test.shape[1])
print("Training accuracy for 3 hidden layer", np.sum((pred_train3 == y_train)/y_train.shape[1]))
print("Test accuracy for 3 hidden layer", np.sum((pred_test3 == y_test)/y_test.shape[1]))

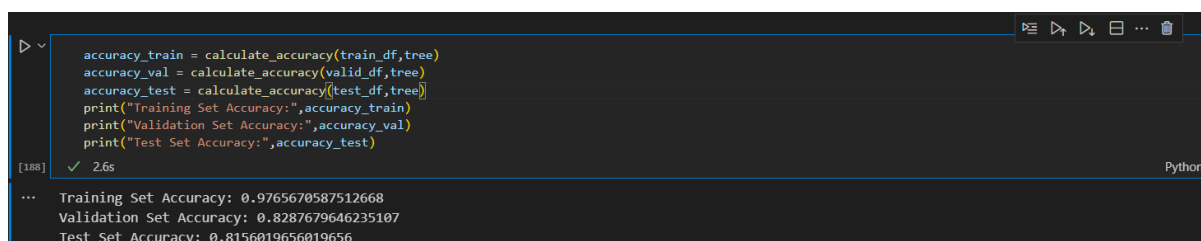
Cost after iteration 9999: 0.3567792384423222
[0.6931471340894803, 0.5482897052378135, 0.5482896424964916, 0.5482894261774786, 0.5482872941994068, 0.43894405313347656, 0.40819115696568914, 0.43894405313347656, 0.43894405313347656, 0.43894405313347656]
Training accuracy for 3 hidden layer 0.8226072607260728
Test accuracy for 3 hidden layer 0.8234272242213672
```

And for a network with three hidden layers we get an accuracy of ~82.3%. Thus, we also changed the learning rates when tanh() was used, and took it as 2. But overall, for this problem it seems that increasing the number of hidden layers didn't help improve the result. And also after trying multiple configurations we got an accuracy ranging from 82% to ~85%.

9. Results

So, after applying all the algorithms to the given dataset, we got the following results.

The data used was successfully used to generate a decision tree which could classify our given data into the two classes. The Training Set Accuracy: 0.9765670587512668, Validation Set Accuracy: 0.8287679646235107, Test Set Accuracy: 0.8156019656019656 was this for the fully grown decision tree.

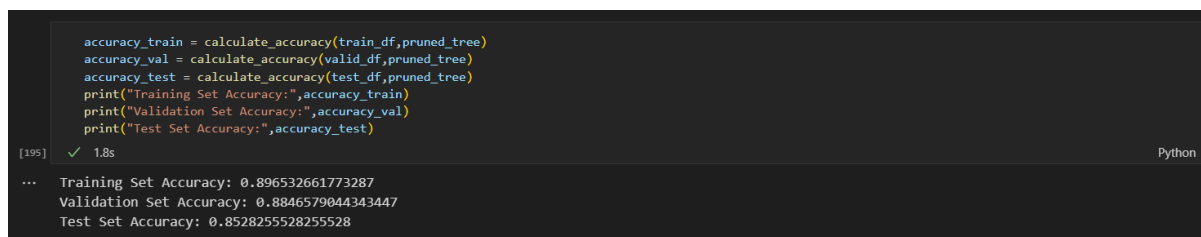


```
accuracy_train = calculate_accuracy(train_df,tree)
accuracy_val = calculate_accuracy(valid_df,tree)
accuracy_test = calculate_accuracy(test_df,tree)
print("Training Set Accuracy:",accuracy_train)
print("Validation Set Accuracy:",accuracy_val)
print("Test Set Accuracy:",accuracy_test)
```

[188] ✓ 2.6s Python

... Training Set Accuracy: 0.9765670587512668
Validation Set Accuracy: 0.8287679646235107
Test Set Accuracy: 0.8156019656019656

After applying post-pruning or “Reduced Error Pruning” we got a smaller decision tree. The Training Set Accuracy: 0.896532661773287, Validation Set Accuracy: 0.8846579044343447, Test Set Accuracy: 0.8528255528255528 was this for the pruned tree. We can see that the accuracy on the test set has increased, despite having lesser accuracy on the training set, providing us a better prediction. Since, we need to bring out the generality from the data, rather than overfitting the training data, the pruned tree is clearly the better choice amongst the two.



```
accuracy_train = calculate_accuracy(train_df,pruned_tree)
accuracy_val = calculate_accuracy(valid_df,pruned_tree)
accuracy_test = calculate_accuracy(test_df,pruned_tree)
print("Training Set Accuracy:",accuracy_train)
print("Validation Set Accuracy:",accuracy_val)
print("Test Set Accuracy:",accuracy_test)
```

[195] ✓ 1.8s Python

... Training Set Accuracy: 0.896532661773287
Validation Set Accuracy: 0.8846579044343447
Test Set Accuracy: 0.8528255528255528

Thus, this was the result for the optimal tree vs the freely growing tree.

Now, we combine the dataset and train a tree. The following are the images for the combined tree dataset split into test and train.

Thus here, we have a dataset of 48,842 sample elements. The train set has 32,724 elements and the test set has 16,118 elements.

```
[57] ✓ 2.7s Python
... Training Set Accuracy: 0.9760726072607261
Test Set Accuracy: 0.8240476485916367
```

```
[58] ✓ 2.0s Python
... Training Set Accuracy: 0.8980259137024813
Test Set Accuracy: 0.8782727385531703
```

Thus this is the resulting accuracy from the tree trained for the combined data set. Both the trees are not the same. But we can see here that the bigger tree is more optimal. The test accuracy after pruning is more than the previous tree. There is nearly a surge of 2.5% in the test accuracy for the tree trained on combined data. Although its size is bigger, it seems that the tree is classifying the unseen data better. This tree has 6907 nodes and a depth of 128, whereas after pruning the tree shrunk to 1519 nodes and a depth of 116. Since, we were able to shrink the tree, it means we have avoided many questions (that was referred to earlier) or rules, and were able to grasp the generality of the data. The problem here was also the one faced earlier, the long waiting time for training the tree. But it was found to be more optimal than the previous tree with an accuracy of 85.28% whereas the tree trained from combined data and after pruning gave an accuracy of 87.83%.

The result for the random forest:

We got an accuracy of 84.861%, which is less than the accuracy of the previous trees. This accuracy corresponds to the forest having 1000 trees (`n_estimators = 1000`). The criterion used for the random forest classification here does not show any change for “entropy” or “log_loss”, the accuracy obtained are same

```
cm = confusion_matrix(Y_pred, Y_test)
print(cm)
✓ 0.0s
[[10486  1406]
 [   874  2294]]

accuracy_score(Y_test, Y_pred)
✓ 0.0s
0.848605577689243

print(classification_report(Y_test, Y_pred))
✓ 0.0s
```

	precision	recall	f1-score	support
0.0	0.88	0.92	0.90	11360
1.0	0.72	0.62	0.67	3700
accuracy			0.85	15060
macro avg	0.80	0.77	0.78	15060
weighted avg	0.84	0.85	0.84	15060

The result for Logistic Regression:

The logistic regression model worked fine on the dataset and could yield an accuracy of ~82% in the test set. This was easy to implement and gave a good result.

The result for Neural Networks:

The neural networks also worked fine on the dataset and gave an accuracy of ~85% percent for the network with one or two hidden layers and then the accuracy kept going down. Thus, it seems that the model with one or two hidden layers works fine and to keep things simple it is better we take a single hidden layer. Analyzing this with the logistic regression model it seems that these two were almost at par, with neural networks having a slight edge above them. As when we tried to run the model with no hidden layers it is equivalent to the logistic regression case and they gave almost similar results.

The result for Naive Bayes Classifier:

The classifier gave an accuracy of ~80% on the testing data. It was tested with both inbuilt libraries and custom made functions for the posterior, prior probabilities calculations and Gaussian Distribution function for the prediction of class conditional probabilities.

10. Conclusion

In conclusion, this study examined the performance of several popular machine learning algorithms, including Decision Trees, Random Forest, Logistic Regression, Naive Bayes, and Neural Networks. The goal was to evaluate their effectiveness in predictive modeling and understand the general trends observed in their performance.

Upon analyzing the results, it was found that the decision tree algorithm exhibited the highest accuracy on the test set, closely followed by neural networks. This suggests that decision trees have a strong predictive power and can effectively capture complex relationships within the data. Neural networks, known for their ability to learn intricate patterns and nonlinearities, also demonstrated promising performance.

Random forest, a powerful ensemble learning method built on decision trees, often showcases robustness and generalization ability. Logistic regression, a linear model widely used for binary classification tasks, can provide interpretable insights and perform well in situations where the relationship between predictors and the response variable is approximately linear. Naive Bayes, based on probabilistic principles, tends to work well in cases where the assumption of feature independence holds.

It is important to note that the observed performance trends may vary depending on the specific dataset and the characteristics of the problem at hand. Consequently, these conclusions should be interpreted within the context of this study and should not be generalized to all scenarios.

In summary, the study demonstrates the comparative performance of Decision Trees, Random Forest, Logistic Regression, Naive Bayes, and Neural Networks. While decision trees and neural networks exhibited the highest accuracy in this particular study, the choice of algorithm should be based on the specific requirements of the problem, the interpretability desired, and the nature of the available data. Further research and experimentation are encouraged to explore the performance of these algorithms across a broader range of datasets and problem domains.

11. References

- Sebastian Mantey. (2019, March 1). Coding a Decision Tree from Scratch in Python[Video file]. Retrieved from https://www.youtube.com/watch?v=y6DmpG_PtN0&list=PLPOTBrypY74xS3WD0G_uzqPjCQfU6lRK-
- edureka!. (2022, March 10).Random Forest Algorithm[Video file]. Retrieved from <https://www.youtube.com/watch?v=3LQl-w7-FuE>