Building a Machine Learning Model for the Prediction of Dependent Variables in the Given Datasets

Assignment 2

Machine Learning

IOT Track

Muhammad Farrukh Mehmood

Reg no. 399602 Fall/22

Table of Contents

List of Figures

List of Tables

Logistic Regression

The objective of the assignment is to develop logistic regression and decision tree models to predict the dependent variables (income and selection decision) based on the independent variables following datasets.

I. Income data set
II. Selection dataset

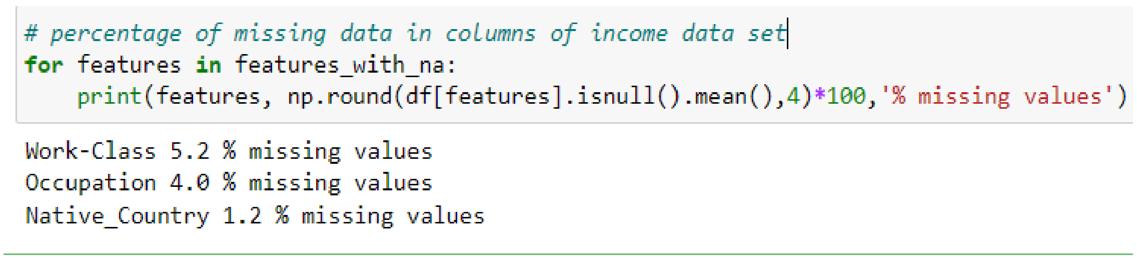
1.1 Road map

To achieve the target following procedural steps were adopted:

- Data cleaning
- Data visualization
- Dividing the data into the label(dependent variable) and features (independent variables)
- Splitting the data into training and test sets
- Finding an appropriate model with suitable hyperparameters
- Training the model with training data
- Validating the model on train and test sets with different scoring parameters

Exploratory Data Analysis

Firstly, the data were checked for missing or null values. To quantify the null values “isnull().any()” command was used. The command reported some null values in income data set.



```
# percentage of missing data in columns of income data set
for features in features_with_na:
    print(features, np.round(df[features].isnull().mean(),4)*100,'% missing values')
```

```
Work-Class 5.2 % missing values
Occupation 4.0 % missing values
Native_Country 1.2 % missing values
```

Figure 1: Missing value report for income data

To visualize the missing values, seaborn’s heatmap function was used.

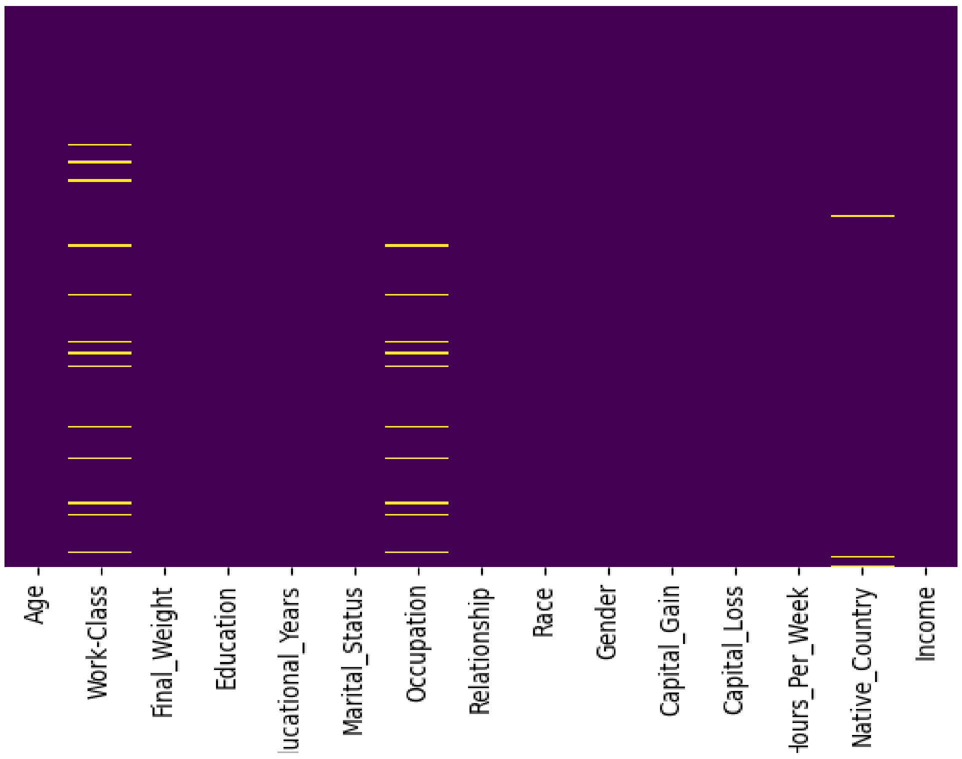


Figure 2: Missing data heatmap

Since the only categorical data contained missing values, the missing positions were filled with the most frequent values occurring in the respective column. The high-frequency values in ‘Work-class’, ‘Occupation’, and ‘Native-Country’ are visualized with the help of the following histograms.

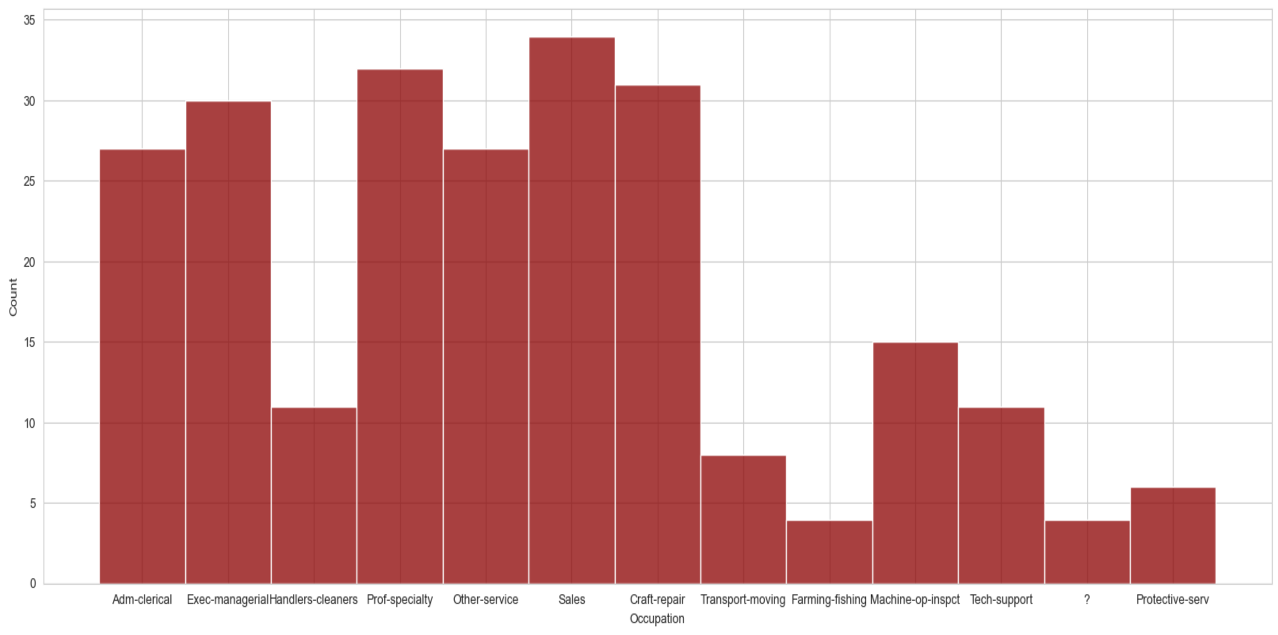


Figure 3: Occupation histogram

'Sales' appears to be frequent in the Occupation column.

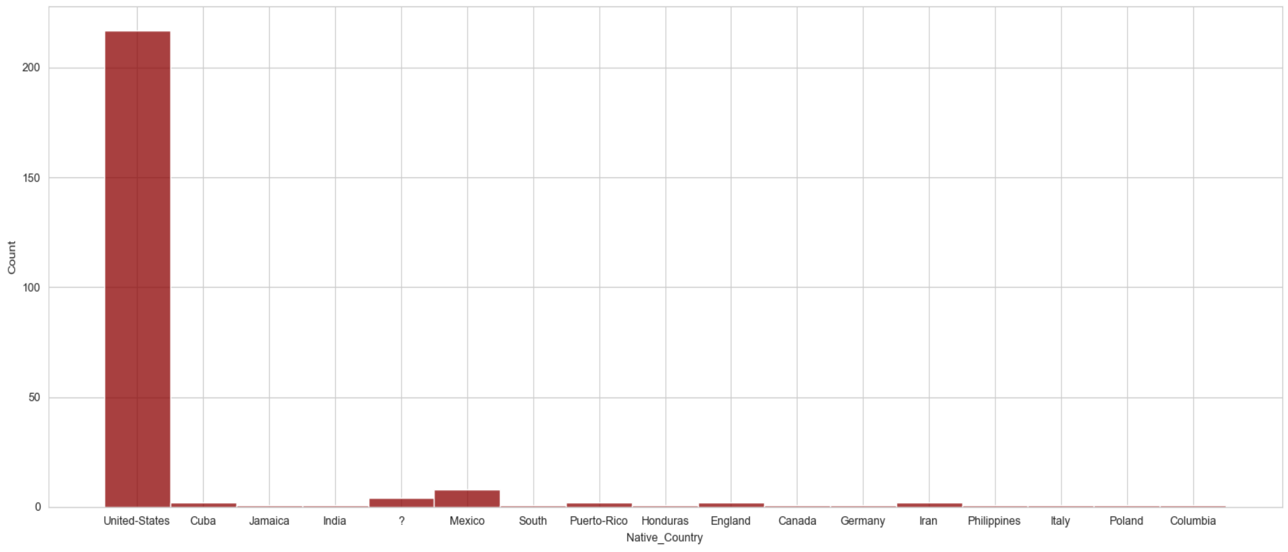


Figure 4: Native country histogram

‘United States' appears to be frequent in the ‘Native-Country’ column.

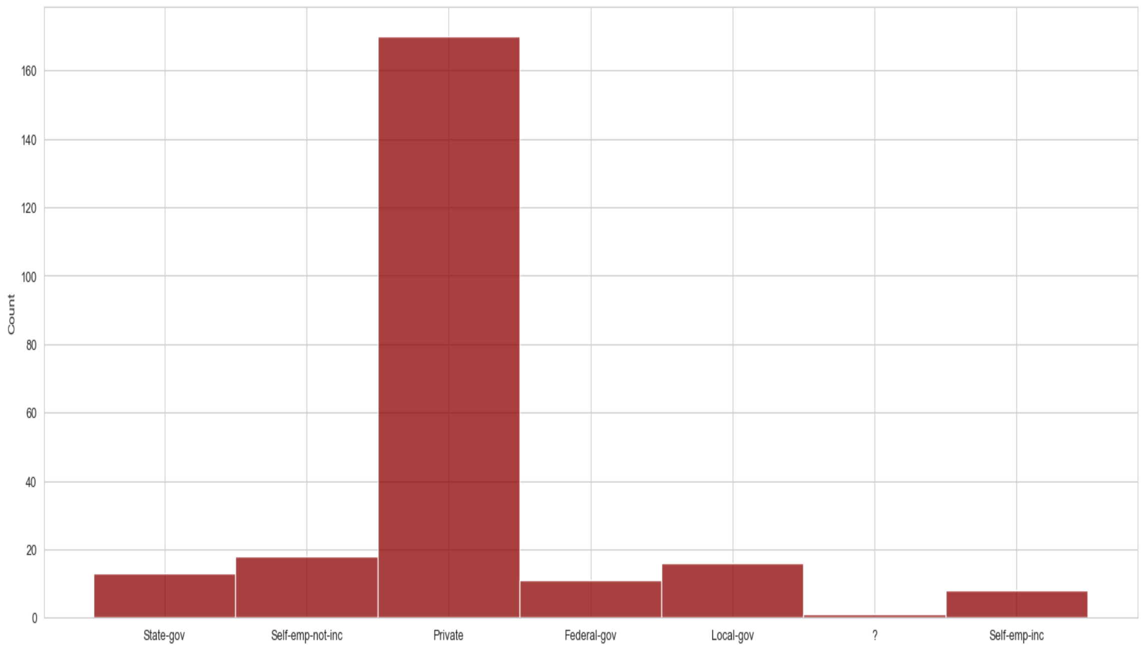


Figure 5: Work-class histogram

‘Private' appears to be frequent in the Work-class column.

After the replacement with the most frequent values, there remained no missing data. This can be visualized from the following chart.

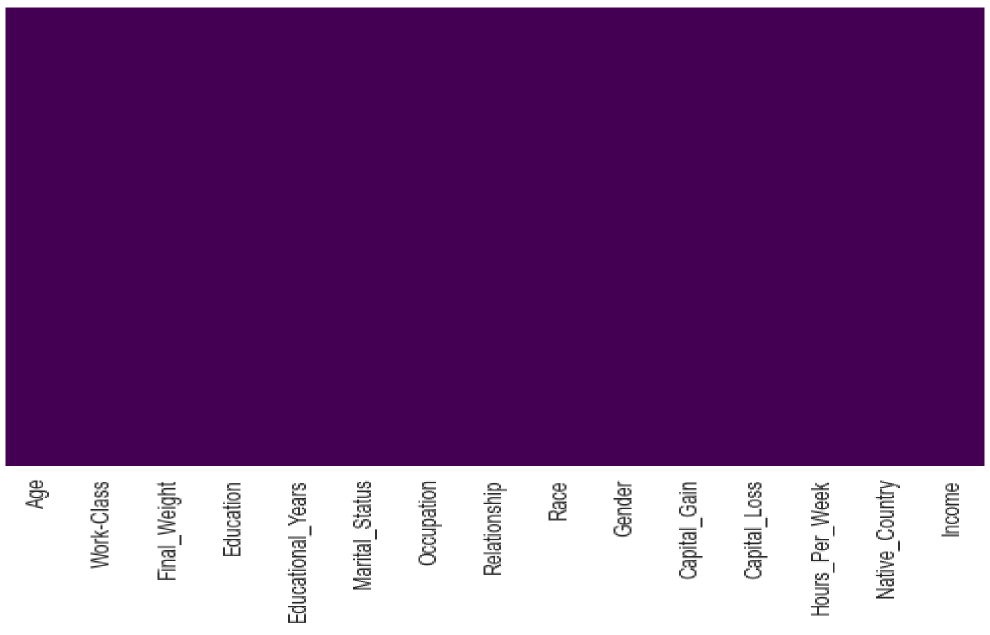


Figure 6: Heatmap of data after replacements

2.1 Value Counts

The following chart shows most of the people have income less than or equal to 50 thousands.

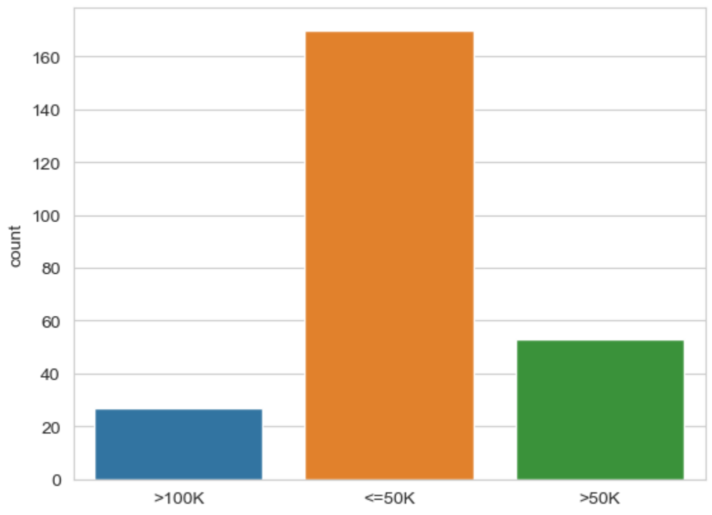


Figure 7: Income class count

The following chart shows the gender-wise distribution of income.

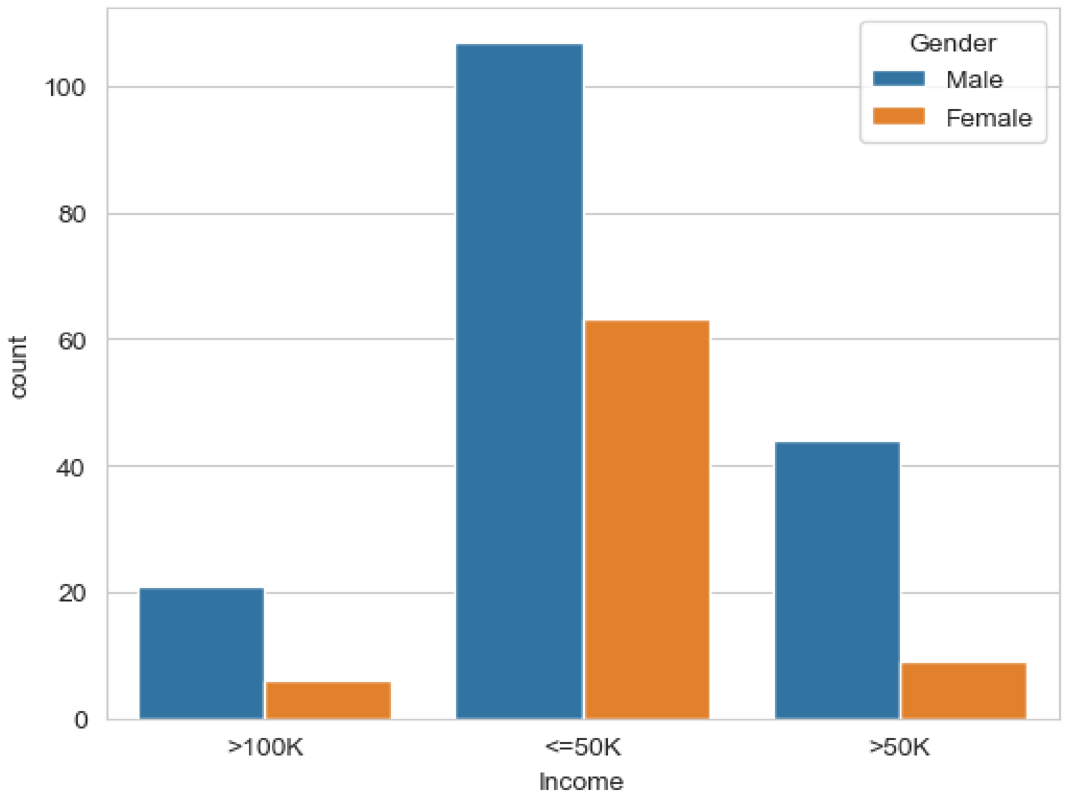


Figure 8: Gender wise distribution of income

Data preprocessing

3.1 Converting the Categorical Data into numerical and Normalization

Most commonly attributes like marital status, race and relationship do not affect the income of the person, so these features were dropped off. Categorical data were converted to numerical data through One Hot encoding.

- Dummy variables were created corresponding to the categorical attributes using ‘get_dummies()’ command
- These dummy variables were concatenated with the actual data set.
- The actual categorial attributes columns were dropped off from the data set to get pure numerical dataset

The implementation of the above procedure can be seen in the code. The data were normalized based on MinMax scaling technique.

3.2 Dividing the Actual data into ‘feature’ and ‘label’ datasets

The actual datasets were divided into two new data sets. All the independent variables were assigned a dataset ‘feature’ and all the single dependent variable was assigned the dataset ‘label’.

3.3 Splitting the data into test and training data set

For model building and validation, the data were split into two parts: 80 percent data for model training and 20 percent data for model testing. For this purpose, ‘train_test_split()’ function was used from ‘sklearn’ library.

3.4 Assigning new classes

The income column of the dataset was divided into 3 separate binary columns. Details are below

- Class 0 contains income less than 50k.
- Class 1 contains income between 50-100k.
- Class 2 contains income above 100k.

4 Training the model

4.1 Logistic Regression

Since the dataset contained 3 classes, logistic regression was implemented using a one-vs-all approach, with a self-written code. Following functions were defined:

- Sigmoid activation function
- Cost function
- Gradient Descent
- Functions for prediction and accuracy calculations

4.2 Regularization

To address the problem of overfitting, L2 regularization was performed with a self-written code. The following plot shows the training and inference accuracies based on given values of regularization parameters.

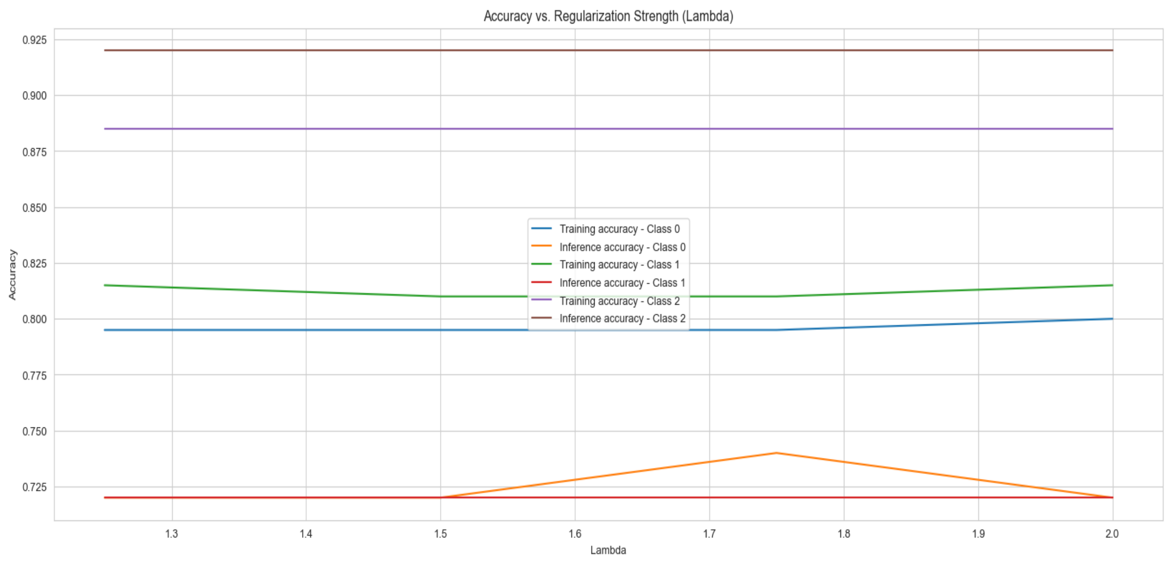


Figure 9: Trainind and Inference accuracy vs regularization parameter

The regularization was tested for different values of lambda. The plot gives the information that we get minimum difference between the train and inference accuracies when lambda is 1.75.

4.3 Effect of Different Initializations

The model was tested for following initialization values of parameters and their effect was observed on cost function and activation function plots.

1. All the parameters initially 0
2. All the parameters initially 1
3. All the parameters initially 0.0001

Plots of cost function and activation function when all the parameters initially 0

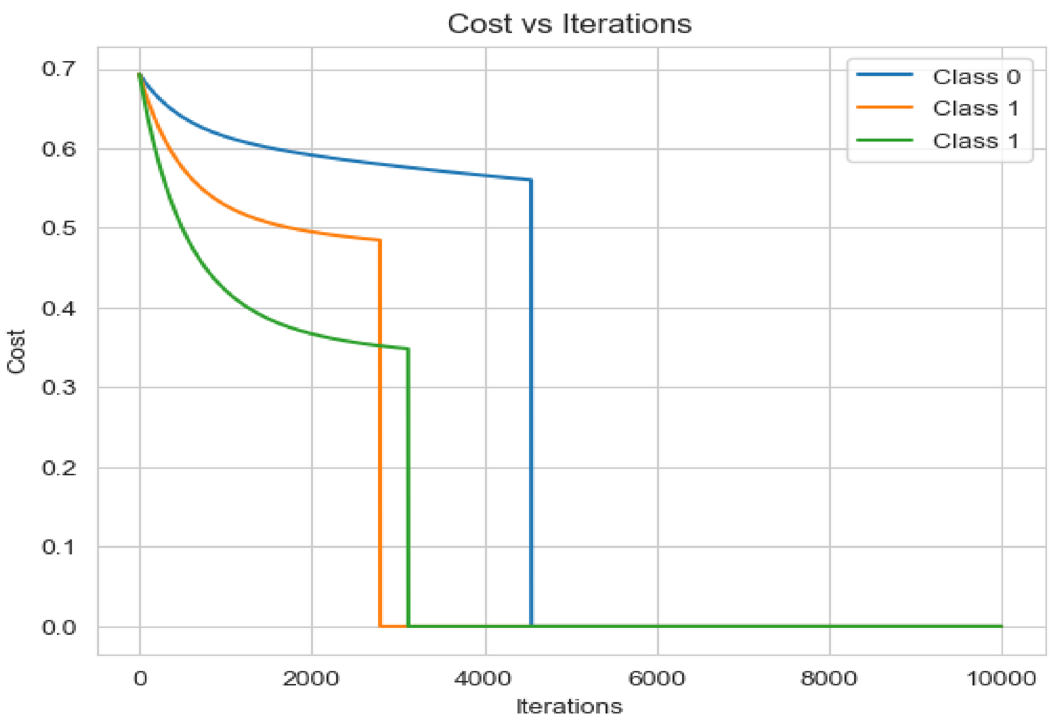


Figure 10: Cost when all the parameters initially 0

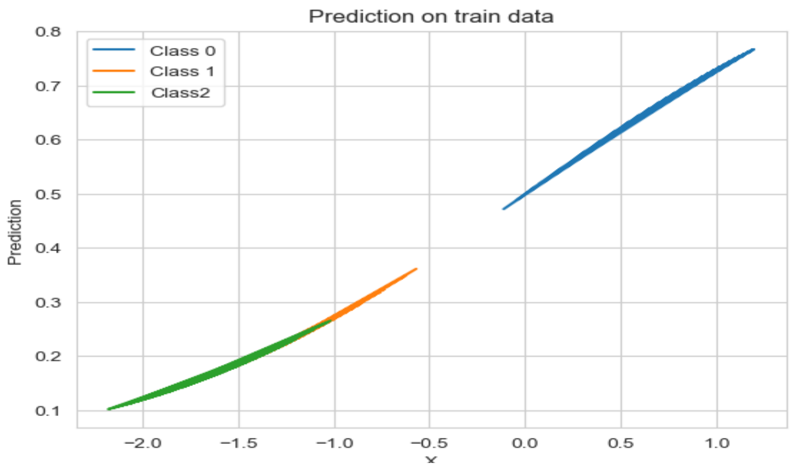


Figure 11: Activation function when all the parameters initially 0

Plot of cost function and activation function when all the parameters initially 1

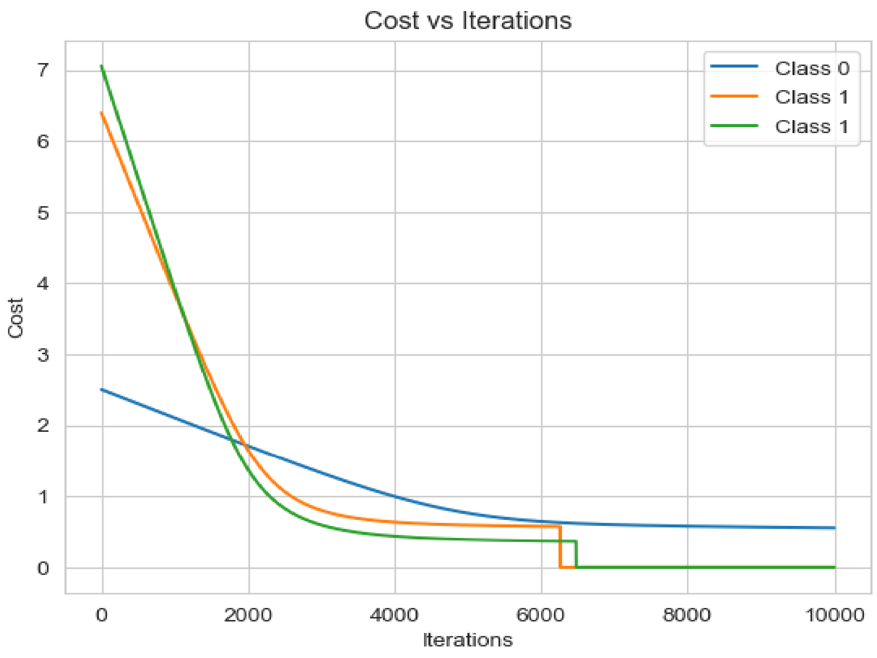


Figure 12: Cost when all the parameters initially 1

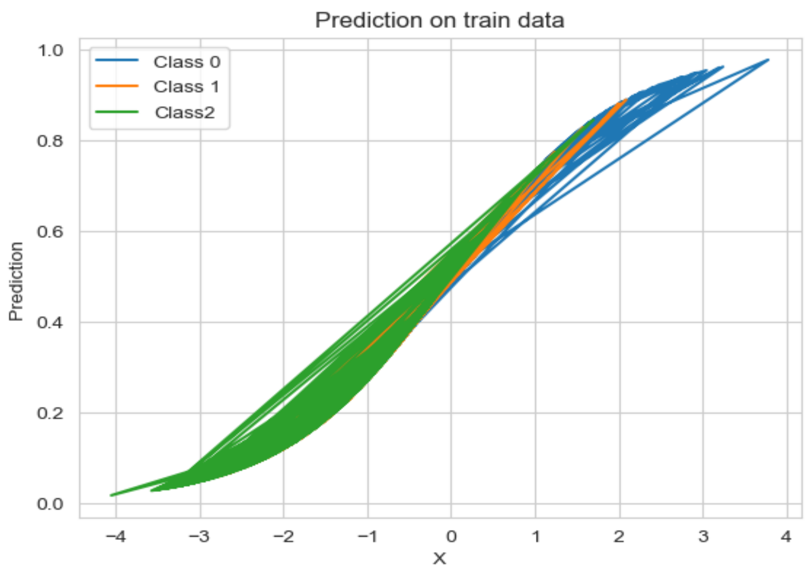


Figure 13: Activation Function when all the parameters initially 1

Plot of cost function and activation when all the parameters initially 0.0001

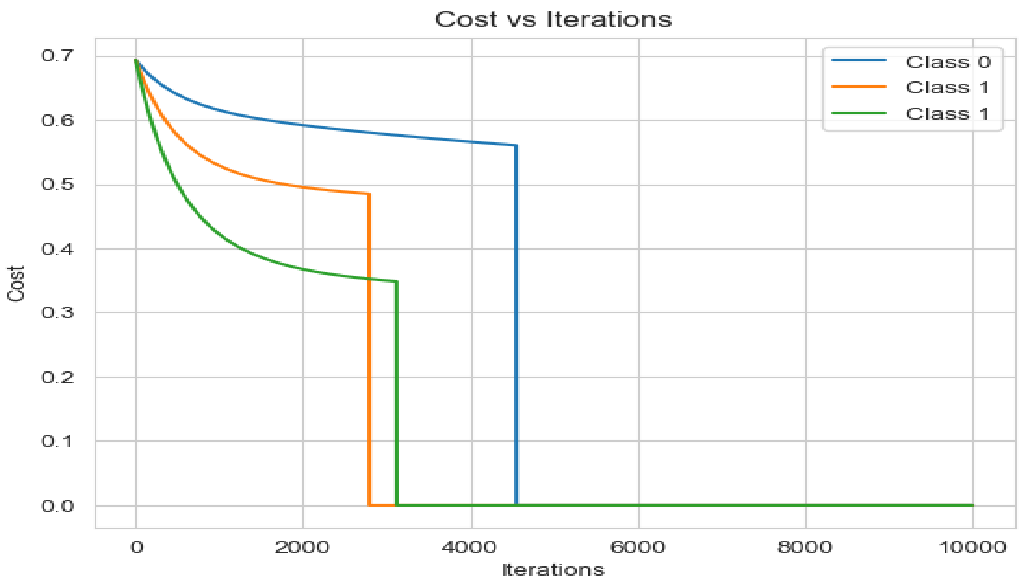


Figure 14: Cost when all the parameters initially 0.0001

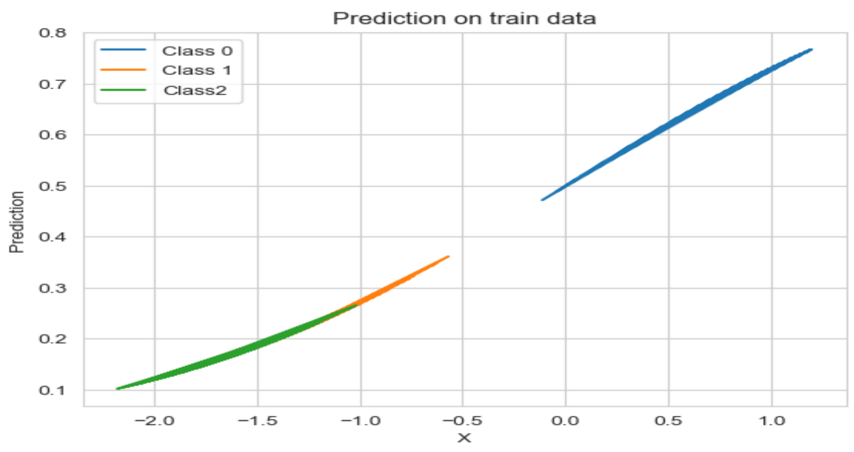


Figure 15: Activation function when all the parameters initially 0.0001

4.4 Optimal values for Hyper Parameters

Based on the observations from the above plots following values were chosen for the model

Table 1: Selected hyperparameter values

Hyperparameter	**Value**
Feature Scaling	MinMax scaler
Regularization technique	L2 regression
Regularization parameter	1.75
Learning Rate	0.01
Tolerance	1e-5
Initial weights value	0.0001

Minmax scaler was chosen for feature scaling because it gives the best suitable for logistic regression when the data contains binary values.

4.5 Evaluation of hyperparameters

The accuracy of the model was tested based on the hyperparameter values obtained, and various evaluation metrics were used to access its performance.

4.6 Gradient Convergence:

For all the classes, the gradient converged before reaching 3000 iterations.

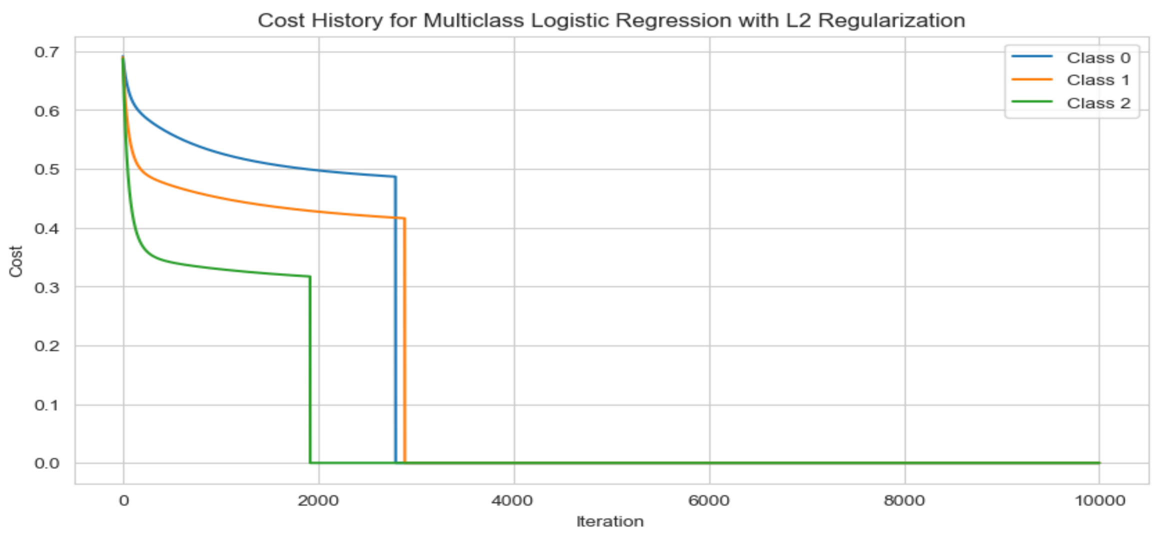


Figure 16: Gradient convergence for logistic regression

Table 2: Training and Inference accuracy of the trained model

	Train Accuracy	**Inference Accuracy**
Class 0	0.785	0.72
Class 1	0.825	0.76
Class 2	0.885	0.92

5 Decision Tree

A decision tree algorithm was implemented on both datasets. All the values of features were of the categorical type which was converted to numeric data through label encoding. Sklearn's DecisionTreeClassifier module was tested for different values of hyperparameters.

5.1 Decision Tree for the income data set

For the income data set decision tree model was built using both gini impurity and entropy for different depth values. The following plot shows the comparison of accuracies.

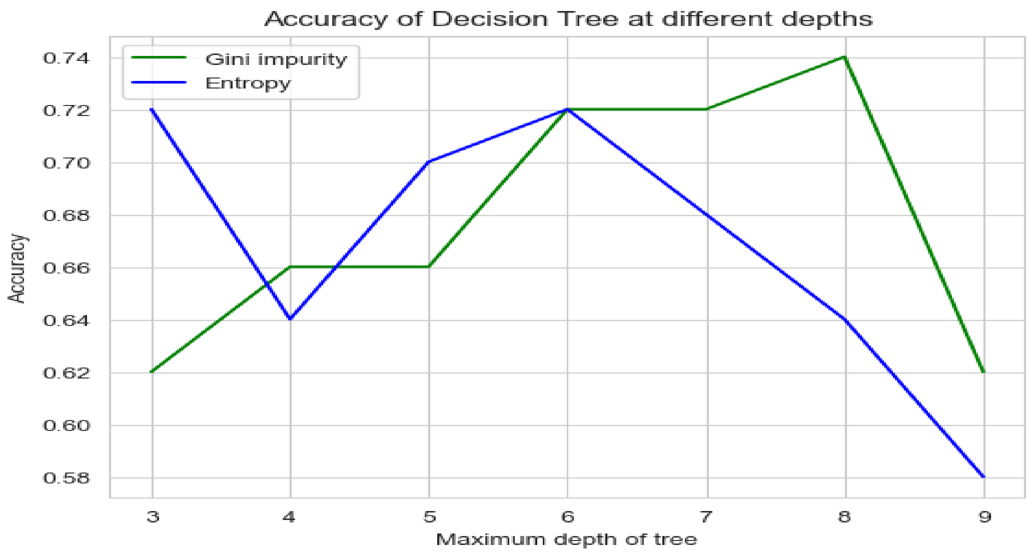


Figure 17: Accuracy vs tree depth

For Gini impurity, maximum accuracy is achieved at depth of 8, and the entropy gave maximum accuracy at the depth of 3 and 6.

5.2 Decision Tree for the selection data set

Due to the limited size of the selection data and the requirement to split it into test and train sets, a decision tree of depth 2 was the maximum that could be built. For both gini impurity and entropy, the accuracy remained 0.5.

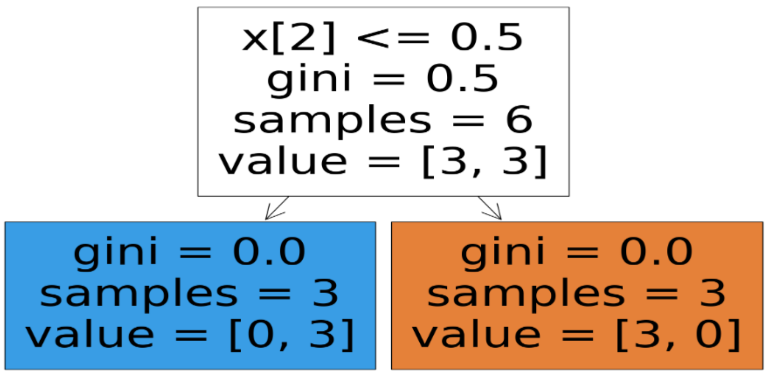


Figure 18: Decision tree based on gini impurit

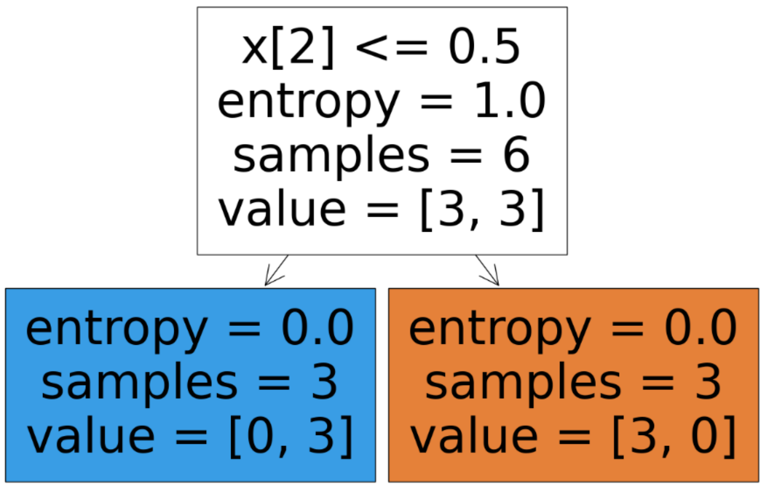


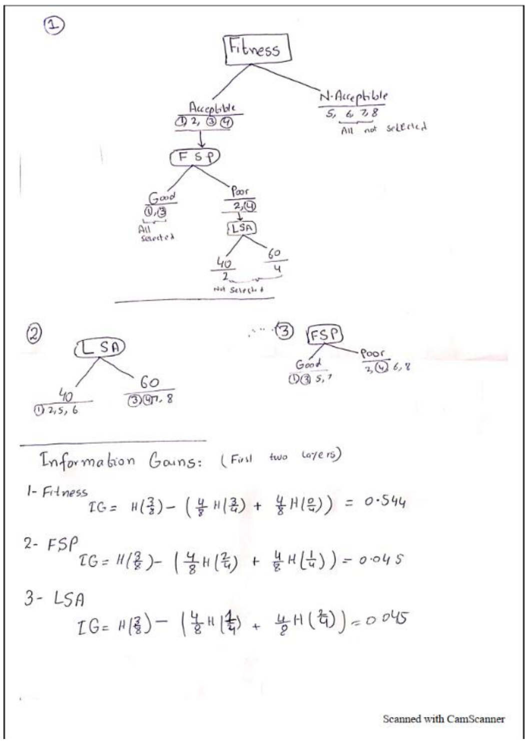
Figure 19: Decision tree based on entropy

5.2.1 Some results of decision tree

Table 3: Decision tree results

	Local Season Average	Foreign Season Performance	Fitness Test	Decision (Predicted)	Decision (Actual)
Entropy	40+	Poor	Acceptable fitness	Not selected	Not selected
	40+	Good	Acceptable fitness	Not selected	Selected
Gini Impurity	40+	Poor	Acceptable fitness	Not selected	Not selected
	40+	Good	Acceptable fitness	Not selected	Selected

5.2.2 Manually built decision tree



Information Gains: (First two layers)

1- Fitness

$IG = H(\frac{3}{8}) - (\frac{4}{8} H(\frac{3}{4}) + \frac{4}{8} H(\frac{0}{4})) = 0.544$

2- FSP

$IG = H(\frac{3}{8}) - (\frac{4}{8} H(\frac{2}{4}) + \frac{4}{8} H(\frac{1}{4})) = 0.045$

3- LSA

$IG = H(\frac{3}{8}) - (\frac{4}{8} H(\frac{1}{4}) + \frac{4}{8} H(\frac{2}{4})) = 0.045$

5.2.3 Comparison

Information gains for all the attributes taking one at time for root node were calculated manually. Maximum gain was found to be for Fitness attribute which is in agreement with gain obtained from python code. A tree was built taking Fitness as root node.