Chittagong University of Engineering & Technology

Department of Computer Science & Engineering

Course Name: Machine Learning (Sessional)
Course Code: CSE - 464

Final Report on

Machine Learning Algorithms Implementation

by

Md. Shimul Mahmud

ID: 1804021

Submitted to

Md. Rashadur Rahman
Lecturer, Dept of CSE, CUET
Hasan Murad
Lecturer, Dept of CSE, CUET

Date of Submission: 25th August 2023

Table of Contents

List of Figures

Abstract

This report contains the five basic machine learning models, such as the Apriori algorithm, Multivariable Linear Regression, K-means Clustering, Decision Trees, and Artificial Neural Network (ANN). The theoretical background of each algorithm, implementation techniques, dataset preparation, data preprocessing, evaluation techniques, and practical applications are explored. Through in-depth study and discussion, this report aims to provide insights into the advantages, disadvantages, and practical considerations associated with each method.

1 Introduction

To evaluate students' understanding of and proficiency with these potent tools, this study offers a comprehensive review of five crucial machine learning algorithms. Our objective is to provide a thorough grasp of each algorithm's theoretical underpinnings, practical implementations, and real-world applications by delving thoroughly into its intricacies. The report's importance stems from its critical function as a crucial evaluation of students' machine-learning knowledge. These algorithms include a variety of approaches, each with unique advantages and valuable uses.

These techniques address a variety of machine learning problems, including discovering correlations in data (Apriori), predicting outcomes (Multivariable Linear Regression), grouping related data points (K-Means grouping), and decision-making using decision trees and artificial neural networks. Here is an overview of five algorithms:

i. Apriori Algorithm:

In transactional databases, the association rule mining method Apriori is used to identify frequently occurring item sets. It pinpoints connections between things that frequently occur together. Apriori develops association rules based on confidence and lift measurements and an itemset frequency threshold.

ii. Multivariable Linear Regression:

A statistical method known as multivariable linear regression models the association between a dependent variable and a number of independent factors. To identify the linear equation that fits

the data the best, it estimates coefficients. This approach is used to comprehend how several factors interact to affect a result.

iii. K-Means Clustering:

K-Means An unsupervised process called clustering is used to organize related data points into clusters. Based on the average of the data points inside each cluster, it divides the data into k clusters. To reduce the sum of squared distances between data points and cluster centers, the algorithm iteratively updates cluster centers.

iv. Decision Tree:

By dividing options into a number of nodes and branches, the decision tree is a tree-like model that aids in decision-making. Each node represents a characteristic or query, while the branches denote potential responses. The tree can categorize or predict outcomes based on feature circumstances because it is constructed utilizing entropy and information gain.

v. Artificial Neural Network (ANN):

A computational model called ANN is modeled after the neural network of the human brain. It is made up of layered networks of interconnected nodes (neurons). To reduce prediction mistakes, data is processed using layers, and weights are changed throughout training. A complex task is no match for ANNs, which use a hierarchical framework to identify patterns in data.

2 Algorithm Descriptions and Implementations

2.1 Apriori Algorithm

A data mining method called association rule mining is used to find intriguing connections or patterns in huge datasets. Finding links between objects or properties in a dataset is the main focus. These relationships are shown as "if-then" rules, where the existence or occurrence of one item implies the existence or occurrence of another.

Market basket analysis is one typical scenario in which association rule mining is used. This method, for instance, can be used in a retail setting to determine which products are frequently bought together, enabling firms to make wise choices about product placement, cross-selling, and promotions.

There are some common terms in association rules mining such as

i. **Generating Frequent k-itemsets:** The frequent k-itemsets found in the preceding phase are used by the algorithm to generate candidate itemsets of size k+1 iteratively. This is accomplished by merging pairs of k-itemsets and pruning those that have infrequently occurring subsets.

ii. **Pruning Infrequent Itemsets:** The method prunes the generated candidate itemsets every iteration if they don't reach the minimum support criterion. This lowers the amount of itemsets that must be taken into account in succeeding iterations.

iii. **Generating Association Rules:** When all of the frequent itemsets have been identified, the algorithm generates association rules from these itemsets. For each frequent itemset, the algorithm generates all possible non-empty subgroups and calculates the confidence of each rule. The frequency with which the "if" portion of the rule leads to the "then" portion is quantified by confidence.

2.1.1 Dataset

The dataset contains some selling information for a market. The data features included some missing values, and the datasets are imbalanced too. Here is a snapshot of the provided store data as follows:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	shrimp	almonds	avocado	getables m	reen grape	ole weat fl	yams	ttage chee	nergy drin	lomato juic	w fat yogu	green tea	honey	salad	ineral wat	salmon	ioxydant ju	zen smoot	spinach	olive oil	
2	burgers	meatballs	eggs																		
3	chutney																				
4	turkey	avocado																			
5	ineral wat	milk	energy bar	ole wheat	green tea																
6	w fat yogurt																				
7	le wheat p	french fries																			
8	soup	light cream	shallot																		
9	zen vegeta	spaghetti	green tea																		
10	french fries																				
11	eggs	pet food																			
12	cookies																				
13	turkey	burgers	ineral wat	eggs	cooking oil																
14	spaghetti	champagne	cookies																		
15	ineral wat	salmon																			
16	ineral water																				
17	shrimp	chocolate	chicken	honey	oil	cooking oil	w fat yogurt														
18	turkey	eggs																			
19	turkey	fresh tuna	tomatoes	spaghetti	ineral wat	black tea	salmon	eggs	chicken	dark chocolate											
20	meatballs	milk	honey	french fries	protein bar																

Figure 01: A Snapshot of Store Data

2.1.2 Implementation

i. **Load Dataset:** Firstly, we load the store data using pandas and generate a dataset for transactions.

ii. **Generating Association Rules:** We generate an association rule from the transaction ID and items using apriori algorithm libraries for custom minimum support and minimum confidence.

```
Enter the minimum support: 20
Enter the minimum confidence: 50
All Association Rules:
Rule: chocolate -> nan
Support: 0.2052
Confidence: 1.0000
Lift: 1.0000

Rule: eggs -> nan
Support: 0.2081
Confidence: 1.0000
Lift: 1.0000

Rule: mineral water -> nan
Support: 0.2997
Confidence: 1.0000
Lift: 1.0000

Rule: spaghetti -> nan
Support: 0.2296
Confidence: 1.0000
Lift: 1.0000
```

Figure 02: Association Rules for Custom Minimum Support and Confidence

iii. **Displaying First k-numbers of Rules:** Here is the first k-th association rules for custom input,

```
Enter the number of association rules to display: 10
Top 10 Association Rules:
Rule: chocolate -> nan
Support: 0.2052
Confidence: 1.0000
Lift: 1.0000

Rule: eggs -> nan
Support: 0.2081
Confidence: 1.0000
Lift: 1.0000

Rule: mineral water -> nan
Support: 0.2997
Confidence: 1.0000
Lift: 1.0000

Rule: spaghetti -> nan
Support: 0.2296
Confidence: 1.0000
Lift: 1.0000
```

Figure 03: Association Rules for the First kth Custom Input.

2.1.3 Performance Evaluation

The performance of the apriori algorithm finds the patterns and gives the association rules, confidence level and support level. As there are no interesting or strong association rules generated with this minimum level of support and confidence, the associated result is shown as **nan**.

Minimum Support – 20%

Minimum Confidence – 50%

```
Rule: eggs -> nan
Support: 0.2081
Confidence: 1.0000
Lift: 1.0000

Rule: mineral water -> nan
Support: 0.2997
Confidence: 1.0000
Lift: 1.0000

Rule: spaghetti -> nan
Support: 0.2296
Confidence: 1.0000
Lift: 1.0000
```

Figure 04: Performance Evaluation for the Given Minimum Support and Confidence

2.2 Multivariable Linear Regression

Multiple independent factors' effects on a dependent variable are examined through multivariable linear regression. It attempts to reduce the discrepancy between expected and actual values by fitting a linear equation to data points. Each impact of an independent variable is represented by a coefficient. The model makes the following assumptions: linearity, error independence, constant error variability, and little multicollinearity. The method aids in quantifying linkages and outcome prediction based on variable interactions by estimating these coefficients.

2.2.1 Dataset

The property listing dataset contain multiple features such as title, bed, area, address, type and so on.

	A	B	C	D	E	F	G	H	I	J	K	L
1	title	beds	bath	area	adress	type	purpose	flooPlan	url	lastUpdate	price	
2	Eminent A	3	4	2,200 sqft	Block A, Ba	Apartment	For Rent	https://im	https://wv	########	50 Thousand	
3	Apartment	3	4	1,400 sqft	South Khul	Apartment	For Rent	https://im	https://wv	25-Jan-22	30 Thousand	
4	Smartly pri	3	4	1,950 sqft	Block F, Ba	Apartment	For Rent	https://im	https://wv	########	30 Thousand	
5	2000 Sq Ft	3	3	2,000 sqft	Sector 9, U	Apartment	For Rent	https://im	https://wv	########	35 Thousand	
6	Strongly St	3	4	1,650 sqft	Block I, Ba	Apartment	For Rent	https://im	https://wv	########	25 Thousand	
7	A nice resi	5	5	3,400 sqft	Gulshan 1,	Apartment	For Rent	https://im	https://wv	########	1.1 Lakh	
8	1600 Squa	3	3	1,600 sqft	Sector 6, U	Apartment	For Rent	https://im	https://wv	6-Aug-22	35 Thousand	
9	Let Us Hel	3	3	1,250 sqft	Block K, Ba	Apartment	For Rent	https://im	https://wv	4-Jan-23	23 Thousand	
10	An Extensi	3	4	2,150 sqft	Sector 10,	Apartment	For Rent	https://im	https://wv	28-Jun-22	40 Thousand	

Figure 05: Property Listing Datasets

2.2.2 Implementation

i. Load dataset:

We load the datasets using pandas mentioned in the figure 06.

ii. Data preprocessing:

As some of the features within the dataset are not needed to explore the multivariable linear regression, we dropped the unnecessary features from the dataset. The dropping columns are,

columns=['title', 'adress', 'type', 'purpose', 'flooPlan', 'url', 'lastUpdated']

```
1  data = pd.read_csv('/content/property_listing_data_in_Bangladesh.csv')
2  data = data.drop(columns=['title', 'adress', 'type', 'purpose', 'flooPlan', 'url', 'lastUpdated'])
3  data.head()
```

Figure 06: Load and Dropped the Data

	beds	bath	area	price
0	3	4	2,200 sqft	50 Thousand
1	3	4	1,400 sqft	30 Thousand
2	3	4	1,950 sqft	30 Thousand
3	3	3	2,000 sqft	35 Thousand
4	3	4	1,650 sqft	25 Thousand

Figure 07: After Dropping the Unnecessary Features

iii. Data Preparation:

Now we need to prepare the data that contains inconsistencies. The features of area, baths, price, and beds need to be regularized to remove inconsistencies in the data. After removing the inconsistency, the data looks like this:

```
[ ]  1  data.head()
```

	beds	bath	area	price
0	3	4	2200.0	50000.0
1	3	4	1400.0	30000.0
2	3	4	1950.0	30000.0
3	3	3	2000.0	35000.0
4	3	4	1650.0	25000.0

Figure 08: Data Preparation

After standardizing the data,

	beds	bath	area	price
0	0.057143	0.333333	0.060897	0.024999
1	0.057143	0.333333	0.035256	0.014999
2	0.057143	0.333333	0.052885	0.014999
3	0.057143	0.222222	0.054487	0.017499
4	0.057143	0.333333	0.043269	0.012499

Figure 09: Data Standardization

iv. Training Data

After completion of data preprocessing, we trained the data where the **"price"** feature was our target. We split the dataset into **80%** for training, **20%** for testing, and **20%** for validation. The model training used gradient descent and the cost compute function for 100 epochs and calculated training and validation losses for every epoch.

learning_rate = 0.01

num_iterations = 100

```
Epoch 1: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 2: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 3: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 4: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 5: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 6: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 7: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 8: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 9: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 10: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 11: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
Epoch 12: Training Loss = 0.0002650224893794115, Validation Loss = 0.00016571343809800143
```

Figure 10: Training and Validation Loss

2.2.3 Performance Evaluation

After completing 100 epochs of training our model, the calculated total loss of our model are as follows,

Test Loss: 0.0005213189566025826

Mean Squared Error (MSE): 0.0010426379132051653

Root Mean Squared Error (RMSE): 0.032289904199380416

Mean Squared Error (MSE) on Test Set: 0.0010426379132051653

Root Mean Squared Error (RMSE) on Test Set: 0.032289904199380416

2.3 K-Means Clustering

K-means A machine learning process called clustering divides a set of data points into groups or clusters according to how similar they are. Each data point is assigned to the cluster with the nearest mean (central) after the data is divided into a preset number of clusters. Up until convergence, the algorithm iteratively changes cluster centers and reassigns points. The algorithm works as follows,

1. Choose the desired number of clusters (k).
2. Randomly initialize k cluster centers.
3. Assign each data point to the nearest cluster center.
4. Update cluster centers as the mean of assigned points.
5. Repeat steps 3 and 4 until cluster centers stabilize.

2.3.1 Dataset

The dataset contains two-dimensional data named x and y coordinates.

	A	B	C
1	x	y	
2	2	5	
3	4	8	
4	6	6	
5	4	7	
6	5	4	
7	4	9	
8	6	10	
9	7	3	
10	3	5	
11	12	4	
12			

Figure 11: Snapshot of K-means Datasets

2.3.2 Implementation

i. Load Dataset

We loaded two-dimensional data from the kmeans_clustering.csv dataset, which contains x and y coordinates using **csv_reader**. After plotting the points, it looks as follows,

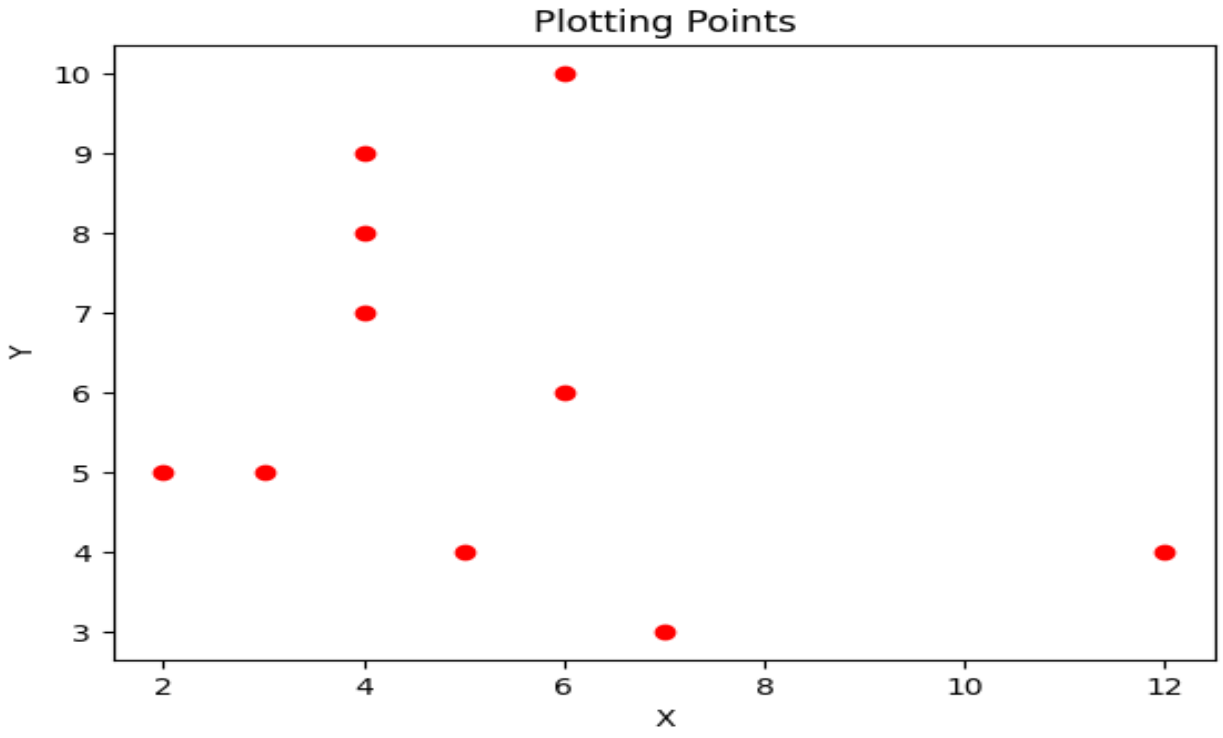


Figure 12: Plotting x and y Coordinates

ii. Euclidean Distance

Euclidean distance is the straight-line distance between two points in a space, commonly used to measure similarity or dissimilarity.

iii. Updating Cluster and Centroids

Initially, the centroids are chosen randomly, and after every iteration, the centroids change according to Euclidean distance, updating the cluster.

2.3.3 Performance Evaluation

Visualization: The final cluster are shown in the below pictures, considering the number of clusters for k is equal to 3.

Problem Analysis: Outlier

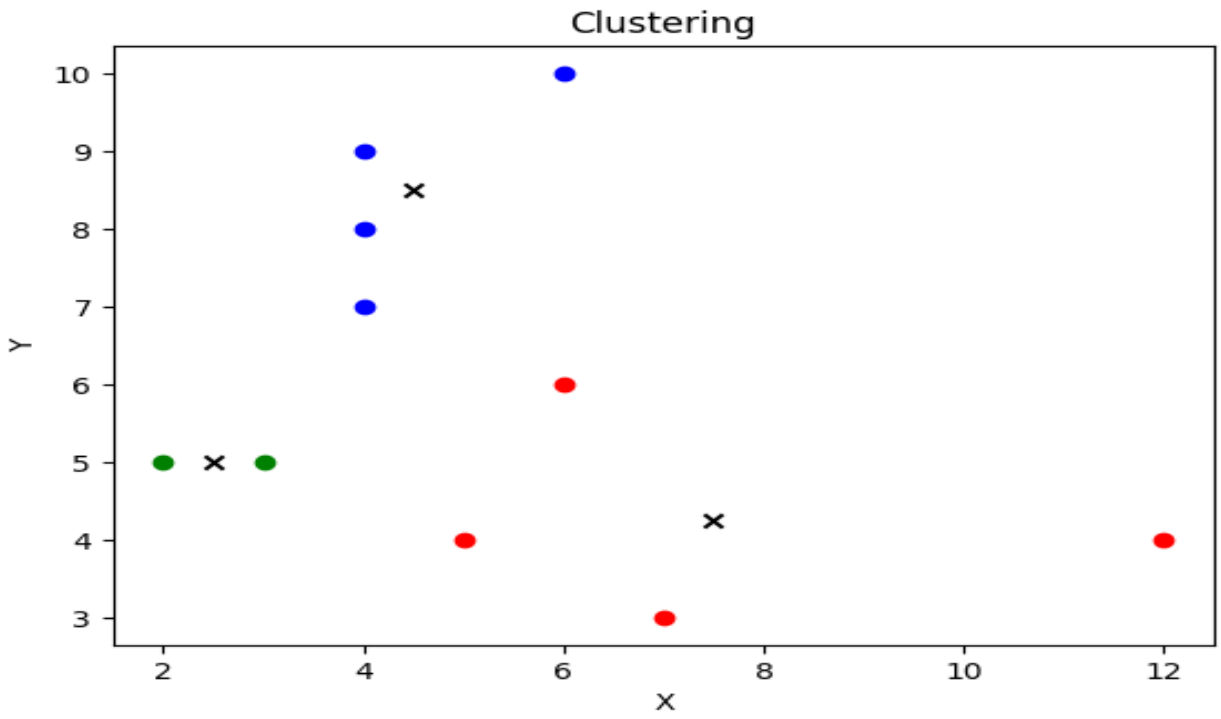


Figure 13: Final Cluster for k=3

2.4 Decision Tree

A decision tree is a graphical representation of a decision-making process that divides options into a number of nodes, branches, and leaves, ultimately leading to outcomes or decisions based on a number of input criteria and rules. It is a commonly used algorithm in machine learning and data analysis for classification and regression tasks, where it

recursively splits data into subsets based on the most important features, seeking to produce a predictive model that identifies patterns in the data.

2.4.1 Dataset

The dataset contains,

Data Features: Day, Temperature, Outlook, Humidity, Rainfall

Target Features: Game played or not

Challenges: Rainfall is numerical

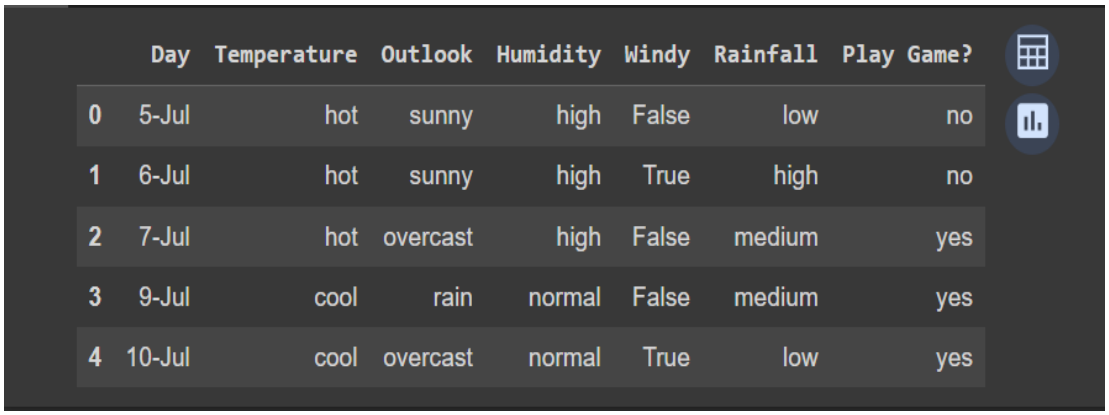
	A	B	C	D	E	F	G	H
1	Day	Temperatu	Outlook	Humidity	Windy	Rainfall	Play Game?	
2	5-Jul	hot	sunny	high	FALSE	120	no	
3	6-Jul	hot	sunny	high	TRUE	200	no	
4	7-Jul	hot	overcast	high	FALSE	180	yes	
5	9-Jul	cool	rain	normal	FALSE	150	yes	
6	10-Jul	cool	overcast	normal	TRUE	140	yes	
7	12-Jul	mild	sunny	high	FALSE	110	no	
8	14-Jul	cool	sunny	normal	FALSE	100	yes	
9	15-Jul	mild	rain	normal	FALSE	170	yes	
10	20-Jul	mild	sunny	normal	TRUE	140	yes	
11	21-Jul	mild	overcast	high	TRUE	130	yes	
12	22-Jul	hot	overcast	normal	FALSE	120	yes	
13	23-Jul	mild	rain	high	TRUE	160	no	
14	26-Jul	cool	rain	normal	TRUE	180	no	
15	30-Jul	mild	rain	high	FALSE	200	yes	

Figure 14: Data Features for Decision Tree

2.4.2 Implementation

Creating a decision tree using entropy and information gain involves arranging features as nodes, their values as branches, and predicted results as leaves. The algorithm picks the feature that best clarifies the result's uncertainty (entropy), using information gain. This helps choose informative features at the start. The process repeats, building a tree model that predicts results, until a stopping point, like a certain depth or minimum data per leaf, is reached.

i. **Challenges:** To convert one numerical feature (Rainfall) from numerical value to categorical label. After converting the features, it will be as follows



	Day	Temperature	Outlook	Humidity	Windy	Rainfall	Play Game?
0	5-Jul	hot	sunny	high	False	low	no
1	6-Jul	hot	sunny	high	True	high	no
2	7-Jul	hot	overcast	high	False	medium	yes
3	9-Jul	cool	rain	normal	False	medium	yes
4	10-Jul	cool	overcast	normal	True	low	yes

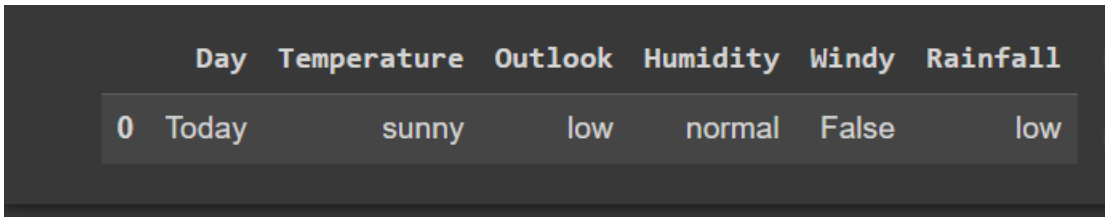
Figure 15: After Converting Numerical Value to Categorical Value

2.4.3 Performance Evaluation

Interpretability: Decision trees offer transparent decision paths, making it easier to understand how choices are made. Each step in the tree corresponds to a specific question or condition, making it simple to follow the logic behind a decision.

Insights: Through the decision tree's construction, important factors that heavily influence predictions, like "Humidity" in this case, become evident. The tree reveals which features contribute significantly to outcomes, giving valuable insights into the prediction process.

Test Data:



	Day	Temperature	Outlook	Humidity	Windy	Rainfall
0	Today	sunny	low	normal	False	low

Figure 16: Testing Data for Decision Tree

Decision: Played game is “yes”

2.5 Artificial Neural Network (ANN)

A computational model called an Artificial Neural Network (ANN) is modeled after the neural network of the human brain. An ANN analyses input and learns from it by utilizing interconnected nodes (neurons). Layers of neurons process input data, assigning weights to connections and generating outputs at each stage. The network modifies these weights throughout training to reduce the discrepancy between expected and actual results. ANNs

thrive in difficult tasks like language processing, prediction, and picture identification. Their hierarchical architecture enables feature extraction and pattern identification, allowing them to address a range of machine learning and artificial intelligence (AI) applications.

2.5.1 Dataset

The dataset used for this implementation is synthetic data generated by artificial intelligence.

Number of samples: 5000

Number of features: 10 (user control)

Number of classes: 3 (user control)

2.5.2 Implementation

The Artificial Neural Network works as follows,

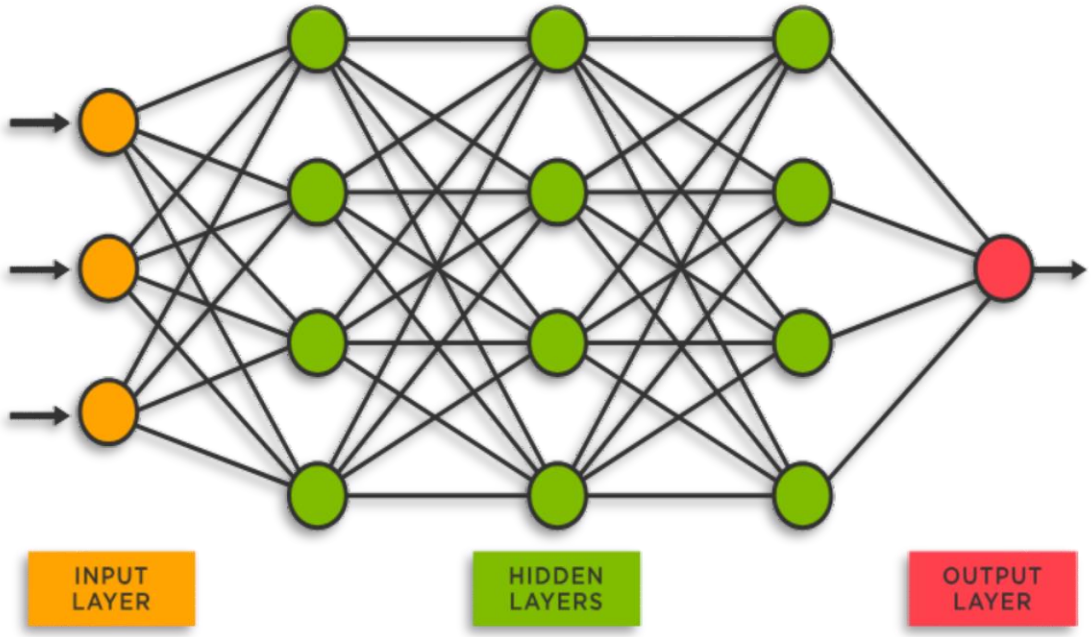


Figure 17: Working Mechanism of Artificial Neural Network

1. **Neurons and Layers**: ANNs consist of layers of interconnected nodes (neurons). The layers are categorized as input, hidden, and output layers. Data flows from the input layer through the hidden layers to the output layer.

2. **Activation Function**: Each neuron computes a weighted sum of its inputs and applies an activation function. This function introduces non-linearity, allowing the network to capture complex patterns.
3. **Forward Propagation**: During inference, data passes through the network in a forward direction. Neurons in each layer compute their outputs based on inputs, weights, and activation functions.
4. **Backpropagation**: Training involves two main steps: forward propagation to get predictions and backpropagation to adjust weights. Backpropagation calculates the gradient of the loss function with respect to weights and updates them using optimization techniques like gradient descent.

2.5.3 Performance Evaluation

Here is the confusion matrix for this classification,

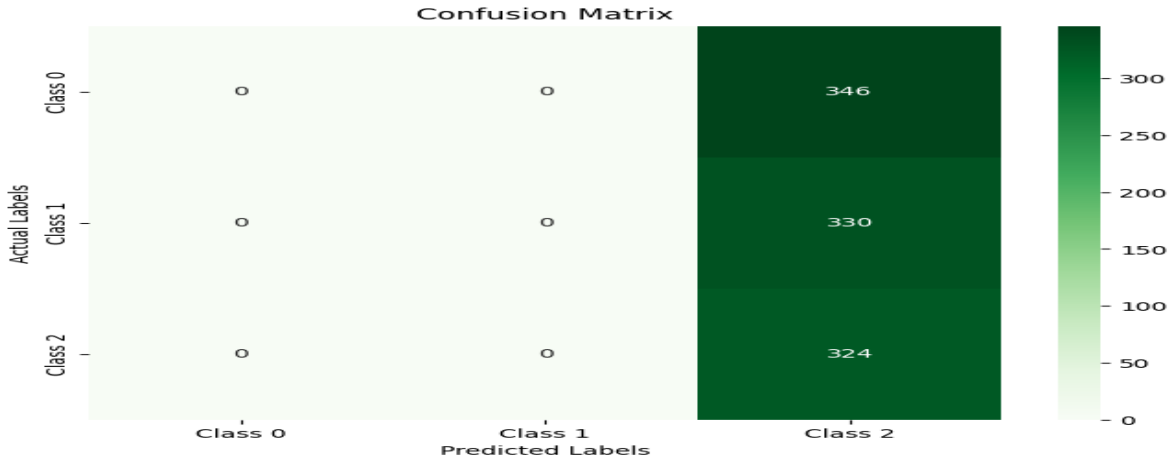


Figure 18: Confusion Matrix for Classification

Classification Report:

```
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.00      0.00      0.00       346
     Class 1       0.00      0.00      0.00       330
     Class 2       0.32      1.00      0.49       324

    accuracy                           0.32      1000
   macro avg       0.11      0.33      0.16      1000
weighted avg       0.10      0.32      0.16      1000
```

Figure 19: Classification Report

Analysis: As the dataset was created by synthetical way, the accuracy was lower than the expectation.

3 Discussion

We gain a thorough understanding of the capabilities and constraints of the five implemented machine learning algorithms by contrasting and analyzing the advantages, disadvantages, and real-world applications of each.

i. Apriori Algorithm:

Effective frequent item recognition, support for association rules in market analysis, and applicability in recommendation systems are all strengths. High-dimensional data problems and the possibility to produce a large number of sophisticated rules are weaknesses. Applications include cross-selling and upselling in retail as well as healthcare for co-occurring medical issues.

ii. Multivariable Linear Regression

Its benefits for predictive modeling come from its simple interpretation of linear correlations. Its assumption of linearity, sensitivity to outliers, and failure to detect complex nonlinear patterns are all flaws. Applications include estimating risk, predicting the economy, and estimating real estate prices.

iii. K-means Clustering

Efficiency with huge datasets, pattern finding, and simplicity of implementation are strengths. The sensitivity to starting positions and difficulties with non-spherical clusters are weaknesses. Applications include anomaly detection, image compression, and consumer segmentation.

iv. Decision Tree:

Features selection, processing different data kinds, and readability are strengths. Overfitting risk, the propensity to overlook complicated patterns, and bias are all examples of weaknesses. Applications include credit risk evaluation, medical diagnostics, and species classification.

v. Artificial Neural Network:

The processing of complex patterns is a strength of artificial neural networks (ANNs), which makes them ideal for jobs like voice and image recognition. Their shortcomings, however, include data aversion, complex design and training issues, and the possibility of overfitting. ANNs are used in image recognition, natural language processing, and driverless cars.

Common Difficulties: Common implementation difficulties were choosing the best parameters, dealing with missing data, and handling non-categorical data. Strategies like normalization, hyperparameter adjustment, and data pretreatment were used to address these problems.

Limitations and Improvement: Enhancing Apriori's scalability for bigger datasets could be investigated. In order to control multicollinearity and overfitting, regularization approaches may be beneficial for multivariable linear regression. Alternative distance measures could be taken into account to improve K-means clustering. The application of ensemble methods like Random Forests could improve Decision Trees. For better performance, ANNs could also benefit from new architectural developments and more effective optimization techniques.

Real World Impact: These algorithms have a lot of use in the real world. Through the use of personalized recommendations, the Apriori algorithm improves user experiences. Planning for policies and making educated business decisions are supported by multivariable linear regression. Market segmentation are identified using K-means clustering. Decision trees aid in strategic decision-making and aid in medical diagnostics. Furthermore, the advanced voice and image recognition capabilities attained by ANNs spark innovation by transforming several sectors.

4 Conclusion

The Apriori algorithm, Multivariable Linear Regression, K-Means Clustering, Decision Tree, and Artificial Neural Network are the five core machine learning methods that are covered in-depth in this report. We've gathered essential insights into each algorithm's unique advantages and disadvantages by carefully examining its theoretical underpinnings, practical applications, evaluation methods, and actual uses. The efficient application and analysis of these algorithms, which demonstrates students' remarkable command of machine learning concepts, is a significant accomplishment of this work.

The value of practical experience in enhancing knowledge and problem-solving skills in machine learning is crucial. Students have developed a profound understanding of model selection, parameter adjustment, and overcoming practical obstacles through active participation with the algorithms.

This report emphasizes the beneficial effects of hands-on learning, emphasizing how it improves students' understanding, critical thinking, and ability to apply machine learning algorithms to challenging problems. The exploratory and practical method used in this study establishes the groundwork for upcoming developments and discoveries in the dynamic field of machine learning.

5 References

i. Book

- Han, J., Kamber, M., & Pei, J. (2011). Data Mining: Concepts and Techniques (3rd Edition). Morgan Kaufmann.
- Data Mining: Concepts and Techniques, 3rd Edition. Jiawei Han, Micheline Kamber, Jian Pei. Database Modeling and Design: Logical Design, 5th Edition.
- Lecture Slides

ii. Online Resources

- https://www.coursera.org/learn/machine-learning
- https://www.geeksforgeeks.org/machine-learning/
- https://en.wikipedia.org/wiki/Machine_learning
- https://www.ibm.com/topics/machine-learning
- https://www.youtube.com/

iii. Dataset Link

- https://drive.google.com/drive/folders/1UA0jfmjgkbxKcxKgUnkUdbeDf3ItTG9_?usp=drive_link

6 Appendices

i. Apriori Algorithm:

+ Code + Text

+ Code +

[1] 0s

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
import pandas as pd
```

0s

```
def load_dataset():
    df = pd.read_csv('store_data.csv')

    df = df.drop_duplicates()

    dataset = []
    for _, row in df.iterrows():
        transaction = set(str(item) for item in row.values)
        dataset.append(transaction)

    return dataset
```

[3] 8s

```

dataset = load_dataset()

te = TransactionEncoder()
te_ary = te.fit_transform(dataset)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

min_support = int(input("Enter the minimum support: "))
min_support/=100

# Run Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(df_encoded, min_support=min_support, use_colnames=True)

if frequent_itemsets.empty:
    print("No frequent itemsets found for the given minimum support.")
    exit()

min_confidence = int(input("Enter the minimum confidence: "))
min_confidence/=100

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)

```

```python
# print(rules)

if rules.empty:
    print("No association rules found for the given minimum confidence.")
    exit()

# Print all association rules
print("All Association Rules:")
for _, rule in rules.iterrows():
    antecedents = ', '.join(rule['antecedents'])
    consequents = ', '.join(rule['consequents'])
    support = rule['support']
    confidence = rule['confidence']
    lift = rule['lift']

    print("Rule: {} -> {}".format(antecedents, consequents))
    print("Support: {:.4f}".format(support))
    print("Confidence: {:.4f}".format(confidence))
    print("Lift: {:.4f}".format(lift))
    print()

```

```python

num_rules = int(input("Enter the number of association rules to display: "))

sorted_rules = rules.sort_values(by="confidence", ascending=False)

top_rules = sorted_rules.head(num_rules)

# Print top N association rules
print("Top {} Association Rules:".format(num_rules))
for _, rule in top_rules.iterrows():
    antecedents = ', '.join(rule['antecedents'])
    consequents = ', '.join(rule['consequents'])
    support = rule['support']
    confidence = rule['confidence']
    lift = rule['lift']

    print("Rule: {} -> {}".format(antecedents, consequents))
    print("Support: {:.4f}".format(support))
    print("Confidence: {:.4f}".format(confidence))
    print("Lift: {:.4f}".format(lift))
    print()


```

ii. Multivariable Linear Regression:

```
+ Code + Text

[ ]
1 import numpy as np
2 import pandas as pd
3 import re
4 from sklearn.model_selection import train_test_split
5 from sklearn.preprocessing import MinMaxScaler, StandardScaler
6 from sklearn.metrics import mean_squared_error
7

[4]
1 data = pd.read_csv('/content/property_listing_data_in_Bangladesh.csv')
2 data = data.drop(columns=['title', 'adress', 'type', 'purpose', 'flooPlan', 'url', 'lastUpdated'])
3 data.head()
```

```
[5]
1 data.shape

(7557, 4)
```

```
1 beds_values = []
2 for value in data['beds']:
3     match = re.search(r'\d+', str(value))
4     if match:
5         beds_values.append(int(match.group()))
6     else:
7         beds_values.append(0)
8 data['beds'] = beds_values
9 data['beds']
```

```
1 bath_values = []
2 for value in data['bath']:
3     match = re.search(r'\d+', str(value))
4     if match:
5         bath_values.append(int(match.group()))
6     else:
7         bath_values.append(0)
8 data['bath'] = bath_values
9
10 data['bath']
```

```
area_values = []
for value in data['area']:
    if isinstance(value, str):
        value = value.replace(',', '')
        value = value.split(' ')[0]
        if value.replace('.', '', 1).isdigit():
            area_values.append(float(value))
        else:
            area_values.append(0.0)
    else:
        area_values.append(0.0)
data['area'] = area_values


data['area']
```

```
price_values = []
for value in data['price']:
    if isinstance(value, str):
        value = value.replace(',', '')
        value = value.replace(' Thousand', '000')
        value = value.replace(' Lakh', '00000')
        if value.replace('.', '', 1).isdigit():
            price_values.append(float(value))
        else:
            price_values.append(0.0)
    else:
        price_values.append(0.0)
data['price'] = price_values
```

```
data.head()
```

```
[11]
scaler = MinMaxScaler()
# scaler = StandardScaler()
numeric_columns = ['beds', 'bath', 'area', 'price']
data[numeric_columns] = scaler.fit_transform(data[numeric_columns])

data.head()
```

```python
def compute_cost(X, y, weights):
    m = len(y)
    predictions = np.dot(X, weights)
    squared_error = np.square(predictions - y)
    cost = np.sum(squared_error) / (2 * m)

    # print(cost)
    return cost
```

```python
def gradient_descent(X, y, weights, learning_rate, num_iterations):
    m = len(y)
    costs = []

    for i in range(num_iterations):
        predictions = np.dot(X, weights)
        error = predictions - y
        gradient = np.dot(X.T, error) / m
        weights = weights - learning_rate * gradient
        cost = compute_cost(X, y, weights)
        costs.append(cost)
    # print(weights,cost)
    return weights, costs

```

```python
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
train_data, val_data = train_test_split(train_data, test_size=0.25, random_state=42)

X_train = train_data.drop('price', axis=1).values
y_train = train_data['price'].values
X_val = val_data.drop('price', axis=1).values
y_val = val_data['price'].values
X_test = test_data.drop('price', axis=1).values
y_test = test_data['price'].values

```

```python
X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
X_val = np.hstack((np.ones((X_val.shape[0], 1)), X_val))
X_test = np.hstack((np.ones((X_test.shape[0], 1)), X_test))

weights = np.zeros(X_train.shape[1])

```

```
[16]
learning_rate = 0.01
num_iterations = 100

weights, costs = gradient_descent(X_train, y_train, weights, learning_rate, num_iterations)
```

```
for epoch in range(num_iterations):
    train_cost = compute_cost(X_train, y_train, weights)
    val_cost = compute_cost(X_val, y_val, weights)
    print(f"Epoch {epoch+1}: Training Loss = {train_cost}, Validation Loss = {val_cost}")
```

```
test_cost = compute_cost(X_test, y_test, weights)
print("Test Loss:", test_cost)
```

```
Test Loss: 0.0005213189566025826
```

```
[19]
y_pred = np.dot(X_test, weights)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Mean Squared Error (MSE): 0.0010426379132051653
Root Mean Squared Error (RMSE): 0.032289904199380416
```

```
y_pred = np.dot(X_test, weights)

mse = np.mean((y_pred - y_test) ** 2)

rmse = np.sqrt(mse)

print("Mean Squared Error (MSE) on Test Set:", mse)
print("Root Mean Squared Error (RMSE) on Test Set:", rmse)
```

```
Mean Squared Error (MSE) on Test Set: 0.0010426379132051653
Root Mean Squared Error (RMSE) on Test Set: 0.032289904199380416
```

iii. K-means Clustering:

+ Code + Text

[1] 1s

```
import csv
import random
import math
import matplotlib.pyplot as plt

```

[2] 0s

```
def load_dataset(filename):
    dataset = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            try:
                point = [float(row[0]), float(row[1])]
                dataset.append(point)
            except ValueError:
                continue
    return dataset

```

[3] 0s

```
def euclidean_distance(point1, point2):
    return math.sqrt(sum((a - b) ** 2 for a, b in zip(point1, point2)))

```

[4] 0s

```
def initialize_centroids(dataset, k):
    return random.sample(dataset, k)
```

[5] 0s

```
def assign_to_cluster(dataset, centroids):
    clusters = [[] for _ in range(len(centroids))]
    for point in dataset:
        distances = [euclidean_distance(point, centroid) for centroid in centroids]
        cluster_index = distances.index(min(distances))
        clusters[cluster_index].append(point)
    return clusters


def update_centroids(clusters):
    return [[sum(coord) / len(cluster) for coord in zip(*cluster)] for cluster in clusters]

```

```python
def kmeans(dataset, k):
    centroids = initialize_centroids(dataset, k)
    while True:
        clusters = assign_to_cluster(dataset, centroids)
        new_centroids = update_centroids(clusters)
        if centroids == new_centroids:
            break
        centroids = new_centroids
        plot_clusters(clusters, centroids)
    return clusters, centroids
```

[7]

```python
def plot_clusters(clusters, centroids):
    colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k']
    for i, cluster in enumerate(clusters):
        color = colors[i % len(colors)]
        x, y = zip(*cluster)
        plt.scatter(x, y, c=color)
    x_centroids, y_centroids = zip(*centroids)
    plt.scatter(x_centroids, y_centroids, marker='x', c='k')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Clustering')
    plt.show()
```

6s

```python
filename = '/content/kmean_clustering.csv'

dataset = load_dataset(filename)


x_coords, y_coords = zip(*dataset)
plt.scatter(x_coords, y_coords, color='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Plotting Points')
plt.grid(False)
plt.show()

k = int(input("Number of clusters k: "))
clusters, centroids = kmeans(dataset, k)
plot_clusters(clusters, centroids)

```

iv. Decision Tree:

+ Code + Text

```
1  import pandas as pd
2  import math
3
```

```
[49]  1  df = pd.read_csv('dt_dataset.csv')
      2  df.head()
```

```
[50]  1  def convert_rainfall_to_category(value):
      2      if 100 <= value <= 140:
      3          return 'low'
      4      elif 141 <= value <= 180:
      5          return 'medium'
      6      else:
      7          return 'high'
      8
      9  df['Rainfall'] = df['Rainfall'].apply(convert_rainfall_to_category)
     10  df.head()
```

```
[51]  1  def calculate_entropy(data):
      2      target_labels = data['Play Game?']
      3      total_instances = len(target_labels)
      4      unique_labels = target_labels.unique()
      5      entropy_val = 0
      6
      7      for label in unique_labels:
      8          probability = len(target_labels[target_labels == label]) / total_instances
      9          entropy_val -= probability * math.log2(probability)
     10
     11      return entropy_val
     12
```

```
[52]  1  def calculate_information_gain(data, attribute):
      2      total_instances = len(data)
      3      attribute_entropy = 0
      4
      5      for value in data[attribute].unique():
      6          subset = data[data[attribute] == value]
      7          subset_entropy = calculate_entropy(subset) * len(subset) / total_instances
      8          attribute_entropy += subset_entropy
      9
     10      return calculate_entropy(data) - attribute_entropy
     11
```

```python
def build_decision_tree(data, candidate_attributes):

    if len(data['Play Game?'].unique()) == 1:
        return data['Play Game?'].iloc[0]


    if len(candidate_attributes) == 0:
        return data['Play Game?'].value_counts().idxmax()

    max_information_gain = -1
    best_attribute = None
    for attribute in candidate_attributes:
        information_gain = calculate_information_gain(data, attribute)
        if information_gain > max_information_gain:
            max_information_gain = information_gain
            best_attribute = attribute

    tree = {best_attribute: {}}
    remaining_attributes = [attr for attr in candidate_attributes if attr != best_attribute]

    for value in data[best_attribute].unique():
        subset = data[data[best_attribute] == value]
        subtree = build_decision_tree(subset, remaining_attributes)
        tree[best_attribute][value] = subtree

    return tree

# List of candidate attributes (excluding the target variable 'Play Game?')
candidate_attributes = ['Temperature', 'Outlook', 'Humidity', 'Windy', 'Rainfall']
```

```
[54]
decision_tree = build_decision_tree(df, candidate_attributes)

print(decision_tree)
```

```
{'Outlook': {'sunny': {'Humidity': {'high': 'no', 'normal': 'yes'}}, 'overcast': 'yes', 'rain': {'Windy': {False: 'yes', True: 'no'}}}}
```

```
[55]
train_data = pd.read_excel('/content/test_data.xlsx')
testing_data = pd.DataFrame(train_data)
testing_data['Rainfall'] = testing_data['Rainfall'].apply(convert_rainfall_to_category)
testing_data.head()
```

```
[56]
def predict_instance(instance, tree):
    attribute = next(iter(tree))
    value = instance[attribute]

    if value in tree[attribute]:
        subtree = tree[attribute][value]
        if isinstance(subtree, dict):
            return predict_instance(instance, subtree)
        else:
            return subtree
    else:
        # If the value is not present in the tree, return the majority class of the subtree
        return list(tree[attribute].values())[0]

predictions = testing_data.apply(lambda x: predict_instance(x, decision_tree), axis=1)

testing_data['Play Game?'] = predictions
print(testing_data[['Play Game?']])
```

v. Artificial Neural Network:

+ Code + Text

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))
```

```
def initialize_parameters(input_size, hidden_size, output_size):
    np.random.seed(42)  # For reproducibility
    hidden_weights = np.random.randn(input_size, hidden_size) * 0.01
    hidden_biases = np.zeros((1, hidden_size))
    output_weights = np.random.randn(hidden_size, output_size) * 0.01
    output_biases = np.zeros((1, output_size))
    return hidden_weights, hidden_biases, output_weights, output_biases
```

```
def forward_propagation(X, hidden_weights, hidden_biases, output_weights, output_biases):
    hidden_layer_output = sigmoid(np.dot(X, hidden_weights) + hidden_biases)
    output_layer_output = sigmoid(np.dot(hidden_layer_output, output_weights) + output_biases)
    return hidden_layer_output, output_layer_output
```

```
def backpropagation(X, y, hidden_layer_output, output_layer_output,
                    hidden_weights, hidden_biases, output_weights, output_biases, learning_rate):
    num_samples = X.shape[0]

    # Compute the gradients for output layer
    output_error = output_layer_output - y.reshape(-1, 1)
    output_delta = output_error * sigmoid_derivative(output_layer_output)
    output_weights_gradient = np.dot(hidden_layer_output.T, output_delta)
    output_biases_gradient = np.sum(output_delta, axis=0, keepdims=True)

    # Compute the gradients for hidden layer
    hidden_error = np.dot(output_delta, output_weights.T)
    hidden_delta = hidden_error * sigmoid_derivative(hidden_layer_output)
    hidden_weights_gradient = np.dot(X.T, hidden_delta)
    hidden_biases_gradient = np.sum(hidden_delta, axis=0, keepdims=True)
```

```
[12]
    # Update the weights and biases
    output_weights -= learning_rate * output_weights_gradient / num_samples
    output_biases -= learning_rate * output_biases_gradient / num_samples
    hidden_weights -= learning_rate * hidden_weights_gradient / num_samples
    hidden_biases -= learning_rate * hidden_biases_gradient / num_samples

    return hidden_weights, hidden_biases, output_weights, output_biases
```

```
[13]
def train(X, y, hidden_units, learning_rate, num_epochs):
    num_samples, num_features = X.shape
    num_classes = np.max(y) + 1

    hidden_weights, hidden_biases, output_weights, output_biases = initialize_parameters(num_features, hidden_units, num_classes)

    for epoch in range(num_epochs):
        # Forward propagation
        hidden_layer_output, output_layer_output = forward_propagation(X, hidden_weights, hidden_biases, output_weights, output_biases)

        # Backpropagation
        hidden_weights, hidden_biases, output_weights, output_biases = backpropagation(X, y, hidden_layer_output, output_layer_output,
                                                                                        hidden_weights, hidden_biases, output_weights, output_biases,
                                                                                        learning_rate)

    return hidden_weights, hidden_biases, output_weights, output_biases
```

```
[14]
def predict(X, hidden_weights, hidden_biases, output_weights, output_biases):
    _, output_layer_output = forward_propagation(X, hidden_weights, hidden_biases, output_weights, output_biases)
    predictions = np.argmax(output_layer_output, axis=1)
    return predictions
```

```
[15]
def calculate_accuracy(predictions, actual_labels):
    correct_predictions = np.sum(predictions == actual_labels)
    total_samples = len(actual_labels)
    accuracy = correct_predictions / total_samples
    return accuracy
```

```
# Generate a synthetic dataset
num_samples = 5000
num_features = int(input("Enter the number of input features: "))
num_classes = int(input("Enter the number of output classes: "))

X = np.random.randn(num_samples, num_features)
y = np.random.randint(num_classes, size=num_samples)

# Split the dataset into training and testing sets
split_ratio = 0.8
split_index = int(split_ratio * num_samples)
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]

# Train the neural network
learning_rate = float(input("Learning rate: "))
num_epochs = int(input("Total number of epochs: "))
hidden_units = int(input("Enter the no. of hidden units: "))

hidden_weights, hidden_biases, output_weights, output_biases = train(X_train, y_train, hidden_units, learning_rate, num_epochs)
# Make predictions on the test set
predictions = predict(X_test, hidden_weights, hidden_biases, output_weights, output_biases)
# Calculate accuracy
accuracy = calculate_accuracy(predictions, y_test)
print("Accuracy:", accuracy*100)
```

```
conf_matrix = confusion_matrix(y_test,predictions)
target_names = [f'Class {i}' for i in range(num_classes)]

plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d',cmap='Greens', xticklabels=target_names, yticklabels=target_names)
plt.xlabel('Predicted Labels')
plt.ylabel("Actual Labels")
plt.title("Confusion Matrix")
plt.show()

report = classification_report(y_test,predictions, target_names= target_names)
print('Classification Report: ')
print(report)
```