## Experiment-1

1. **Implement and demonstratethe FIND-Salgorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.**

### ****Experiment Overview****

### The FIND-S algorithm identifies the most specific hypothesis consistent with a given set of positive training examples. It updates the hypothesis based on positive instances and ignores negative examples.

### ****Advantages and Disadvantages****

#### ****Advantages****:

* **Simple**: The algorithm is easy to understand and implement.
* **Efficiency**: It efficiently generates the most specific hypothesis from a clean dataset.

#### ****Disadvantages****:

* **Limited to Binary Classification**: It only works for binary target attributes (e.g., "Yes"/"No").
* **Noise Sensitivity**: The algorithm assumes that the data is noise-free and may not perform well with inconsistent or contradictory data.
* **Ignores Negative Examples**: It does not consider negative examples, which may result in an incomplete hypothesis.

### ****Salient Features****

* **Specificity**: The algorithm constructs a hypothesis that is the most specific.
* **Updates Hypothesis**: It progressively refines the hypothesis based on each positive example.
* **Efficiency**: It focuses on positive examples, ignoring the complexity of negative examples.
* **Simple Structure**: It uses simple comparison and substitution of attributes with "?" or "Ø".

### ****Real-World Applications****

* **Medical Diagnosis**: It can be used to classify medical conditions based on specific symptoms.
* **Pattern Recognition**: Helps in identifying specific patterns in clean datasets.
* **Fault Detection**: In systems where only certain conditions need to be met for an anomaly or fault detection.

### ****Algorithm****

1. **Initialize Hypothesis**: Start with a hypothesis containing all "Ø" values for the attributes.
2. **Process Positive Examples**: For each positive training example:
   * If the attribute in the hypothesis is "Ø", replace it with the attribute from the example.
   * If the attribute differs, replace it with "?".
3. **Output the Final Hypothesis**: The hypothesis is updated until all positive examples are processed.

### Sample DataSet

| **Color** | **Shape** | **Size** | **Texture** | **Taste** | **Is Inside White** | **Has Seeds** | **Sugar** | **Calories** | **Edible** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Red | Round | Small | Smooth | Sweet | No | Yes | Low | Medium | Yes |
| Yellow | Round | Medium | Smooth | Sweet | No | Yes | Low | Medium | Yes |
| Red | Oval | Medium | Smooth | Sweet | No | Yes | Low | Medium | No |
| Yellow | Oval | Medium | Smooth | Sweet | No | Yes | Low | Medium | Yes |
| Red | Round | Medium | Rough | Sweet | No | Yes | Low | Medium | No |
| Yellow | Round | Medium | Smooth | Sweet | No | Yes | Low | Medium | Yes |
| Green | Oval | Medium | Smooth | Sweet | No | Yes | Low | Medium | Yes |
| Red | Round | Medium | Smooth | Sweet | No | Yes | Low | Medium | Yes |

### DataSet Link

https://github.com/bhanupattemz/ML/blob/main/Experiment1/fruit.csv

### ****Algorithm Implementation****

#### ****With Libraries (Using Pandas)****

import pandas as pd

def find\_s\_algorithm(filename):

    data = pd.read\_csv(filename)

    features = data.iloc[:, :-1].values

    target = data.iloc[:, -1].values

    hypothesis = ["Ø"] \* features.shape[1]

    for i, example in enumerate(features):

        if target[i] == "Yes":

            for j in range(len(hypothesis)):

                if hypothesis[j] == "Ø":

                    hypothesis[j] = example[j]

                elif hypothesis[j] != example[j]:

                    hypothesis[j] = "?"

    return hypothesis

print(find\_s\_algorithm(r"c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment1\fruit.csv"))

**Output**

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment1\with.py"

['?', '?', '?', '?', 'Sweet', 'No', 'Yes', 'Low', 'Medium']

#### ****Without Libraries (Without Pandas)****

def find\_s\_algorithm\_without\_libraries(filename):

    with open(filename, 'r') as file:

        lines = file.readlines()

    data = [line.strip().split(',') for line in lines]

    features = [line[:-1] for line in data]

    target = [line[-1] for line in data]

    hypothesis = ["Ø"] \* len(features[0])

    for i, example in enumerate(features):

        if target[i] == "Yes":

            for j in range(len(hypothesis)):

                if hypothesis[j] == "Ø":

                    hypothesis[j] = example[j]

                elif hypothesis[j] != example[j]:

                    hypothesis[j] = "?"

    return hypothesis

hypothesis = find\_s\_algorithm\_without\_libraries(r"c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment1\fruit.csv")

print("Final Hypothesis:", hypothesis)

### ****Output****

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment1\without.py"

Final Hypothesis: ['?', '?', '?', '?', 'Sweet', 'No', 'Yes', 'Low', 'Medium']

[Done] exited with code=0 in 0.073 seconds

## Experiment-2

**For a given set of training data examples stored in a .CSV file, implement and demonstrate the**

**Candidate-Elimination algorithmto output a description of the set of all hypotheses consistent with the training examples.**

#### ****Experiment Overview****

The Candidate-Elimination algorithm is a concept-learning approach used in Machine Learning to generate a hypothesis that fits a given set of training data. It maintains two boundaries — the most general hypothesis (G) and the most specific hypothesis (S). By processing positive and negative examples, it refines these boundaries iteratively to find a consistent hypothesis that classifies the data correctly.

#### ****Advantages****

· Considers both positive and negative examples for learning.

· Provides a set of consistent hypotheses rather than a single one.

· Tracks the learning process and updates boundaries accordingly.

#### ****Disadvantages****

· It is computationally intensive for large datasets.

· Sensitive to noisy or inconsistent data.

· May struggle with incomplete or ambiguous data.

#### ****Salient Features****

· Maintains and updates general (G) and specific (S) boundaries.

· Refines hypotheses based on each training example.

· Helps to identify a range of possible hypotheses instead of one fixed result.

#### ****Real-World Applications****

· **Medical Diagnosis**: Predicting diseases based on patient symptoms.

· **Spam Filtering**: Classifying emails as spam or not based on content.

· **Pattern Recognition**: Identifying objects or text from images.

· **Fault Detection**: Detecting failures in machinery based on sensor data.

#### ****Algorithm****

**Initialize Hypotheses:**

* Set the **specific hypothesis (S)** to the most restrictive (specific) hypothesis.
* Set the **general hypothesis (G)** to the most general hypothesis.

**Process Each Example:**

For a **positive example**:

* Generalize **S** only when needed to include the example.
* Remove any hypothesis from **G** that does not classify the example as positive.

For a **negative example**:

* Specialize **G** only when needed to exclude the example.
* Remove any hypothesis from **S** that classifies the example as positive.

**Update Boundaries:**

* Ensure that **S** remains the most specific hypothesis and **G** remains the most general.
* Eliminate any hypothesis from **G** that is inconsistent with the data.

**Terminate:**

* After processing all examples, output the final set of consistent hypotheses.

**Sample DataSet**

| **Hair** | **Feathers** | **Eggs** | **Milk** | **Fins** | **Legs** | **Habitat** | **Diet** | **Tail** | **Domesticated** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 1 | 0 | Water | Carnivore | Yes | No |
| 0 | 1 | 1 | 0 | 0 | 2 | Air | Carnivore | Yes | No |
| 0 | 0 | 1 | 0 | 0 | 4 | Land | Herbivore | No | No |
| 1 | 0 | 0 | 1 | 0 | 4 | Both | Carnivore | Yes | No |
| 1 | 0 | 0 | 1 | 0 | 2 | Both | Herbivore | No | Yes |
| 0 | 0 | 1 | 0 | 1 | 0 | Water | Carnivore | Yes | No |
| 1 | 0 | 0 | 1 | 0 | 4 | Land | Omnivore | Yes | Yes |
| 0 | 0 | 1 | 0 | 0 | 4 | Land | Carnivore | No | No |

#### ****Dataset Link:****

https://github.com/bhanupattemz/ML/blob/main/Experiment2/zoo.csv

#### ****Algorithm Implementation****

##### ****With Libraries (Using Pandas)****

import pandas as pd

def candidate\_elimination\_with\_libraries(filename):

    data = pd.read\_csv(filename)

    features = data.iloc[:, :-1].values

    target = data.iloc[:, -1].values

    general\_hypothesis = ['?' for \_ in range(features.shape[1])]

    specific\_hypothesis = features[0].copy()

    for i, example in enumerate(features):

        if target[i] == 'Yes':

            for j in range(len(general\_hypothesis)):

                if general\_hypothesis[j] == '?':

                    general\_hypothesis[j] = example[j]

                elif general\_hypothesis[j] != example[j]:

                    general\_hypothesis[j] = '?'

        else:

            for j in range(len(specific\_hypothesis)):

                if specific\_hypothesis[j] != example[j]:

                    specific\_hypothesis[j] = '?'

    return general\_hypothesis, specific\_hypothesis

general,specific=candidate\_elimination\_with\_libraries(r"c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment2\zoo.csv")

print("Most General Hypothesis:", general)

print("Most Specific Hypothesis:", specific)

##### ****Output****

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment2\with.py"

Most General Hypothesis: [1, 0, 0, 1, 0, '?', '?', 'Herbivore', '?']

Most Specific Hypothesis: ['?' '?' '?' '?' '?' '?' '?' '?' '?']

[Done] exited with code=0 in 2.515 seconds

##### ****Without Libraries (Without Pandas)****

def candidate\_elimination\_without\_libraries(filename):

    with open(filename, 'r') as file:

        lines = file.readlines()

    data = [line.strip().split(',') for line in lines]

    features = [line[:-1] for line in data]

    target = [line[-1] for line in data]

    general\_hypothesis = ['?' for \_ in range(len(features[0]))]

    specific\_hypothesis = features[0].copy()

    for i, example in enumerate(features):

        if target[i] == 'Yes':

            for j in range(len(general\_hypothesis)):

                if general\_hypothesis[j] == '?':

                    general\_hypothesis[j] = example[j]

                elif general\_hypothesis[j] != example[j]:

                    general\_hypothesis[j] = '?'

        else:

            for j in range(len(specific\_hypothesis)):

                if specific\_hypothesis[j] != example[j]:

                    specific\_hypothesis[j] = '?'

    return general\_hypothesis, specific\_hypothesis

general, specific = candidate\_elimination\_without\_libraries("zoo.csv")

print("Most General Hypothesis:", general)

print("Most Specific Hypothesis:", specific)

**Output**

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment2\without.py"

Most General Hypothesis: ['1', '0', '0', '1', '0', '?', '?', 'Herbivore', '?']

Most Specific Hypothesis: ['?', '?', '?', '?', '?', '?', '?', '?', '?']

[Done] exited with code=0 in 0.085 seconds

## Experiment -3

**Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an**

**appropriate data set for building the decision tree and apply this knowledge to classify a new**

**sample.**

#### ****Experiment Overview :****

**To implement and demonstrate the working of the ID3 algorithm, which builds a decision tree for classifying new samples based on a given training dataset.**

### ****Advantages****:

1. **Clear Visualization**: Decision trees are easy to visualize, making them interpretable.
2. **Handles Both Categorical and Continuous Data**: ID3 works well with both types of data, allowing it to be versatile.

### ****Disadvantages****:

1. **Prone to Overfitting**: Decision trees can overfit if the tree becomes too complex or deep.
2. **Bias towards Features with More Levels**: The algorithm might be biased towards features with more unique values.

### ****Silent Features****:

1. **Information Gain**: The algorithm chooses the feature with the highest information gain at each node.
2. **Recursive Splitting**: The dataset is recursively divided based on information gain until the stopping criterion is met.

### ****Real-world Applications****:

1. **Medical Diagnosis**: Decision trees are widely used in diagnosing diseases based on symptoms and test results.
2. **Customer Segmentation**: Companies use decision trees to classify customers based on their behaviors and characteristics.

### ****Algorithm****:

1. **Calculate Entropy**: Compute the entropy for each feature in the dataset to understand its impurity.
2. **Calculate Information Gain**: For each feature, calculate the information gain using entropy to determine which feature best splits the data.
3. **Recursive Splitting**: Select the feature with the highest information gain and split the dataset recursively.
4. **Stopping Criteria**: The recursion stops when all samples are classified or no features are left to split.

### ****Dataset****:

| **Action** | **RPG** | **Sports** | **Multiplayer** | **OpenWorld** | **Rating\_M** | **Success** |
| --- | --- | --- | --- | --- | --- | --- |
| Yes | No | No | Online | Open | Mature | AAA |
| No | Yes | No | Online | Open | Mature | AAA |
| Yes | No | No | Online | Limited | Teen | Mid |
| No | Yes | No | Online | Open | Mature | AAA |
| Yes | No | Yes | Offline | Limited | Teen | Indie |

**Dataset Link:**

<https://github.com/bhanupattemz/ML/blob/main/Experiment3/games.csv>

**Code With Library :**

from sklearn.tree import DecisionTreeClassifier, export\_text

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

import pandas as pd

filename = r"c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment3\games.csv"

data = pd.read\_csv(filename)

label\_encoder = LabelEncoder()

for column in data.columns:

    data[column] = label\_encoder.fit\_transform(data[column])

X = data.iloc[:, :-1]

y = data.iloc[:, -1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier(criterion='entropy', random\_state=42)

clf.fit(X\_train, y\_train)

tree\_rules = export\_text(clf, feature\_names=list(X.columns))

print("Decision Tree Rules:")

print(tree\_rules)

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100:.2f}%")

**Output :**

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment3\main.py"

Decision Tree Rules:

|--- Multiplayer <= 0.50

|   |--- Action <= 0.50

|   |   |--- class: 1

|   |--- Action >  0.50

|   |   |--- Rating\_M <= 0.50

|   |   |   |--- class: 1

|   |   |--- Rating\_M >  0.50

|   |   |   |--- class: 2

|--- Multiplayer >  0.50

|   |--- OpenWorld <= 0.50

|   |   |--- Sports <= 0.50

|   |   |   |--- class: 2

|   |   |--- Sports >  0.50

|   |   |   |--- Action <= 0.50

|   |   |   |   |--- class: 2

|   |   |   |--- Action >  0.50

|   |   |   |   |--- class: 2

|   |--- OpenWorld >  0.50

|   |   |--- Rating\_M <= 0.50

|   |   |   |--- class: 0

|   |   |--- Rating\_M >  0.50

|   |   |   |--- class: 2

Accuracy: 76.19%

[Done] exited with code=0 in 10.495 seconds

##### Code Without Libraries:

import csv

import math

def calculate\_entropy(data):

    value\_counts = {}

    for val in data:

        if val not in value\_counts:

            value\_counts[val] = 0

        value\_counts[val] += 1

    entropy = 0

    for count in value\_counts.values():

        prob = count / len(data)

        entropy -= prob \* math.log2(prob)

    return entropy

def split\_data(data, attribute):

    splits = {}

    for row in data:

        key = row[attribute]

        if key not in splits:

            splits[key] = []

        splits[key].append(row)

    return splits

def id3(data, attributes, target\_col):

    target\_values = [row[target\_col] for row in data]

    if len(set(target\_values)) == 1:

        return target\_values[0]

    if not attributes:

        return max(set(target\_values), key=target\_values.count)

    base\_entropy = calculate\_entropy(target\_values)

    best\_gain = -1

    best\_attr = None

    best\_splits = None

    for attr in attributes:

        splits = split\_data(data, attr)

        weighted\_entropy = 0

        for subset in splits.values():

            target\_subset = [row[target\_col] for row in subset]

            weighted\_entropy += (len(target\_subset) / len(data)) \* calculate\_entropy(target\_subset)

        info\_gain = base\_entropy - weighted\_entropy

        if info\_gain > best\_gain:

            best\_gain = info\_gain

            best\_attr = attr

            best\_splits = splits

    tree = {headers[best\_attr]: {}}

    for key, subset in best\_splits.items():

        subtree = id3(subset, [attr for attr in attributes if attr != best\_attr], target\_col)

        tree[headers[best\_attr]][key] = subtree

    return tree

filename = r"c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment3\games.csv"

data = []

with open(filename, newline='') as f:

    reader = csv.reader(f)

    headers = next(reader)

    for row in reader:

        if len(row) != len(headers):

            continue

        data.append(row)

attributes = list(range(len(headers) - 1))

target\_col = len(headers) - 1

decision\_tree = id3(data, attributes, target\_col)

print("Decision Tree:")

for key, value in decision\_tree.items():

    print(f"{key} => {value}")

**Output:**

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment3\without.py"

Decision Tree:

OpenWorld => {'Open': {'Rating\_M': {'Mature': 'AAA', 'Teen': {'Action': {'No': {'RPG': {'Yes': {'Sports': {'No': {'Multiplayer': {'Online': 'Mid'}}}}}}}}}}, 'Limited': {'Multiplayer': {'Online': {'Sports': {'No': 'Mid', 'Yes': {'Action': {'No': {'RPG': {'No': {'Rating\_M': {'Teen': 'Mid'}}}}, 'Yes': 'Mid'}}}}, 'Offline': {'Action': {'Yes': {'Sports': {'Yes': {'RPG': {'No': {'Rating\_M': {'Teen': 'Mid'}}}}, 'No': 'Indie'}}, 'No': 'Indie'}}}}}

[Done] exited with code=0 in 0.102 seconds

# Experiment - 4

## Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

### Experiment Overview:

To implement and demonstrate the working of an Artificial Neural Network (ANN) using the Backpropagation algorithm for training the network on a dataset and evaluate its performance on classification tasks.

### Advantages:

1. **Universal Function Approximation**: ANNs can approximate any continuous function, making them versatile for various problems.
2. **Pattern Recognition**: Excellent at recognizing complex patterns in data that might not be obvious to humans.
3. **Adaptability**: Can learn from examples and adapt to changing input, allowing them to perform better over time.
4. **Parallel Processing**: Neural networks process information in parallel, enabling efficient computation.

### Disadvantages:

1. **Black Box Nature**: The internal decision-making process is often difficult to interpret.
2. **Data Dependency**: Requires large amounts of data for effective training.
3. **Computational Intensity**: Training can be computationally expensive and time-consuming.
4. **Overfitting Risk**: Prone to overfitting if not properly regularized, especially with limited data.

### Silent Features:

1. **Backpropagation**: The algorithm efficiently calculates gradients to update weights and minimize error.
2. **Gradient Descent**: Optimizes the network by iteratively adjusting weights to reduce loss.
3. **Activation Functions**: Uses functions like sigmoid to introduce non-linearity into the model.
4. **Multi-layer Architecture**: Consists of input, hidden, and output layers for complex pattern recognition.

### Real-world Applications:

1. **Image Recognition**: Used in facial recognition, object detection, and medical image analysis.
2. **Natural Language Processing**: Powers language translation, sentiment analysis, and chatbots.
3. **Financial Forecasting**: Predicts stock prices, credit scoring, and fraud detection.
4. **Medical Diagnosis**: Assists in diagnosing diseases based on symptoms and test results.

### Algorithm:

1. **Initialization**: Randomly initialize weights and biases for the network.
2. **Forward Propagation**: Pass input data through the network to generate output.
3. **Error Calculation**: Compute the difference between predicted output and actual values.
4. **Backward Propagation**: Propagate the error backward through the network to calculate gradients.
5. **Weight Update**: Adjust weights using gradient descent to minimize the error.
6. **Iteration**: Repeat steps 2-5 for multiple epochs until convergence or stopping criteria are met.

### **Dataset:**

For this experiment, we will use the Iris dataset, which is a classic dataset in machine learning:

* Features: Sepal length, Sepal width, Petal length, Petal width
* Classes: Setosa, Versicolor, and Virginica

**Dataset Link:**

https://archive.ics.uci.edu/ml/datasets/iris

### Code With Library:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.datasets import load\_iris

import seaborn as sns

iris = load\_iris()

X = iris.data

y = iris.target.reshape(-1, 1)

encoder = OneHotEncoder(sparse\_output=False)

y\_encoded = encoder.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

class NeuralNetwork:

    def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate=0.01):

        self.input\_size = input\_size

        self.hidden\_size = hidden\_size

        self.output\_size = output\_size

        self.learning\_rate = learning\_rate

        self.W1 = np.random.randn(self.input\_size, self.hidden\_size) \* np.sqrt(1 / self.input\_size)

        self.b1 = np.zeros((1, self.hidden\_size))

        self.W2 = np.random.randn(self.hidden\_size, self.output\_size) \* np.sqrt(1 / self.hidden\_size)

        self.b2 = np.zeros((1, self.output\_size))

        self.loss\_history = []

    def sigmoid(self, x):

        return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

    def sigmoid\_derivative(self, x):

        return x \* (1 - x)

    def forward(self, X):

        self.z1 = np.dot(X, self.W1) + self.b1

        self.a1 = self.sigmoid(self.z1)

        self.z2 = np.dot(self.a1, self.W2) + self.b2

        self.a2 = self.sigmoid(self.z2)

        return self.a2

    def backward(self, X, y, output):

        m = X.shape[0]

        self.output\_error = y - output

        self.output\_delta = self.output\_error \* self.sigmoid\_derivative(output)

        self.hidden\_error = np.dot(self.output\_delta, self.W2.T)

        self.hidden\_delta = self.hidden\_error \* self.sigmoid\_derivative(self.a1)

        self.W2 -= np.dot(self.a1.T, self.output\_delta) \* self.learning\_rate

        self.b2 -= np.sum(self.output\_delta, axis=0, keepdims=True) \* self.learning\_rate

        self.W1 -= np.dot(X.T, self.hidden\_delta) \* self.learning\_rate

        self.b1 -= np.sum(self.hidden\_delta, axis=0, keepdims=True) \* self.learning\_rate

    def compute\_loss(self, y\_true, y\_pred):

        return np.mean(np.square(y\_true - y\_pred))

    def train(self, X, y, epochs=1000):

        for epoch in range(epochs):

            output = self.forward(X)

            loss = self.compute\_loss(y, output)

            self.loss\_history.append(loss)

            self.backward(X, y, output)

            if epoch % 100 == 0:

                print(f"Epoch {epoch}, Loss: {loss}")

    def predict(self, X):

        output = self.forward(X)

        return output

    def evaluate(self, X, y):

        predictions = self.predict(X)

        predicted\_classes = np.argmax(predictions, axis=1)

        true\_classes = np.argmax(y, axis=1)

        accuracy = np.mean(predicted\_classes == true\_classes)

        return accuracy, predictions

input\_size = X\_train.shape[1]

hidden\_size = 8

output\_size = y\_train.shape[1]

nn = NeuralNetwork(input\_size, hidden\_size, output\_size, learning\_rate=0.01)

nn.train(X\_train, y\_train, epochs=1000)

train\_accuracy, train\_predictions = nn.evaluate(X\_train, y\_train)

test\_accuracy, test\_predictions = nn.evaluate(X\_test, y\_test)

print(f"Training Accuracy: {train\_accuracy \* 100:.2f}%")

print(f"Testing Accuracy: {test\_accuracy \* 100:.2f}%")

plt.figure(figsize=(10, 5))

plt.plot(nn.loss\_history)

plt.title('Loss vs. Epochs')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.grid(True)

plt.show()

predicted\_classes = np.argmax(test\_predictions, axis=1)

true\_classes = np.argmax(y\_test, axis=1)

cm = confusion\_matrix(true\_classes, predicted\_classes)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

            xticklabels=iris.target\_names,

            yticklabels=iris.target\_names)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

### Expected Output:

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment4\with.py"

Epoch 0, Loss: 0.2727588096037077

Epoch 100, Loss: 0.7644928265208484

Epoch 200, Loss: 0.7722817068712879

Epoch 300, Loss: 0.7743554774248277

Epoch 400, Loss: 0.7753037366137366

Epoch 500, Loss: 0.775844865779481

Epoch 600, Loss: 0.7761939273217213

Epoch 700, Loss: 0.7764374194651774

Epoch 800, Loss: 0.7766167693165875

Epoch 900, Loss: 0.7767542761323283

Training Accuracy: 25.00%

Testing Accuracy: 20.00%

[Done] exited with code=0 in 14.965 seconds

### 

### 

### Code Without Libraries (Pure Implementation):

import numpy as np

import csv

import random

import math

def load\_data(filename):

    data = []

    labels = []

    with open(filename, 'r') as f:

        reader = csv.reader(f)

        for row in reader:

            if len(row) > 0:

                features = [float(x) for x in row[:-1]]

                data.append(features)

                label = row[-1]

                if label == "Iris-setosa":

                    labels.append([1, 0, 0])

                elif label == "Iris-versicolor":

                    labels.append([0, 1, 0])

                else:

                    labels.append([0, 0, 1])

    return np.array(data), np.array(labels)

def normalize(X):

    X\_norm = np.zeros\_like(X, dtype=float)

    for i in range(X.shape[1]):

        col\_min = np.min(X[:, i])

        col\_max = np.max(X[:, i])

        X\_norm[:, i] = (X[:, i] - col\_min) / (col\_max - col\_min)

    return X\_norm

def train\_test\_split(X, y, test\_size=0.2):

    combined = list(zip(X, y))

    random.shuffle(combined)

    X, y = zip(\*combined)

    X, y = np.array(X), np.array(y)

    split\_idx = int(len(X) \* (1 - test\_size))

    X\_train, X\_test = X[:split\_idx], X[split\_idx:]

    y\_train, y\_test = y[:split\_idx], y[split\_idx:]

    return X\_train, X\_test, y\_train, y\_test

class NeuralNetwork:

    def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate=0.01):

        self.input\_size = input\_size

        self.hidden\_size = hidden\_size

        self.output\_size = output\_size

        self.learning\_rate = learning\_rate

        self.W1 = np.random.randn(self.input\_size, self.hidden\_size) \* 0.01

        self.b1 = np.zeros((1, self.hidden\_size))

        self.W2 = np.random.randn(self.hidden\_size, self.output\_size) \* 0.01

        self.b2 = np.zeros((1, self.output\_size))

        self.loss\_history = []

    def sigmoid(self, x):

        return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

    def sigmoid\_derivative(self, x):

        return x \* (1 - x)

    def forward(self, X):

        self.z1 = np.dot(X, self.W1) + self.b1

        self.a1 = self.sigmoid(self.z1)

        self.z2 = np.dot(self.a1, self.W2) + self.b2

        self.a2 = self.sigmoid(self.z2)

        return self.a2

    def backward(self, X, y, output):

        m = X.shape[0]

        self.output\_error = y - output

        self.output\_delta = self.output\_error \* self.sigmoid\_derivative(output)

        self.hidden\_error = np.dot(self.output\_delta, self.W2.T)

        self.hidden\_delta = self.hidden\_error \* self.sigmoid\_derivative(self.a1)

        self.W2 += np.dot(self.a1.T, self.output\_delta) \* self.learning\_rate

        self.b2 += np.sum(self.output\_delta, axis=0, keepdims=True) \* self.learning\_rate

        self.W1 += np.dot(X.T, self.hidden\_delta) \* self.learning\_rate

        self.b1 += np.sum(self.hidden\_delta, axis=0, keepdims=True) \* self.learning\_rate

    def compute\_loss(self, y\_true, y\_pred):

        return np.mean(np.sum(np.square(y\_true - y\_pred), axis=1))

    def train(self, X, y, epochs=1000):

        for epoch in range(epochs):

            output = self.forward(X)

            loss = self.compute\_loss(y, output)

            self.loss\_history.append(loss)

            self.backward(X, y, output)

            if epoch % 100 == 0:

                print(f"Epoch {epoch}, Loss: {loss:.6f}")

    def predict(self, X):

        return self.forward(X)

    def evaluate(self, X, y):

        predictions = self.predict(X)

        predicted\_classes = np.argmax(predictions, axis=1)

        true\_classes = np.argmax(y, axis=1)

        accuracy = np.mean(predicted\_classes == true\_classes)

        return accuracy

if \_\_name\_\_ == "\_\_main\_\_":

    X, y = load\_data( r"c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment4\iris.csv")

    X = normalize(X)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

    input\_size = X\_train.shape[1]

    hidden\_size = 8

    output\_size = y\_train.shape[1]

    nn = NeuralNetwork(input\_size, hidden\_size, output\_size, learning\_rate=0.01)

    nn.train(X\_train, y\_train, epochs=1000)

    train\_accuracy = nn.evaluate(X\_train, y\_train)

    test\_accuracy = nn.evaluate(X\_test, y\_test)

    print(f"Training Accuracy: {train\_accuracy \* 100:.2f}%")

    print(f"Testing Accuracy: {test\_accuracy \* 100:.2f}%")

### Output:

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment4\without.py"

Epoch 0, Loss: 0.750733

Epoch 100, Loss: 0.665215

Epoch 200, Loss: 0.648014

Epoch 300, Loss: 0.447292

Epoch 400, Loss: 0.330156

Epoch 500, Loss: 0.299378

Epoch 600, Loss: 0.282113

Epoch 700, Loss: 0.269835

Epoch 800, Loss: 0.260237

Epoch 900, Loss: 0.251882

Training Accuracy: 95.83%

Testing Accuracy: 86.67%

[Done] exited with code=0 in 0.358 seconds

## Experiment - 5

## Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

### Experiment Overview:

To implement the Naïve Bayesian classifier algorithm from scratch and evaluate its performance on classification tasks using a dataset stored in CSV format.

### Advantages:

1. **Simplicity**: Easy to implement and understand.
2. **Efficiency**: Requires relatively small amount of training data to estimate parameters.
3. **Speed**: Fast training and prediction compared to more complex models.
4. **Performance with Text**: Works well with text classification problems like spam filtering.
5. **Handles Missing Data**: Can handle missing values by simply ignoring them during probability calculations.

### Disadvantages:

1. **Independence Assumption**: Assumes all features are independent, which is rarely true in real-world data.
2. **Zero Frequency Problem**: Struggles when a categorical variable has a category in test data that wasn't observed in training data.
3. **Probability Estimation**: Doesn't provide reliable probability estimates.
4. **Feature Correlation**: Performance degrades when features are highly correlated.

### Silent Features:

1. **Probabilistic Approach**: Based on Bayes' theorem for conditional probability.
2. **Feature Independence**: Assumes all features are independent given the class label.
3. **Maximum Likelihood**: Uses maximum likelihood estimation for parameter estimation.
4. **Versatile**: Works with both discrete and continuous features.

### Real-world Applications:

1. **Spam Filtering**: Classifying emails as spam or non-spam.
2. **Document Classification**: Categorizing documents into predefined topics.
3. **Sentiment Analysis**: Determining sentiment (positive/negative) from text.
4. **Medical Diagnosis**: Predicting diseases based on symptoms.
5. **Recommendation Systems**: Basic recommendation systems based on user preferences.

### Algorithm:

1. **Data Preparation**: Split data into training and testing sets.
2. **Training Phase**:
   1. Calculate prior probabilities for each class.
   2. Calculate likelihood of features given each class.
3. **Prediction Phase**:
   1. For each test instance, compute posterior probability for each class.
   2. Assign the class with the highest posterior probability.
4. **Evaluation**: Compute accuracy by comparing predicted and actual classes.

### Dataset:

For this experiment, we will use the Iris dataset, a classic dataset for classification tasks:

* Features: Sepal length, Sepal width, Petal length, Petal width
* Classes: Setosa, Versicolor, Virginica

### Code With Library:

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

iris = pd.read\_csv(r"c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment4\iris.csv")

X = iris.iloc[:, :-1].values

y = iris.iloc[:, -1].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

nb\_classifier = GaussianNB()

nb\_classifier.fit(X\_train, y\_train)

y\_pred = nb\_classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100:.2f}%")

print("\nClassification Report:")

print(report)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

            xticklabels=nb\_classifier.classes\_,

            yticklabels=nb\_classifier.classes\_)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

feature\_importance = np.abs(nb\_classifier.theta\_)

feature\_names = iris.columns[:-1]

plt.figure(figsize=(10, 6))

for i, class\_name in enumerate(nb\_classifier.classes\_):

    plt.bar(feature\_names, feature\_importance[i], alpha=0.7, label=class\_name)

plt.xlabel('Features')

plt.ylabel('Importance')

plt.title('Feature Importance per Class')

plt.legend()

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

### Output:

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment5\with.py"

Accuracy: 86.67%

Classification Report:

                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19

Iris-versicolor       0.89      0.62      0.73        13

 Iris-virginica       0.71      0.92      0.80        13

       accuracy                           0.87        45

      macro avg       0.86      0.85      0.84        45

   weighted avg       0.88      0.87      0.86        45

[Done] exited with code=0 in 8.81 seconds

### Code Without Libraries:

import numpy as np

import csv

import random

import math

def load\_csv(filename):

    dataset = []

    with open(filename, 'r') as file:

        csv\_reader = csv.reader(file)

        for row in csv\_reader:

            if not row:

                continue

            dataset.append(row)

    return dataset

def str\_column\_to\_float(dataset, column):

    for row in dataset:

        row[column] = float(row[column].strip())

def str\_column\_to\_int(dataset, column):

    class\_values = [row[column] for row in dataset]

    unique = set(class\_values)

    lookup = dict()

    for i, value in enumerate(unique):

        lookup[value] = i

    for row in dataset:

        row[column] = lookup[row[column]]

    return lookup

def split\_dataset(dataset, split\_ratio):

    train\_size = int(len(dataset) \* split\_ratio)

    train\_set = []

    test\_set = list(dataset)

    while len(train\_set) < train\_size:

        index = random.randrange(len(test\_set))

        train\_set.append(test\_set.pop(index))

    return train\_set, test\_set

def separate\_by\_class(dataset):

    separated = {}

    for i in range(len(dataset)):

        vector = dataset[i]

        class\_value = vector[-1]

        if class\_value not in separated:

            separated[class\_value] = []

        separated[class\_value].append(vector)

    return separated

def mean(numbers):

    return sum(numbers) / float(len(numbers))

def stdev(numbers):

    avg = mean(numbers)

    variance = sum([(x - avg) \*\* 2 for x in numbers]) / float(len(numbers) - 1)

    return math.sqrt(variance)

def summarize\_dataset(dataset):

    summaries = [(mean(column), stdev(column), len(column)) for column in zip(\*dataset)]

    del(summaries[-1])

    return summaries

def summarize\_by\_class(dataset):

    separated = separate\_by\_class(dataset)

    summaries = {}

    for class\_value, rows in separated.items():

        summaries[class\_value] = summarize\_dataset(rows)

    return summaries

def calculate\_probability(x, mean, stdev):

    exponent = math.exp(-((x - mean) \*\* 2 / (2 \* stdev \*\* 2)))

    return (1 / (math.sqrt(2 \* math.pi) \* stdev)) \* exponent

def calculate\_class\_probabilities(summaries, row):

    total\_rows = sum([summaries[label][0][2] for label in summaries])

    probabilities = {}

    for class\_value, class\_summaries in summaries.items():

        probabilities[class\_value] = summaries[class\_value][0][2] / float(total\_rows)

        for i in range(len(class\_summaries)):

            mean, stdev, \_ = class\_summaries[i]

            probabilities[class\_value] \*= calculate\_probability(row[i], mean, stdev)

    return probabilities

def predict(summaries, row):

    probabilities = calculate\_class\_probabilities(summaries, row)

    best\_label, best\_prob = None, -1

    for class\_value, probability in probabilities.items():

        if best\_label is None or probability > best\_prob:

            best\_prob = probability

            best\_label = class\_value

    return best\_label

def naive\_bayes(train, test):

    summarize = summarize\_by\_class(train)

    predictions = []

    for row in test:

        output = predict(summarize, row)

        predictions.append(output)

    return predictions

def accuracy\_metric(actual, predicted):

    correct = 0

    for i in range(len(actual)):

        if actual[i] == predicted[i]:

            correct += 1

    return correct / float(len(actual)) \* 100.0

def evaluate\_algorithm(dataset, algorithm, split\_ratio):

    train, test = split\_dataset(dataset, split\_ratio)

    test\_set = list(test)

    predictions = algorithm(train, test\_set)

    actual = [row[-1] for row in test]

    accuracy = accuracy\_metric(actual, predictions)

    return accuracy

filename = r"c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment5\iris.csv"

dataset = load\_csv(filename)

for i in range(len(dataset[0])-1):

    str\_column\_to\_float(dataset, i)

str\_column\_to\_int(dataset, len(dataset[0])-1)

split\_ratio = 0.7

accuracy = evaluate\_algorithm(dataset, naive\_bayes, split\_ratio)

print(f'Accuracy: {accuracy:.2f}%')

### Output:

[Running] python -u "c:\Users\bhanu\OneDrive\Desktop\@jntua\ML\_lab\Experiment5\without.py"

Accuracy: 95.56%

[Done] exited with code=0 in 0.224 seconds