Machine Learning Lab Manual

**Objective: ** This lab manual provides a set of practical exercises to understand and implement fundamental machine learning algorithms. Each experiment focuses on a specific technique, allowing students to gain hands-on experience with data processing, model building, and evaluation.

**Instructions: **

* For each experiment, carefully read the problem statement and understand the underlying concepts.
* Implement the algorithms using Python (or Java, as specified). You are encouraged to use relevant libraries like NumPy, Pandas, scikit-learn, etc., to streamline your implementations.
* Ensure your code is well-commented and easy to understand.
* Prepare a lab report for each experiment, including the problem statement, algorithm, code, results (including outputs and visualizations where applicable), and your observations/conclusions.

Experiment 1: FIND-S Algorithm

**Problem Statement: ** Implement and demonstrate the FIND-S algorithm to find the most specific hypothesis consistent with a given set of positive training examples. Read the training data from a `.CSV` file.

**Similar Question: ** Consider the following training examples for a concept "EnjoySport":

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Assuming the hypothesis space is a conjunction of attributes, trace the execution of the FIND-S algorithm and determine the final most specific hypothesis.

Conceptual Code (Python):
```python
import pandas as pd

def find_s(data):
    hypothesis = None
    for i, row in data.iterrows():
        if row['EnjoySport'] == 'Yes':
            if hypothesis is None:
                hypothesis = list(row[:-1]) # Initialize with the first positive example
            else:
                for j in range(len(hypothesis)):
                    if hypothesis[j] != row[j]:
                        hypothesis[j] = '?' # Generalize if attributes don't match
    return hypothesis

# Sample CSV data (hypothetical data.csv):
# Sky,AirTemp,Humidity,Wind,Water,Forecast,EnjoySport
# Sunny,Warm,Normal,Strong,Warm,Same,Yes
# Sunny,Warm,High,Strong,Warm,Same,Yes
# Sunny,Warm,High,Strong,Cool,Change,Yes

data = pd.read_csv('data.csv')
specific_hypothesis = find_s(data)
print("Most Specific Hypothesis:", specific_hypothesis)
```

Potential Output:

```
Most Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Experiment 2: Candidate-Elimination Algorithm

Problem Statement: For a given set of training data examples stored in a `.CSV` file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples (version space).

Similar Question: Using the same "EnjoySport" dataset from Experiment 1, trace the Candidate-Elimination algorithm, showing the evolution of the General Boundary (G) and Specific Boundary (S) after each training example.

Conceptual Code (Python):

Python

```
import pandas as pd

def is_consistent(hypothesis, instance):
    for i in range(len(hypothesis)):
        if hypothesis[i] != '?' and hypothesis[i] != instance[i]:
            return False
    return True

def candidate_elimination(data):
    num_attributes = len(data.columns) - 1
    specific_boundary = ['?' for _ in range(num_attributes)]
    general_boundary = [['?' for _ in range(num_attributes)] for _ in
range(num_attributes)]

    for i, row in data.iterrows():
        instance = list(row[:-1])
        target = row['EnjoySport']

        if target == 'Yes':
            for j in range(num_attributes):
                if specific_boundary[j] == '?':
                    specific_boundary[j] = instance[j]
                elif specific_boundary[j] != instance[j]:
                    specific_boundary[j] = '?'

            for g in list(general_boundary):
                if not is_consistent(g, instance):
                    general_boundary.remove(g)

        elif target == 'No':
            new_generalizations = []
            for j in range(num_attributes):
                if specific_boundary[j] != '?' and specific_boundary[j] !=
instance[j]:
                    new_general_hypothesis = list(specific_boundary)
                    new_general_hypothesis[j] = '?'
                    if new_general_hypothesis not in new_generalizations and
new_general_hypothesis not in general_boundary:
                        new_generalizations.append(new_general_hypothesis)

            for new_hyp in new_generalizations:
                is_more_general = True
                for g in general_boundary:
                    if all((gh == '?' or gh == nh) for gh, nh in zip(g,
new_hyp)):
                        is_more_general = False
                        break
                if is_more_general:
                    general_boundary.append(new_hyp)

            general_boundary[:] = [g for g in general_boundary if not all((s
== '?' or s == g[i]) for i, s in enumerate(specific_boundary))]


    final_general_boundary = []
    for g1 in general_boundary:
        is_minimal = True
        for g2 in general_boundary:
            if g1 != g2 and all((g2_val == '?' or g2_val == g1_val) for
g1_val, g2_val in zip(g1, g2)) and any(g1_val != g2_val for g1_val, g2_val in
zip(g1, g2)):
                is_minimal = False
                break
        if is_minimal and g1 not in final_general_boundary:
            final_general_boundary.append(g1)


    return specific_boundary, final_general_boundary

# Assuming 'data.csv' from Experiment 1
data = pd.read_csv('data.csv')
s_boundary, g_boundary = candidate_elimination(data)
print("Specific Boundary (S):", s_boundary)
print("General Boundary (G):", g_boundary)
```

Potential Output (may vary based on the dataset):

```
Specific Boundary (S): ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

```
General Boundary (G): [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?',
'?', '?', '?'], ['?', '?', '?', 'Strong', '?', '?'], ['?', '?', '?', '?',
'?', '?']]
```

Experiment 3: ID3 Algorithm

Problem Statement: 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.

Similar Question: Consider the following "PlayTennis" dataset:

Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rainy	Mild	Normal	Strong	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

Export to Sheets

Calculate the initial entropy of the "PlayTennis" attribute. Then, calculate the information gain for the "Outlook" attribute.

Conceptual Code (Python - using a library for brevity):

Python

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Sample CSV data (hypothetical tennis.csv):
# Outlook,Temperature,Humidity,Wind,PlayTennis
# Sunny,Hot,High,Weak,No
# ... (rest of the data)

data = pd.read_csv('tennis.csv')
X = data.drop('PlayTennis', axis=1)
y = data['PlayTennis']
X = pd.get_dummies(X, drop_first=True) # Convert categorical features

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = DecisionTreeClassifier(criterion='entropy')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Predicting for a new sample
new_sample = pd.DataFrame([{'Outlook_Rainy': 0, 'Outlook_Sunny': 1,
'Temperature_Hot': 0, 'Temperature_Mild': 1, 'Wind_Weak': 1,
'Humidity_Normal': 1}])
prediction = model.predict(new_sample)
print("Prediction for new sample:", prediction)
```

Potential Output (may vary):

```
Accuracy: 0.6666666666666666
Prediction for new sample: ['Yes']
```

Experiment 4: Backpropagation Algorithm

Problem Statement: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Similar Question: Explain the steps involved in the Backpropagation algorithm for a single layer perceptron with a sigmoid activation function. Illustrate with a simple example.

Conceptual Code (Python - using a library for brevity):

Python

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

```
model = MLPClassifier(hidden_layer_sizes=(5,), activation='logistic',
max_iter=1000, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Potential Output (may vary):

```
Accuracy: 0.9777777777777777
```

Experiment 5: Naive Bayesian Classifier

Problem Statement: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a `.CSV` file. Compute the accuracy of the classifier, considering few test data sets.

Similar Question: Given the following training data for classifying emails as "Spam" or "Not Spam":

Word1	Word2	Word3	Class
Yes	No	Yes	Spam
No	Yes	No	Not Spam
Yes	Yes	No	Spam
No	No	Yes	Not Spam

Export to Sheets

Calculate the probability of an email containing (Word1=Yes, Word2=No, Word3=Yes) being classified as "Spam" using the Naive Bayes approach.

Conceptual Code (Python - using a library for brevity):

Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Sample CSV data (hypothetical email.csv):
# Word1,Word2,Word3,Class
# Yes,No,Yes,Spam
# ...

data = pd.read_csv('email.csv')
X = pd.get_dummies(data.drop('Class', axis=1), drop_first=True)
y = data['Class']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = GaussianNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Potential Output (may vary):

```
Accuracy: 0.75
```

Experiment 6: Naive Bayesian Classifier for Document Classification

Problem Statement: Assuming a set of documents that need to be classified, use the naive Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Similar Question (Java Focused): Outline the steps involved in building a Naive Bayes classifier for text classification using Java libraries like Apache Mahout or Weka.

Conceptual Code (Python - using scikit-learn for text processing):

Python

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Sample document data (hypothetical documents.txt - each line is a document
with label)
# This is a positive document. POS
# This is another positive one. POS
# This is a negative review. NEG
# Another negative sentence here. NEG

with open('documents.txt', 'r') as f:
    documents = [line.strip().split(' ', -1) for line in f]
    texts = [' '.join(doc[:-1]) for doc in documents]
    labels = [doc[-1] for doc in documents]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3,
random_state=42)
```

```
model = MultinomialNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

Potential Output (may vary):

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
```

Experiment 7: Bayesian Network for Medical Diagnosis

Problem Statement: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using a standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

Similar Question: Describe the structure of a simple Bayesian network for diagnosing a specific medical condition (e.g., flu) based on symptoms like fever, cough, and sore throat. Define the conditional probability tables for each node.

Conceptual Code (Python - using a library for Bayesian Networks):

Python

```
import pandas as pd
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

# Sample Heart Disease Data (hypothetical heart.csv - simplified)
# ChestPain,BlockedArtery,HeartDisease
# Yes,Yes,Yes
# No,Yes,Yes
# Yes,No,No
# No,No,No

data = pd.read_csv('heart.csv')

# Define the Bayesian Network structure
model = BayesianNetwork([('ChestPain', 'HeartDisease'), ('BlockedArtery',
'HeartDisease')])

# Estimate parameters from data
model.fit(data, estimator=MaximumLikelihoodEstimator)
```

```
# Perform inference
inference = VariableElimination(model)
query_result = inference.query(variables=['HeartDisease'],
evidence={'ChestPain': 'Yes', 'BlockedArtery': 'Yes'})
print(query_result)
```

Potential Output (may vary):

```
+-----------------+-----------------+-----------------------+
| HeartDisease    |   phi(HeartDisease) |
|-----------------|-----------------|-----------------------|
| HeartDisease(0) |          0.1000 | 0.10000000000000002   |
| HeartDisease(1) |          0.9000 | 0.9                   |
```