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.

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

Trace the FIND-S algorithm's execution and identify the last, most specific hypothesis, assuming the hypothesis space is a conjunction of characteristics.

```
Conceptual Code (Python):

import pandas as pd

def generate_specific_hypothesis(dataset):
    most_specific = None

    for index, example in dataset.iterrows():
        if example['EnjoySport'] == 'Yes':
            current_attributes = list(example[:-1])  # exclude target column

            if most_specific is None:
                most_specific = current_attributes.copy()
            else:
                for i in range(len(most_specific)):
                    if most_specific[i] != current_attributes[i]:
                        most_specific[i] = '?'

    return most_specific

# Sample dataset in CSV: 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

df = pd.read_csv('data.csv')
result_hypothesis = generate_specific_hypothesis(df)
print("Final Hypothesis (Specific):", result_hypothesis)
```

Expected Output:

```
Final Hypothesis (Specific): ['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):

```
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] = '?'

            general_boundary = [g for g in general_boundary if is_consistent(g,
instance)]

        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))
            ]

    # Minimize general 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)):
                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

# Sample 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')
s_boundary, g_boundary = candidate_elimination(data)
print("Specific Boundary (S):", s_boundary)
print("General Boundary (G):", g_boundary)
```

Expected Output:

```
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.

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):

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

# Load the tennis dataset
df = pd.read_csv('tennis.csv')

# Separate features and target
features = df.drop('PlayTennis', axis=1)
target = df['PlayTennis']

# Convert categorical features to numeric using one-hot encoding
encoded_features = pd.get_dummies(features, drop_first=True)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    encoded_features, target, test_size=0.2, random_state=42
```

```
)

# Initialize and train the decision tree model
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)

# Make predictions on the test data
predictions = clf.predict(X_test)

# Evaluate the model
acc = accuracy_score(y_test, predictions)
print("Model Accuracy:", acc)

# Prepare a new sample to predict on (make sure the order of columns matches)
sample_data = pd.DataFrame([{
    'Outlook_Sunny': 1,
    'Outlook_Rainy': 0,
    'Temperature_Mild': 1,
    'Temperature_Hot': 0,
    'Humidity_Normal': 1,
    'Wind_Weak': 1
}])

# Align columns with training data (in case of order mismatch)
sample_data = sample_data.reindex(columns=encoded_features.columns,
fill_value=0)

# Predict for the new instance
sample_prediction = clf.predict(sample_data)
print("Should play tennis on new sample?:", sample_prediction[0])
```

Expected Output (may vary):

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

Experiment 4: Backpropagation Algorithm

Problem Statement: Create an artificial neural network using the backpropagation process, then test it with relevant data sets.

The Backpropagation method for a single-layer perceptron with a sigmoid activation function has steps that you need to explain. Give a simple example to show.

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

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

# Load the iris dataset
iris_data = load_iris()
inputs = iris_data.data
labels = iris_data.target

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    inputs, labels, test_size=0.3, random_state=42
)

# Define and train the MLP model (1 hidden layer with 5 neurons, sigmoid
activation)
neural_net = MLPClassifier(
    hidden_layer_sizes=(5,),
    activation='logistic',
    max_iter=1000,
    random_state=42
)
neural_net.fit(X_train, y_train)

# Predict on test set
predicted_labels = neural_net.predict(X_test)

# Evaluate accuracy
model_accuracy = accuracy_score(y_test, predicted_labels)
print("Model Accuracy:", model_accuracy)
```

Expected 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.

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):

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

# Load the email dataset
df = pd.read_csv('email.csv')

# Extract features and labels
features = df.drop('Class', axis=1)
labels = df['Class']

# Convert categorical text features into binary indicators (One-Hot Encoding)
encoded_features = pd.get_dummies(features, drop_first=True)

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    encoded_features, labels, test_size=0.2, random_state=42
)

# Create and train a Naive Bayes model
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# Make predictions
test_predictions = nb_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, test_predictions)
print("Model Accuracy:", accuracy)
```

Expected 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.

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):

```
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

# Load and process the document file
with open('documents.txt', 'r') as file:
    lines = [line.strip().rsplit(' ', 1) for line in file]  # rsplit to handle
last label only

# Separate texts and their labels
text_data = [entry[0] for entry in lines]
text_labels = [entry[1] for entry in lines]

# Convert text data into bag-of-words vectors
bow_vectorizer = CountVectorizer()
X_features = bow_vectorizer.fit_transform(text_data)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_features, text_labels, test_size=0.3, random_state=42
)

# Train a Multinomial Naive Bayes model
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)

# Predict and evaluate
predicted_labels = nb_model.predict(X_test)
acc = accuracy_score(y_test, predicted_labels)
prec = precision_score(y_test, predicted_labels, average='weighted')
rec = recall_score(y_test, predicted_labels, average='weighted')

# Output evaluation metrics
print("Model Accuracy:", acc)
print("Model Precision:", prec)
print("Model Recall:", rec)
```

Expected 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.

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):

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

# Load heart disease dataset
heart_df = pd.read_csv('heart.csv')

# Define the structure of the Bayesian network
heart_model = BayesianNetwork([
    ('ChestPain', 'HeartDisease'),
    ('BlockedArtery', 'HeartDisease')
])

# Learn conditional probability tables using MLE
heart_model.fit(heart_df, estimator=MaximumLikelihoodEstimator)

# Set up inference engine
inference_engine = VariableElimination(heart_model)

# Perform query: P(HeartDisease | ChestPain=Yes, BlockedArtery=Yes)
result = inference_engine.query(
    variables=['HeartDisease'],
    evidence={'ChestPain': 'Yes', 'BlockedArtery': 'Yes'}
)

# Display result
print(result)
```

Expected Output (may vary):

HeartDisease	phi(HD)
HeartDisease = No	0.0
HeartDisease = Yes	1.0