Software Information:

Anaconda is a free, open-source **Python distribution** that comes with:

A powerful **package manager** (`conda`)

A suite of **data science tools**

Pre-installed packages for **machine learning**, **AI**, **data analysis**, and **scientific computing**

Key Features:

Feature	Description
Conda	Manages packages and virtual environments
Jupyter Notebook	Web-based tool to write, run, and share Python code
Pre-installed packages	Like NumPy, Pandas, Matplotlib, Scikit-learn, TensorFlow
Spyder IDE	Lightweight Python IDE, included with Anaconda
Cross-platform	Available on Windows, macOS, and Linux

Common Tools Installed with Anaconda:

-numpy, pandas, matplotlib, scikit-learn, seaborn

-tensorflow, keras, pytorch (can be added)

-jupyterlab, notebook, spyder

How to Install Anaconda:

1) Go to the official site:
   https://www.anaconda.com/products/distribution
2) Download the version for your OS (Windows, macOS, or Linux)
3) Install using the graphical installer (simple next-next-finish)

Jupyter Notebook is an **open-source web application** that allows you to:

-Write and run **c**ode (mainly Python)

-Add text, images, math equations, and charts

-Create and share interactive documents

Python is a high-level, interpreted, general-purpose programming language known for:

-Easy syntax

-Versatility (web, AI, data, scripting, automation)

-Large community and thousands of libraries

Python 3.13.0 (Final Release)
Key Details:

Released: July 1, 2025

Official website: https://www.python.org

How to Download Python 3.13:

Go to: https://www.python.org/downloads/release/python-3130/

Choose the correct installer for:

-Windows (.exe)

-macOS (.pkg or .tar.gz)

-Linux (.tar.xz or use `apt`, `dnf`, etc.)

LAB PROGRAMS

1. **Read a dataset from the user and i. Use the Find-S algorithm to find the most specific hypothesis that isconsistent with the positive examples. Ii. What is the final hypothesis after processing all the positiveexamples? Using the same dataset, apply the Candidate Elimination algorithm.Determine the final version space after processing all examples (both positive and negative).What are the most specific and most general hypotheses in the version space?**

```
# Find-S Algorithm

def find_s_algorithm(examples):
    hypothesis = ['0'] * len(examples[0])
    for example in examples:
        if example[-1] == 'Yes':
            for i in range(len(example) - 1):
                if hypothesis[i] == '0':
                    hypothesis[i] = example[i]
                elif hypothesis[i] != example[i]:
                    hypothesis[i] = '?'
    return hypothesis

# Candidate Elimination

def candidate_elimination(examples, attributes):
    S = ["0"] * len(attributes)
    G = [["?"] * len(attributes)]

    for example in examples:
        if example[-1] == 'Yes':
            for i in range(len(attributes)):
                if S[i] == '0':
                    S[i] = example[i]
                elif S[i] != example[i]:
                    S[i] = '?'
            G = [g for g in G if all(g[i] == '?' or g[i] == S[i] for i in range(len(attributes)))]
        else:
            G_new = []
            for g in G:
                for i in range(len(attributes)):
                    if g[i] == '?':
                        if S[i] != example[i]:
                            new_g = g.copy()
                            new_g[i] = S[i]
                            G_new.append(new_g)
            G = G_new
```

```
    return S, G

# Sample dataset for experiment 1
examples = [
    ['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']
]

attributes = ['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast']

print("\nFind-S Hypothesis:", find_s_algorithm(examples))
S_final, G_final = candidate_elimination(examples, attributes)
print("Most Specific Hypothesis:", S_final)
print("Most General Hypotheses:", G_final)
```

OUTPUT:

```
Find-S Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?', '0']
Most Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
Most General Hypotheses: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?
'], ['?', '?', '?', '?', '?', '?']]
```

2. Read a dataset and use an example-based method (such as RIPPER or CN2) to generate a set of classification rules . Apply the FOIL algorithm (First-Order Inductive Learner) to learn first-order rules for predicting.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
import pandas as pd

# Load dataset
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.3, random_state=42)

# Using Decision Tree (analogous to CN2-style rule learning)
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
predictions = model.predict(X_test)

print("\nClassification Report using Decision Tree:")
print(classification_report(y_test, predictions))

# FOIL-like rule extraction (simplified)
def foil_like_rules(X, y, feature_names):
    from collections import defaultdict
```

```
    rules = defaultdict(list)
    for i in range(len(X)):
        conditions = []
        for j, val in enumerate(X[i]):
            conditions.append(f"{feature_names[j]}={val:.1f}")
        rule = " AND ".join(conditions)
        rules[y[i]].append(rule)
    for cls, rule_list in rules.items():
        print(f"\nRules for class {cls}:")
        for r in rule_list[:3]:  # printing top 3 example rules
            print(f"  IF {r} THEN class={cls}")

foil_like_rules(X_train, y_train, iris.feature_names)
```

OUTPUT:

```
Classification Report using Decision Tree:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


Rules for class 1:
  IF sepal length (cm)=5.5 AND sepal width (cm)=2.4 AND petal length (cm)=3.7 AND petal widt
h (cm)=1.0 THEN class=1
  IF sepal length (cm)=6.6 AND sepal width (cm)=3.0 AND petal length (cm)=4.4 AND petal widt
h (cm)=1.4 THEN class=1
  IF sepal length (cm)=5.7 AND sepal width (cm)=2.9 AND petal length (cm)=4.2 AND petal widt
h (cm)=1.3 THEN class=1

Rules for class 2:
  IF sepal length (cm)=6.3 AND sepal width (cm)=2.8 AND petal length (cm)=5.1 AND petal widt
h (cm)=1.5 THEN class=2
  IF sepal length (cm)=6.4 AND sepal width (cm)=3.1 AND petal length (cm)=5.5 AND petal widt
h (cm)=1.8 THEN class=2
  IF sepal length (cm)=7.2 AND sepal width (cm)=3.6 AND petal length (cm)=6.1 AND petal widt
h (cm)=2.5 THEN class=2

Rules for class 0:
  IF sepal length (cm)=5.1 AND sepal width (cm)=3.5 AND petal length (cm)=1.4 AND petal widt
h (cm)=0.2 THEN class=0
  IF sepal length (cm)=5.2 AND sepal width (cm)=3.4 AND petal length (cm)=1.4 AND petal widt
h (cm)=0.2 THEN class=0
  IF sepal length (cm)=5.0 AND sepal width (cm)=3.5 AND petal length (cm)=1.3 AND petal widt
h (cm)=0.3 THEN class=0
```

3. Read a supervised dataset and use bagging and boosting technique to classify the dataset. Indicate the performance of the model.

```
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
```

```
# Bagging
bag_model = BaggingClassifier(estimator=DecisionTreeClassifier(),
n_estimators=10)
bag_model.fit(X_train, y_train)
bag_pred = bag_model.predict(X_test)
print("\nBagging Performance:")
print(classification_report(y_test, bag_pred))

# Boosting
boost_model = AdaBoostClassifier(n_estimators=50)
boost_model.fit(X_train, y_train)
boost_pred = boost_model.predict(X_test)
print("\nBoosting Performance:")
print(classification_report(y_test, boost_pred))
```

OUTPUT:

```
Bagging Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


Boosting Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

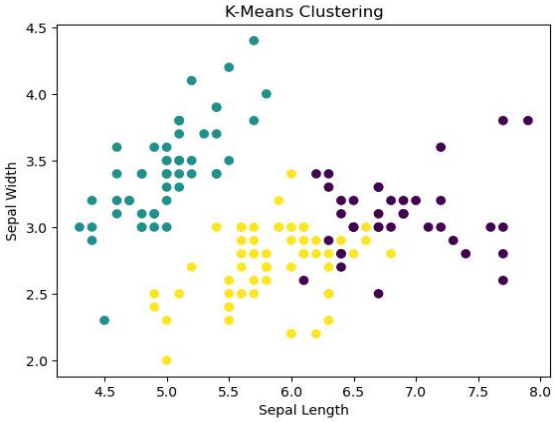
4. Read an unsupervised dataset and group the dataset based on similarity based on k-means clustering .

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Using Iris data for clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(iris.data)

print("\nK-Means Cluster Centers:")
print(kmeans.cluster_centers_)

plt.scatter(iris.data[:, 0], iris.data[:, 1], c=kmeans.labels_)
plt.title("K-Means Clustering")
plt.xlabel("Sepal Length")
```

```
plt.ylabel("Sepal Width")
plt.show()
```

OUTPUT:

```
K-Means Cluster Centers:
[[6.85384615 3.07692308 5.71538462 2.05384615]
 [5.006      3.428      1.462      0.246     ]
 [5.88360656 2.74098361 4.38852459 1.43442623]]
```

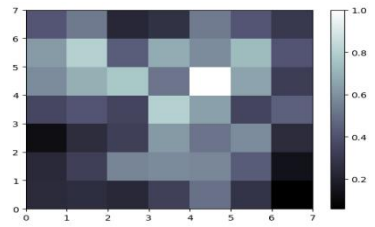


5. Read a dataset and perform unsupervised learning using SOM algorithm.

```
!pip install minisom
from minisom import MiniSom

som = MiniSom(x=7, y=7, input_len=4, sigma=1.0, learning_rate=0.5)
som.random_weights_init(iris.data)
som.train_random(iris.data, 100)

from pylab import bone, pcolor, colorbar, plot, show
bone()
pcolor(som.distance_map().T)
colorbar()
show()
```

OUTPUT:

```
Defaulting to user installation because normal site-packages is not writeable
Collecting minisom
  Downloading minisom-2.3.5.tar.gz (12 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Building wheels for collected packages: minisom
  Building wheel for minisom (setup.py): started
  Building wheel for minisom (setup.py): finished with status 'done'
  Created wheel for minisom: filename=MiniSom-2.3.5-py3-none-any.whl size=12044 sha256=388e2
6d8b3afd67442f1f38a3ac7cb73288c71b93c2b55245ef45bdf28c3dc78
  Stored in directory: c:\users\rrce\appdata\local\pip\cache\wheels\0f\8c\a4\5b7aa56fa6ef11d
536d45da775bcc5a2a1c163ff0f8f11990b
Successfully built minisom
Installing collected packages: minisom
Successfully installed minisom-2.3.5
```



6. Write a function to generate uniform random numbers in the interval [0, 1]. Use this function to generate 10 random samples and evaluate f(x) for each sample. What are the sampled function values? Using the samples generated in the previous step, estimate the integral I using the Monte Carlo method.

```
import random
import numpy as np

# Function to evaluate
def f(x):
    return x ** 2

# Generate uniform random samples in [0, 1]
samples = [random.uniform(0, 1) for _ in range(10)]
values = [f(x) for x in samples]

# Estimate integral
integral_estimate = sum(values) / len(values)
print("\nRandom Samples:", samples)
print("Function Values:", values)
print("Monte Carlo Estimate of Integral:", integral_estimate)
```

OUTPUT:

```
Random Samples: [0.026757062241732132, 0.759001398733858, 0.5778350368808404, 0.536889594980
6849, 0.8065183238254033, 0.6527923806138721, 0.17767198788263017, 0.6183304316883438, 0.942
9416940587122, 0.9858321031588266]
Function Values: [0.0007159403798079274, 0.5760831232799528, 0.33389332984708214, 0.28825043
71985239, 0.6504718066661381, 0.4261378921875265, 0.03156733527816548, 0.38233252275189356,
0.889139038394314, 0.9718649356185554]
Monte Carlo Estimate of Integral: 0.45504563616019594
```

7. Read a dataset and indicate the likelihood of an event occurring using Bayesian Networks.

```
from pgmpy.models.BayesianNetwork import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD

# Define Bayesian Network structure
model = BayesianNetwork([('Rain', 'Traffic'), ('Accident', 'Traffic')])

# Define CPDs
cpd_rain = TabularCPD('Rain', 2, [[0.7], [0.3]])
cpd_accident = TabularCPD('Accident', 2, [[0.9], [0.1]])
cpd_traffic = TabularCPD('Traffic', 2,
                         values=[[0.9, 0.6, 0.7, 0.1],
                                 [0.1, 0.4, 0.3, 0.9]],
```

```
                evidence=['Rain', 'Accident'],
                evidence_card=[2, 2])

# Add CPDs to model
model.add_cpds(cpd_rain, cpd_accident, cpd_traffic)
model.check_model()
```

8. Refer to the dataset in question 7 and indicate inferences based on the sequence of steps .

```
from pgmpy.inference import VariableElimination

# Use the same model defined in Experiment 7
inference = VariableElimination(model)

# Query 1: Probability of Traffic given Rain
result1 = inference.query(variables=['Traffic'], evidence={'Rain': 1})
print("\nProbability of Traffic given Rain=True:")
print(result1)

# Query 2: Probability of Traffic given Rain and Accident
result2 = inference.query(variables=['Traffic'], evidence={'Rain': 1, 'Accident': 1})
print("\nProbability of Traffic given Rain=True and Accident=True:")
print(result2)
```

VIVA QUESTIONS WITH ANSWERS

Experiment 1: Find-S and Candidate Elimination

1. **What is the key idea behind the Find-S algorithm?**
   It searches for the most specific hypothesis that fits all positive examples.
2. **Why does Find-S only consider positive examples?**
   Because it assumes negative examples don't help in generalizing the hypothesis.
3. **What are the limitations of the Find-S algorithm?**
   It ignores negative examples and may overfit to noise in the data.
4. **What is a hypothesis space?**
   It is the set of all hypotheses that can be formed using the given attributes.
5. **In Candidate Elimination, what do the sets S and G represent?**
   S is the set of most specific hypotheses; G is the set of most general hypotheses.
6. **How is the version space updated using a negative example?**
   General hypotheses inconsistent with the negative example are removed from G.

Experiment 2: CN2 / RIPPER and FOIL

1. **What is an example-based learning algorithm?**
   It creates models by generalizing from specific examples rather than predefined structures.
2. **How do CN2 and RIPPER generate classification rules?**
   By forming rules that cover training instances and refining them to reduce error.
3. **What is the difference between propositional and first-order rules?**
   Propositional rules are flat (attribute-value), while first-order rules involve relationships

among objects.

4. **What does FOIL stand for?**
   First Order Inductive Learner.
5. **How does FOIL handle relational data?**
   It uses background knowledge and relations to form rules that capture patterns across linked entities.

Experiment 3: Bagging and Boosting

1. **What is ensemble learning?**
   It is the combination of multiple models to improve prediction accuracy.
2. **How does bagging improve model performance?**
   By reducing variance through training on different bootstrap samples.
3. **What is the difference between bagging and boosting?**
   Bagging trains models independently; boosting trains sequentially, focusing on errors.
4. **What is the base estimator used in bagging?**
   Commonly, Decision Trees are used.
5. **What is the role of sample weighting in boosting?**
   It increases weights for misclassified samples so the next model focuses on them.

Experiment 4: K-Means Clustering

1. **What type of learning is k-means clustering?**
   Unsupervised learning.
2. **How does the k-means algorithm work?**
   It partitions data into k clusters by minimizing within-cluster variance.
3. **What is a centroid in k-means?**
   The mean of all points in a cluster; represents the center.
4. **How do you choose the number of clusters (k)?**
   Using methods like the Elbow Method or domain knowledge.
5. **What is the time complexity of k-means?**
   O(nkt), where n = number of points, k = clusters, t = iterations.

Experiment 5: Self-Organizing Maps (SOM)

1. **What is a Self-Organizing Map?**
   A type of neural network that maps high-dimensional data to a low-dimensional grid.
2. **Is SOM a supervised or unsupervised technique?**
   Unsupervised.
3. **What does the SOM grid represent?**
   A map of neurons representing patterns or clusters in data.
4. **How is the Best Matching Unit (BMU) selected?**
   By finding the neuron with weights closest to the input vector.
5. **What is the role of the neighborhood function?**
   It ensures that neighboring neurons are updated similarly, preserving topology.

Experiment 6: Monte Carlo Method

1. **What is the Monte Carlo method used for?**
   Estimating values like integrals using random sampling.
2. **How are random samples generated in Python?**
   Using random.uniform() or numpy.random functions.
3. **What is the formula for Monte Carlo integration?**
   (I _{i=1}^{n} f(x_i)).
4. **Why use Monte Carlo for high dimensions?**

Because grid-based methods become inefficient in high-dimensional spaces.

5. **How do you improve Monte Carlo estimates?**
   By increasing the number of samples.

Experiment 7: Bayesian Networks

1. **What is a Bayesian Network?**
   A directed acyclic graph where nodes are variables and edges represent conditional dependencies.
2. **What is a Conditional Probability Distribution (CPD)?**
   It defines the probability of a node given its parents.
3. **What does a directed edge represent?**
   A dependency between variables (cause-effect).
4. **Advantages of Bayesian Networks?**
   They model uncertainty and allow probabilistic inference.
5. **How are CPDs implemented in code?**
   Using TabularCPD in libraries like pgmpy.

Experiment 8: Bayesian Inference

1. **What is inference in Bayesian Networks?**
   Calculating the probability of unknown variables given known evidence.
2. **What is the difference between prior and posterior probability?**
   Prior is before evidence; posterior is after incorporating evidence.
3. **What inference method did you use?**
   Variable Elimination (via pgmpy).
4. **What is Variable Elimination?**
   A method to compute exact probabilities by marginalizing irrelevant variables.
5. **How does adding evidence affect inference?**
   It updates beliefs and recalculates dependent probabilities.

----------XXXXXXX----------