

Table of Contents

Lab 1 : Intelligent Agents	1
Objective	1
Basic Theory	1
Experimentation	2
Task 1: Simple Reflex Agent	2
Task 2: Model-Based Reflex Agent	3
Task 3: Custom Environment - Blind Dog in Park	4
Task 4: Performance Comparison Table	5
Discussion	5
Conclusion	6
Lab 2: Basics of Natural Language Processing (NLP)	7
Objective	7
Basic Theory	7
Experimentation	8
Discussion	10
Conclusion	10
Lab 3: K-Nearest Neighbors (KNN) Classification	11
Objective	11
Basic Theory	11
Experimentation	12
Discussion	13
Conclusion	13
Lab 4: Artificial Neural Network (ANN) for Letter Classification	14
Objective	14
Basic Theory	14
Experimentation	15
Discussion	16
Conclusion	16
Lab 5: Generative AI-Based Tools	17
Basic Theory	17
Experimentation/Study of Tools	17
Discussion	18
Conclusion	18

Lab 1: Intelligent Agents

Implementation of Intelligent Agents and Reflex Agent

Objective

The objective of this lab are:

- To understand intelligent agents and environment interaction.
- To implement a Simple Reflex Agent in Python.
- To implement a Model-Based Reflex Agent with memory.
- To design a custom environment class.
- To compare performance between different agents.

Basic Theory

An Intelligent Agent is a system that perceives its environment through sensors and acts upon the environment using actuators. The behavior of an agent is determined by the agent program.

Artificial Intelligence agents are designed to perform tasks automatically and intelligently.

Concepts

- **Agent**: An agent is anything that can perceive its environment and take actions.
- **Environment**: The environment is the surrounding where the agent operates.
- **Simple Reflex Agent**: A simple Reflex Agent acts only according to the current percept and does not remember previous states.
- **Model Based Reflex Agent**: A Model-Based Agent maintains memory of past percepts and uses an internal model to make better decisions.

Types of Intelligent Agents

1. Simple Reflex Agent
2. Model Based Reflex Agent
3. Goal Based Agent
4. Utility Based Agent
5. Learning Agent

Vacuum world Environment

The vacuum world consists of two locations:

- A
- B

Each location can either be:

- clean
- Dirty

Actions Performed By Agent

- Suck
- Left
- Right
- NoOp

Dataset Used

No external dataset was used in this experiment.

Experimentation

Task 1: Simple Reflex Agent

Source code:

```
from agents import Agent
from agents import TrivialVacuumEnvironment
loc-A, loc-B = (0,0), (1,0)
def simple-reflex-program(percept):
    location, status = percept
    if status == 'Dirty':
        return 'Suck'
    elif location == loc-A:
        return 'Right'
    elif location == loc-B:
        return 'Left'
reflex-agent = Agent(simple-reflex-program)
env = TrivialVacuumEnvironment()
env.add-thing(reflex-agent)
print("Initial Environment: ", env.status)
env.run(10)
print("Final Environment: ", env.status)
print("Performance Score: ", reflex-agent.performance)
```

Output:

Initial Environment: $\{(0,0): \text{'Dirty'}, (1,0): \text{'Clean'}\}$
Final Environment: $\{(0,0): \text{'Clean'}, (1,0): \text{'Clean'}\}$
Performance Score: 7

Observation:

- The Simple Reflex Agent reacts according to current percept only.
- It cleans dirty locations using the "Suck" action.
- After both rooms become clean, the agent keeps moving left and right.
- The agent has no memory of previous states.

Task 2: Model Based Reflex Agent

Source Code:

```
from agents import Agent
from agents import TrivialVacuumEnvironment
loc_A, loc_B = (0,0), (1,0)
def model-based-program(percept):
    location, status = percept
    model-based-program.model[location] = status
    if model-based-program.model[loc_A] == 'clean' and \
        model-based-program.model[loc_B] == 'clean':
        return 'NoOp'
    elif status == 'Dirty':
        return 'Suck'
    elif location == loc_A:
        return 'Right'
    elif location == loc_B:
        return 'Left'
model-based-program.model = {
    loc_A: None,
    loc_B: None
}
model_agent = Agent(model-based-program)
env = TrivialVacuumEnvironment()
env.add-thing(model_agent)
env.run(10)
print("Final Environment: ", env.status)
print("Performance: ", model_agent.performance)
```

Output:

Final Environment: {(0,0): 'clean', (1,0): 'clean'}

Performance: 18

Observation

- The Model-Based Agent stores information about visited locations.
- It stops automatically after both locations become clean.
- The performance score is better than the simple Reflex Agent.
- Memory helps the agent avoid unnecessary movement.

Task 3: Custom Environment - Blind Dog in Park

Source code:

```
from agents import Agent, Environment, Thing
class Food(Thing):
    pass
class Water(Thing):
    pass
class Park(Environment):
    def percept(self, agent):
        return self.list_things_at(agent.location)
    def execute_action(self, agent, action):
        if action == 'move down':
            agent.location += 1
            agent.performance -= 1
        elif action == 'eat':
            items = self.list_things_at(agent.location, tclass = Food)
            if items:
                self.delete_thing(items[0])
                agent.performance += 10
        elif action == 'drink':
            items = self.list_things_at(agent.location, tclass = Water)
            if items:
                self.delete_thing(items[0])
                agent.performance += 5
class BlindDog(Agent):
    location = 1
    def __init__(self):
        def program(percepts):
            for thing in percepts:
                if isinstance(thing, Food):
                    return 'eat'
                if isinstance(thing, Water):
                    return 'drink'
            return 'move down'
        super().__init__(program)
Park = Park()
dog = BlindDog()
Park.add_thing(dog, 1)
Park.add_thing(Food(), 5)
Park.add_thing(Water(), 8)
Park.add_thing(Food(), 12)
Park.run(15)
print("Dog performance:", dog.performance)
```

Output:

Dog Performance: 12

Observation

- The BlindDog agent moves step by step in the environment.
- The dog gains performance points after eating food and drinking water.
- The agent loses points while moving.
- The environment updates dynamically.

Task 4: Performance Comparison Table

Agent Name	Average Score	Has Memory?	Observation
ReflexVacuumAgent	7	No	Keeps moving even after cleaning.
ModelBasedVacuumAgent	18	Yes	Stops after cleaning both rooms.
BlindDogAgent	12	No	collects food and water while moving.

Discussion

The lab demonstrated the working principles of intelligent agents in Artificial Intelligence. The Simple Reflex Agent works only on current percepts and lacks memory, whereas the Model-Based Agent maintains internal memory and performs better.

The BlindDog custom environment showed how agents interact with objects inside environments using actions and percepts.

This lab helped in understanding:

- Agent-environment interaction.
- Agent memory
- Performance evaluation
- Environment simulation

Conclusion

The experiment successfully implemented different intelligent agents in Python. The simple Reflex Agent was able to react to current conditions, while the Model-Based Agent used memory to improve performance.

The custom BlindDog environment demonstrated how agents can interact with complex environments. Overall, the lab provided practical knowledge about intelligent agent and their behavior in Artificial Intelligence system.

Lab 2: Basics of Natural Language Processing (NLP)

Implementation of Basic Natural Language Processing using Python

Objective

The objective of this lab are:

- To understand the fundamentals of Natural Language Processing (NLP).
- To perform lexical analysis on text.
- To implement tokenization using Python.
- To perform basic text analysis.
- To understand the role of grammar in language processing.

Basic Theory

Natural Language Processing (NLP) is a branch of Artificial Intelligence that enables computers to understand, analyze, interpret, and generate human language. NLP combines computational linguistics with machine learning and deep learning techniques to process textual and spoken language.

NLP is widely used in applications such as chatbots, language translation, spam filtering, sentiment analysis, speech recognition, and search engines.

NLP Tools

- **NLTK (Natural Language Toolkit)**: Used for tokenization, stemming, lemmatization, and text processing.
- **spaCy**: Efficient library for industrial NLP applications.
- **TextBlob**: Simple library for text processing and sentiment analysis.
- **Transformers (Hugging Face)**: Used for advanced language models like BERT and GPT.

Key Concepts

Lexical Analysis

Lexical Analysis is the process of breaking text into smaller units called tokens. These tokens may be words, numbers, or punctuation marks.

Tokenization

Tokenization divides a sentence into individual words or tokens for further processing.

Example:

Input:

Artificial Intelligence is amazing.

Output:

['Artificial', 'Intelligence', 'is', 'amazing', '.']

Grammar Analysis

Grammar analysis studies the grammatical structure of a sentence to determine relationships between words.

Text Analysis

Text analysis extracts useful information from textual data by ~~counting~~ counting words, identifying patterns, and analyzing sentence structure.

Dataset Used

No external dataset was used in this experiment.

Sample text:

Artificial Intelligence is transforming the world. NLP helps computers understand human language.

Experimentation

Source code:

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk import pos_tag
from collections import Counter
# Download required resources (run once)
nltk.download('punkt')
nltk.download('averaged-perceptron-tagger')

text = "Artificial Intelligence is transforming the world. NLP helps computers understand human language."
# Sentence Tokenization
sentences = sent_tokenize(text)
print("Sentences:")
print(sentences)
# Word Tokenization
words = word_tokenize(text)
print("\n Words:")
print(words)
# Number of words
print("In Total words:", len(words))
# Word Frequency
frequency = Counter(words)
print("\n Word Frequency:")
print(frequency)
# Part-of-Speech Tagging
tags = pos_tag(words)
print("\n POS Tags:")
print(tags)
```


Output:

Sentences:

['Artificial Intelligence is transforming the world.',
'NLP helps computers understand human language.']

Words:

['Artificial', 'Intelligence', 'is', 'transforming', 'the',
'world', '.', 'NLP', 'helps', 'computers',
'understand', 'human', 'language', '.']

Total Words: 14

Word Frequency:

Counter{

': 2,

'Artificial': 1,

'Intelligence': 1,

'is': 1,

'transforming': 1,

'the': 1,

'world': 1,

'NLP': 1,

'helps': 1,

'computers': 1,

'understand': 1,

'human': 1,

'language': 1

})

POS Tags:

[('Artificial', 'JJ'),

('intelligence', 'NN'),

('is', 'VBZ'),

('transforming', 'VBG'),

('the', 'DT'),

('world', 'NN'),

('.', '.'),

('NLP', 'NNP'),

('helps', 'VBZ'),

('computers', 'NNS'),

('understand', 'VB'),

('human', 'JJ'),

('language', 'NN'),

('.', '.')]]

Discussion

The experiment demonstrated the basic concepts of Natural Language Processing using Python and the NLTK library. The input text was divided into sentences and words through tokenization. Lexical analysis identified individual tokens, while word frequency analysis showed the occurrence of each word. Part-of-Speech tagging classified each token according to its grammatical role, such as noun, verb, adjective, or determiner. These preprocessing techniques are fundamental steps for advanced NLP tasks like text classification, machine translation, question answering, and sentiment analysis.

Conclusion

The experiment successfully demonstrated the implementation of basic Natural Language Processing techniques. Lexical analysis, tokenization, word frequency analysis, and grammar based Part-of-Speech tagging were performed using Python. The lab provided practical knowledge of text processing, which serves as the foundation for developing intelligent language-processing applications such as chatbots, virtual assistants, and machine translation systems.

Lab 5: K-Nearest Neighbors (KNN) Classification Implementation of K-Nearest Neighbors (KNN) Algorithm Using Python.

Objective

The objective of this lab are:

- To understand the K-Nearest Neighbors (KNN) algorithm.
- To implement KNN using Python.
- To classify data based on the nearest neighbors.
- To evaluate the performance of the classifier.

Basic Theory

K-Nearest Neighbors (KNN) is a simple and widely used supervised machine learning algorithm used for classification and regression tasks. It predicts the class of a new data point by identifying the K nearest training samples based on a distance metric such as Euclidean distance.

The value of K represents the number of nearest neighbors considered during classification. The class that appears most frequently among the K neighbors is assigned to the new data point.

Advantages

- Simple and easy to implement.
- No training phase is required.
- Effective for small datasets.

Limitations

- Computationally expensive for large datasets.
- Sensitive to irrelevant features and noisy data.
- Choosing the optimal value of K can be challenging.

Dataset Used

Dataset Name: Iris Dataset

Dataset Description

The Iris dataset is one of the most popular datasets in machine learning. It contains 150 flower samples belonging to three different species:

- Iris Setosa
- Iris Versicolor
- Iris Virginica

Each sample contains four features:

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width

The objective is to classify the flowerspecies based on these features.

Experimentation:

Source code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

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

# Create KNN model
knn = KNeighborsClassifier(n_neighbors=3)

# Train model
knn.fit(X_train, y_train)

# Predict
y_pred = knn.predict(X_test)

# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Output:

Accuracy: 1.0

Confusion Matrix:

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

(Accuracy may vary slightly depending on the train-test split)

Discussion

The K-Nearest Neighbors algorithm successfully classified the Iris flower dataset using four numerical features. The classifier compared each test sample with its nearest neighbors and predicted the class based on majority voting. The high accuracy demonstrates that KNN performs well on small, structured datasets like Iris. However, the algorithm becomes slower as the dataset size increases because it calculates distances to all training samples during prediction.

Conclusion

The experiment successfully implemented the K-Nearest Neighbors algorithm using Python and the Scikit-learn library. The Iris dataset was classified with high accuracy, demonstrating the effectiveness of KNN for classification tasks. This lab provided practical experience in data preprocessing, model training, prediction, and performance evaluation using a supervised machine learning algorithm.

Implementation of Letter Classification using ANN

Objective

The objective of this lab are:

- To understand the concept of Artificial Neural Network (ANN).
- To implement an ANN model for letter classification.
- To train and evaluate the neural network using a letter recognition dataset.
- To understand how ANN learns patterns from data.

Basic Theory

Artificial Neural Network (ANN) is a machine learning model inspired by the structure and functioning of the human brain. It consists of interconnected processing units called neurons, which work together to process information and solve complex problems such as classification, prediction and pattern recognition.

An ANN is composed of three main layers:

- **Input Layer:** Receives input features.
- **Hidden Layer(s):** Processes the information through weighted connections.
- **Output Layer:** Produces the final prediction or classification.

ANNs learn by adjusting the weights of connections during training using algorithms such as Backpropagation.

Components of ANN

- Input Layer
- Hidden Layer
- Output Layer
- Weights
- Bias
- Activation Function

Activation Function

Activation functions introduce non-linearity into the network. Common activation functions include:

- ReLU (Rectified Linear Unit)
- Sigmoid
- Tanh
- Softmax

Dataset Used
Dataset Name: Letter Recognition Dataset

Dataset Description
The Letter Recognition Dataset contains 20,000 samples representing the 26 uppercase English letters (A-Z).

Each sample contains 16 numerical features extracted from images of handwritten letters. The objective is to classify each sample into its corresponding alphabet letter.

Experimentation

Source Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Load dataset
data = pd.read_csv("letter-recognition.csv")

# Features and target
X = data.iloc[:, 1:]
y = data.iloc[:, 0]

# Encode labels
encoder = LabelEncoder()
y = encoder.fit_transform(y)

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

# Create ANN model
ann = MLPClassifier(
    hidden_layer_sizes=(100),
    max_iter=500,
    random_state=42
)

# Train model
ann.fit(X_train, y_train)

# Prediction
y_pred = ann.predict(X_test)

# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output:

Accuracy: 0.97 (this ~~entire~~ accuracy may vary depending on dataset split)

Discussion

The Artificial Neural Network successfully learned the patterns from the training data and classified ~~hardware~~ handwritten letters with high accuracy. During training, the network adjusted its weights using the backpropagation algorithm to minimize prediction errors. The Letter Recognition Dataset provided sufficient examples for the ANN to distinguish between different alphabet letters. This experiment demonstrated the effectiveness of neural networks for solving classification problems involving pattern recognition.

Conclusion

The experiment successfully implemented an Artificial Neural Network for letter classification using Python. The ANN achieved high classification accuracy by learning the relationships between input features and output classes. This lab provided practical knowledge of neural network architecture, model training, prediction, and evaluation. It also demonstrated the importance of ANNs in image recognition, handwriting recognition, and other Artificial Intelligence applications.

Lab 5: Generative AI - Based Tools

Study and Overview of Generative AI - Based Tools

Basic Theory

Generative Artificial Intelligence (Generative AI) refers to the AI systems that can create new content such as text, images, audio, video and code. These models are trained on large datasets and learn patterns to generate human-like outputs.

Generative AI is based on deep learning models such as:

- Large Language Models (LLMs)
- Diffusion Models
- Generative Adversarial Networks (GANs)
- Transformer-based architectures

These tools are widely used in education, software development, content creation, healthcare, marketing, and entertainment.

Dataset Used

Not applicable (theoretical study and tool-based analysis)

Experimentation/Study of Tools

In this lab, we studied and explored various Generative AI tools used for different applications.

1. Text-Based Generative AI (LLMs)

These tools generate human-like text responses, code and explanations.

- **ChatGPT** - AI assistant for ~~conversation~~, ~~coding~~ conversation, coding, writing.
- **Gemini** - Google's multimodal AI assistant
- **Claude** - AI assistant focused on reasoning and writing
- **Microsoft Copilot** - AI integrated into Microsoft tools.

2. Text-to-Image Generation Tools

These tools generate images from text prompts.

- **DALL·E** - Creates realistic and artistic images from prompts
- **Midjourney** - High quality artistic image generation tool
- **Stable Diffusion** - Open-source image generation model
- **Adobe Firefly** - AI-powered creative design tool

3. Text-to-Video Generation Tools

These tools generate short videos from text descriptions.

- **Runway ML** - AI video generation and editing
- **Pika Labs** - Text-to-video generation tool
- **Synthesia** - AI avatar-based video creation

4. AI Voice and Audio Tools

These tools generate speech, music, and audio content.

- **ElevenLabs** - AI voice cloning and speech generation.
- **Suno AI** - AI music generation tool
- **Udio** - AI-based music composition tool

Output/Observation

- Generative AI tools can produce high-quality text, images, videos and audio.
- They significantly reduce time required for content creation.
- Outputs vary based on prompt quality and model type.
- These tools are widely used in real-world applications such as education, media and software development.

Discussion

The study of Generative AI tools shows how artificial intelligence has evolved from simple data processing to content creation systems. Large Language Models like ChatGPT and Gemini can understand context and generate meaningful responses, while diffusion models like Midjourney and Stable Diffusion generate realistic images from text prompts. These tools demonstrate the power of deep learning and transformer architectures. However, they also raise concerns such as misinformation, ethical use, copyright issues, and dependency on AI-generated content.

Conclusion

This lab provided an overview of various generative AI tools and their applications. It helped in understanding how AI can generate text, images, videos, and audio content using advanced machine learning models. Generative AI is a rapidly growing field that is transforming industries and improving productivity across multiple domains.