PROJECT BASED LEARNING REPORT
on
“VITERBI ALGORITHM”

Submitted in the partial fulfillment of the requirements

for the Project based learning (PBL) INFORMATION THEORY AND COADING

in

Electronics & Communication Engineering

By

PRN	NAME	ROLL NUMBER
2214110408	SNEHA AMODKAR	02
2214110427	APOORVA DAS	34
2214110446	ANUSHKA SRIVASTAV	31

Under the guidance of Course In-charge

Prof. TANUJA DHOPA

Department of Electronics & Communication Engineering

Bharati Vidyapeeth (Deemed to be University)
College of Engineering,
Pune – 4110043

Academic Year: 2024-25

CERTIFICATE

Certified that the Project Based Learning report entitled, "VITERBI ALGORITHM in PYTHON" is work done by

2214110407	SNEHA AMODKAR
2214110427	APOORVA DAS
2214110446	ANUSHKA SRIVASTAV

in partial fulfillment of the requirements for the award of credits for Project Based Learning (PBL) in Electromagnetic Waves & Propagation of Bachelor of Technology Semester V, in Department of Electronics and Communication Engineering.

Date:

Prof. Tanuja Dhopa	Dr. Arundhati A. Shinde
Course In-charge	Professor & Head

INDEX

CHAPTER-1

Why Do We Need "Viterbi Algorithm"

The Viterbi algorithm is crucial because it efficiently solves the problem of finding the most likely sequence of hidden states in systems with uncertainty, such as **Hidden Markov Models (HMMs)**, which are common in fields like speech recognition, digital communications, and bioinformatics.

Key Applications:

1. **Hidden Markov Models (HMMs)**:
    - HMMs represent systems where the observed data is generated by hidden states (e.g., in speech recognition, the hidden states are phonemes, and the observed data is sound waves).
    - The Viterbi algorithm efficiently determines the most probable hidden state sequence given a sequence of observations, ensuring accurate interpretation of noisy or incomplete data.
2. **Error Correction in Digital Communications**:
    - In wireless and satellite communications, the transmitted data often gets corrupted by noise. The Viterbi algorithm is used in **convolutional codes** to correct errors by identifying the most likely transmitted bit sequence from the received, noisy signal.
    - This makes it invaluable in systems like mobile phones, satellite links, and deep-space communication, where error-free data transmission is critical.
3. **Bioinformatics (Gene and Protein Sequence Alignment)**:
    - In computational biology, the Viterbi algorithm is used to align DNA, RNA, or protein sequences by finding the best match between sequences, crucial for understanding evolutionary relationships or predicting gene structures.
    - It plays a key role in gene prediction by identifying which regions of a genome are likely to code for proteins.

Solution Of "Viterbi Algorithm"

The **Viterbi algorithm** finds the most probable sequence of hidden states in a **Hidden Markov Model (HMM)**, given a sequence of observations. Here's a simplified explanation:

Problem Setup:

- **States**: {s1,s2,…,sN}\{s_1, s_2, \dots, s_N\}{s1,s2,…,sN}
- **Observations**: {o1,o2,…,oT}\{o_1, o_2, \dots, o_T\}{o1,o2,…,oT}
- **Transition Probabilities**: P(sj|si)P(s_j | s_i)P(sj|si) (probability of moving from sis_isi to sjs_jsj)
- **Emission Probabilities**: P(ot|si)P(o_t | s_i)P(ot|si) (probability of observing oto_tot from state sis_isi)
- **Initial Probabilities**: P(si)P(s_i)P(si)

Steps:

1. **Initialization**:

   δ1(i)=P(si)×P(o1|si)\delta_1(i) = P(s_i) \times P(o_1 | s_i)δ1(i)=P(si)×P(o1|si)

   This initializes the probability for each state at time t=1t = 1t=1.

2. **Recursion** (for each time step t=2,…,Tt = 2, \dots, Tt=2,…,T):

   δt(j)=max⁡i[δt−1(i)×P(sj|si)]×P(ot|sj)\delta_t(j) = \max_i [\delta_{t-1}(i) \times P(s_j | s_i)] \times P(o_t | s_j)δt(j)=imax[δt−1(i)×P(sj|si)]×P(ot|sj)

   This computes the most probable path to each state.

3. **Termination**:

   P∗=max⁡iδT(i)P^* = \max_i \delta_T(i)P∗=imaxδT(i)

   This finds the most likely final state.

4. **Backtracking**: Using stored backpointers, trace back from the final state to determine the most likely sequence of hidden states.

Example:

For a simple weather model with states like "Rainy" and "Sunny" and observations like "walk", "shop", and "clean", the Viterbi algorithm finds the most likely sequence of weather conditions that explains the observations.

CHAPTER-2

INTRODUCTION

The **Viterbi algorithm** is a dynamic programming algorithm designed to find the most probable sequence of hidden states in systems modeled by **Hidden Markov Models (HMMs)**. HMMs are statistical models where the system being modeled is assumed to follow a Markov process with hidden states. The algorithm was introduced by Andrew Viterbi in 1967 and has since become an essential tool in many fields, especially those involving signal processing, communications, and computational biology.

Key Concept:

The Viterbi algorithm addresses a fundamental problem in systems with uncertainty: given a sequence of observed events, it calculates the most likely sequence of hidden states that could have generated those observations. This is useful in a wide range of applications where observations are noisy or incomplete, and the goal is to infer the underlying state sequence that produced them.

How It Works:

The algorithm operates by systematically computing the most probable path to each state at each time step using dynamic programming. At each time step, it chooses the best transition (based on probabilities) from the previous state, ensuring that unnecessary computations are avoided. Once the final state is reached, the algorithm traces back through the stored paths to construct the full sequence of states.

Efficiency:

The Viterbi algorithm is highly efficient because it avoids recalculating paths for each possible sequence of states. Instead, it breaks the problem into subproblems, building on previously computed results. This makes it suitable for real-time applications, as it operates in **$O(n * m^2)$** time, where n is the length of the observation sequence and m is the number of hidden states.

Applications:

1. **Telecommunications**: The algorithm is extensively used in decoding convolutional codes, a type of error-correcting code in digital communications. It helps recover transmitted data that has been corrupted by noise, making it essential for systems like mobile phones, satellite communication, and deep-space probes.
2. **Speech Recognition**: In automatic speech recognition, the Viterbi algorithm helps determine the most likely sequence of phonemes (or words) that produced the observed acoustic signal, which is crucial for accurate speech-to-text conversion.
3. **Bioinformatics**: In computational biology, the algorithm is used to align DNA, RNA, and protein sequences. It helps predict gene structures and find the most likely evolutionary paths between sequences, essential for understanding biological functions and evolutionary relationships.
4. **Natural Language Processing (NLP)**: The algorithm is used in part-of-speech tagging, where each word in a sentence is assigned its most probable syntactic category based on observed data. It also helps in sequence labeling tasks.

Why It’s Important:

The Viterbi algorithm’s ability to handle uncertainty in a probabilistic framework, coupled with its computational efficiency, makes it indispensable in any domain that requires decoding noisy or uncertain data. It is favored in real-time systems where quick and accurate decoding is essential, such as in mobile communications, automatic speech recognition, and biological sequence analysis.

In summary, the Viterbi algorithm is a powerful and efficient tool for solving problems that involve finding the most likely sequence of events or states in systems with hidden information. Its widespread use and versatility have made it a cornerstone in areas where data interpretation is crucial, particularly in fields where signals are prone to noise or uncertainty

CHAPTER-3

SOFTWARE USED

VS CODE

Visual Studio Code (VS Code) is a popular open-source code editor developed by Microsoft. It's widely used for various programming languages and offers a range of features that make coding more efficient and enjoyable. Here are some key aspects of VS Code:

Key Features

1. **IntelliSense**: Offers smart completions based on variable types, function definitions, and imported modules.
2. **Debugging**: Built-in debugging support allows you to debug code directly within the editor, set breakpoints, and step through code.
3. **Extensions**: A vast marketplace of extensions lets you customize your environment, adding support for additional programming languages, tools, themes, and more.
4. **Integrated Terminal**: You can run command-line tasks directly in the editor without switching windows.
5. **Git Integration**: Built-in version control features allow you to manage your Git repositories, view diffs, and make commits.
6. **Customizable Interface**: You can personalize the layout, color themes, and settings to match your workflow.
7. **Remote Development**: Supports remote development, allowing you to connect to a remote server or container.
8. **Markdown Support**: Includes a Markdown editor with preview functionality, which is great for documentation.

Supported Languages

VS Code supports a wide range of programming languages, including:

- JavaScript
- Python
- Java
- C/C++
- Go
- PHP
- TypeScript
- HTML/CSS

Platforms

VS Code is cross-platform, available on Windows, macOS, and Linux, making it accessible to a wide range of developers.

Getting Started

To get started with VS Code:

1. **Download and Install**: You can download it from the official website.
2. **Install Extensions**: Explore the Extensions Marketplace to find tools that suit your development needs.
3. **Create or Open a Project**: You can create a new file, open an existing folder, or clone a repository from Git.
4. **Explore Features**: Familiarize yourself with the interface, including the Activity Bar, Side Bar, and Editor.

If you have specific questions about using VS Code or need tips on certain features, feel free to ask!

Python is a high-level, interpreted programming language known for its readability and versatility. It was created by Guido van Rossum and first released in 1991. Here's an overview of its key features, uses, and community:

Key Features

1. **Readability**: Python's syntax is designed to be clear and intuitive, making it accessible to beginners.
2. **Interpreted Language**: Python code is executed line-by-line, which makes debugging easier but can be slower than compiled languages.
3. **Dynamic Typing**: You don't need to declare variable types explicitly; Python determines the type at runtime.
4. **Extensive Libraries**: Python has a rich ecosystem of libraries and frameworks, including:
    - **Data Science**: NumPy, pandas, Matplotlib, SciPy
    - **Web Development**: Django, Flask, FastAPI
    - **Machine Learning**: TensorFlow, PyTorch, scikit-learn
    - **Game Development**: Pygame
5. **Multi-Paradigm**: Supports various programming styles, including procedural, object-oriented, and functional programming.
6. **Cross-Platform**: Python can run on various operating systems like Windows, macOS, and Linux.

Common Uses

1. **Web Development**: Building server-side applications using frameworks like Django and Flask.
2. **Data Science and Analytics**: Analyzing data and creating visualizations with libraries like pandas and Matplotlib.
3. **Machine Learning and AI**: Developing algorithms and models with libraries such as TensorFlow and scikit-learn.
4. **Scripting and Automation**: Writing scripts to automate tasks and manipulate files.

5. **Game Development**: Creating games using libraries like Pygame.
6. **Internet of Things (IoT)**: Programming microcontrollers and devices using Python.

Community and Ecosystem

Python has a large and active community, contributing to a wealth of resources, documentation, and support. The Python Package Index (PyPI) hosts thousands of third-party packages that extend Python's capabilities.

Getting Started

1. **Installation**: You can download Python from the official website. It's available for Windows, macOS, and Linux.
2. **IDEs and Editors**: Common tools for writing Python code include:
   - **IDEs**: PyCharm, Visual Studio Code, Jupyter Notebook
   - **Text Editors**: Sublime Text, Atom
3. **Learning Resources**: There are numerous tutorials, online courses, and books available to help you learn Python. Some popular platforms include Codecademy, Coursera, and freeCodeCamp.

If you have specific questions or topics related to Python that you'd like to explore further, let me know!

CHAPTER- 4

CODE

```
def viterbi(observations, states, start_prob, trans_prob, emit_prob):
"""
Viterbi algorithm to find the most probable sequence of states given a
sequence of observations.
Parameters:
observations : tuple
A sequence of observed events.
states : tuple
A sequence of possible states.
start_prob : dict
The initial probabilities for each state.
trans_prob : dict
The transition probabilities from one state to another.
emit_prob : dict
The emission probabilities for each state to each observation.
Returns:
max_prob : float
The highest probability of the best state sequence.
best_path : list
The most probable sequence of states.
"""
# Validate input parameters
validate_input(observations, states, start_prob, trans_prob, emit_prob)
# Initialize the Viterbi matrix and the path tracker
V = [{}] # List of dictionaries to store the probabilities of the states at
each time step
path = {} # Dictionary to store the best path leading to each state
# Initialize the base cases (t == 0)
print("Initialization:")
for state in states:
# Probability of starting in state and emitting the first observation
V[0][state] = start_prob[state] * emit_prob[state][observations[0]]
# Initialize path for each state as starting with itself
path[state] = [state]
print(f"State: {state}, Probability: {V[0][state]}, Path: {path[state]}")
# Run Viterbi for t > 0 (for each observation after the first) 10
```

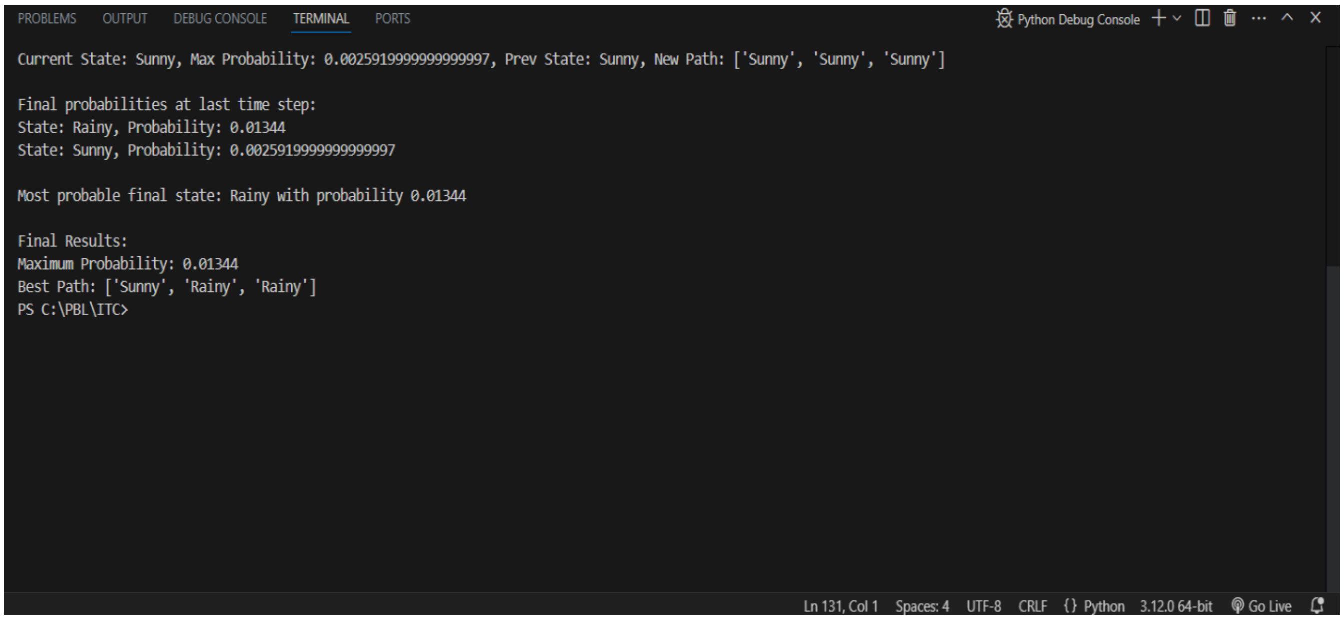
```
for t in range(1, len(observations)):
V.append({}) # Add a new layer for time step t
new_path = {} # Temporary dictionary to store new paths
print(f"\nTime step {t}: Observation = {observations[t]}")
# Loop through each state for the current time step
for current_state in states:
# Compute the maximum probability and the previous state that gives that
probability
max_prob, prev_st = max(
(V[t - 1][prev_state] * trans_prob[prev_state][current_state] *
emit_prob[current_state][observations[t]], prev_state)
for prev_state in states
)
# Set the max probability for the current state at time t
V[t][current_state] = max_prob
# Update the new path by adding the current state to the best path from the
previous state
new_path[current_state] = path[prev_st] + [current_state]
print(f"Current State: {current_state}, Max Probability: {max_prob}, Prev
State: {prev_st}, New Path: {new_path[current_state]}")
# Replace the old paths with the new ones for the next iteration
path = new_path
# Find the final most probable state and its probability
max_prob, final_state = max((V[-1][state], state) for state in states)
print("\nFinal probabilities at last time step:")
for state in states:
print(f"State: {state}, Probability: {V[-1][state]}")
print(f"\nMost probable final state: {final_state} with probability
{max_prob}")
# Return the highest probability and the best path (most probable sequence of
states)
return max_prob, path[final_state]
def validate_input(observations, states, start_prob, trans_prob, emit_prob):
"""
Validate the inputs to the Viterbi algorithm to ensure correctness.
Parameters: 11
```

```
observations : tuple
The sequence of observations.
states : tuple
The sequence of possible states.
start_prob : dict
The starting probability for each state.
trans_prob : dict
The state transition probabilities.
emit_prob : dict
The emission probabilities for each observation from each state.
"""
assert isinstance(observations, (list, tuple)), "Observations must be a list
or tuple."
assert isinstance(states, (list, tuple)), "States must be a list or tuple."
assert set(start_prob.keys()) == set(states), "Start probabilities must match
states."
assert set(trans_prob.keys()) == set(states), "Transition probabilities must
match states."
assert all(set(trans_prob[s].keys()) == set(states) for s in states),
"Transition probabilities must be defined for all states."
assert set(emit_prob.keys()) == set(states), "Emission probabilities must
match states."
assert all(set(emit_prob[s].keys()) == set(observations) for s in states),
"Emission probabilities must be defined for all observations."
print("Input validation passed.\n")
# Example usage
observations = ('walk', 'shop', 'clean')
states = ('Rainy', 'Sunny')
start_prob = {'Rainy': 0.6, 'Sunny': 0.4} # Probability of starting in each
state
trans_prob = {
'Rainy': {'Rainy': 0.7, 'Sunny': 0.3}, # Transition probabilities from Rainy
to other states
'Sunny': {'Rainy': 0.4, 'Sunny': 0.6}, # Transition probabilities from Sunny
to other states
}
emit_prob = {
'Rainy': {'walk': 0.1, 'shop': 0.4, 'clean': 0.5}, # Emission probabilities
for Rainy state
'Sunny': {'walk': 0.6, 'shop': 0.3, 'clean': 0.1}, # Emission probabilities
for Sunny state
} 12
```

```
# Run the Viterbi algorithm
print("Running Viterbi algorithm...\n")
max_prob, best_path = viterbi(observations, states, start_prob, trans_prob,
emit_prob)
# Output the results
print("\nFinal Results:")
print(f"Maximum Probability: {max_prob}")
print(f"Best Path: {best_path}")
```

CHAPTER-5

OUTPUT



```
Current State: Sunny, Max Probability: 0.0025919999999999997, Prev State: Sunny, New Path: ['Sunny', 'Sunny', 'Sunny']

Final probabilities at last time step:
State: Rainy, Probability: 0.01344
State: Sunny, Probability: 0.0025919999999999997

Most probable final state: Rainy with probability 0.01344

Final Results:
Maximum Probability: 0.01344
Best Path: ['Sunny', 'Rainy', 'Rainy']
PS C:\PBL\ITC>
```

CHAPTER-6

CONCLUSION

In summary, the Viterbi algorithm stands out as an essential tool in various fields, including natural language processing, bioinformatics, and communications. Its ability to efficiently compute the most probable state sequence allows for effective modeling of sequential data where the underlying processes are hidden. By utilizing dynamic programming, the algorithm reduces the computational complexity compared to naive approaches, making it feasible to analyze larger datasets and more complex models.

The algorithm's strengths lie in its clarity, robustness, and adaptability to different applications, such as speech recognition, gene prediction, and error correction in digital communications. However, it is important to note its reliance on the assumption of the Markov property, which may not hold in all real-world scenarios. Despite this limitation, the Viterbi algorithm remains a foundational technique in probabilistic modeling and continues to inspire advancements in related fields.

Overall, the Viterbi algorithm not only provides a method for sequence decoding but also enhances our understanding of stochastic processes, paving the way for further innovations in machine learning and data analysis. As technology evolves, the principles behind the Viterbi algorithm will likely continue to play a crucial role in the development of sophisticated algorithms for tackling complex, real-world problems.

.

OUTCOME

The outcome of the **Viterbi Algorithm** is the identification of the **most probable sequence of hidden states** in a **Hidden Markov Model (HMM)**, given a sequence of observed events. This result, also known as the **Viterbi path**, represents the most likely explanation for the observations based on the system's internal dynamics, such as transition and emission probabilities.

Key Outcomes:

1. **Most Probable State Sequence**: The algorithm produces the sequence of hidden states that best matches the given observed data.
2. **Maximum Likelihood**: It provides the maximum likelihood (probability) of this sequence, which shows how well the path explains the observed events.
3. **Efficient Computation**: By breaking down the problem using dynamic programming, the algorithm computes the most probable path efficiently, even for large state spaces and long sequences.
4. **State Transitions**: Along with the state sequence, the Viterbi Algorithm tracks the transitions between hidden states, giving insight into how the system evolves over time.

This outcome is used in various domains such as **speech recognition**, **bioinformatics**, and **natural language processing**, where the goal is to infer hidden structures or states from observed data.

COURSE OUTCOME(CO)

With this PBL CO4 is satisfied

CO-4 - To make students aware of various error control coding algorithms

REFERENCE

"Pattern Recognition and Machine Learning" by Christopher M. Bishop

GITHUB LINK