Artificial Intelligence course work

TASK ONE

1. **Why is this an important problem to solve?** For business been able to offer services and products at an efficient and fast speed is extremely important. It is the difference between a customer using your product again or that been the first and final time they use it. For example, a milk factory will need the milk that is been delivered to arrive promptly to hit weekly quota in what should be produced and packaged as well as to maintain the freshness of the milk, and this is similar in most businesses involved with food or quickly perishable goods. For the customer I would say that they are the most important when it comes to satisfaction, because without the customer there is no revenue and without revenue there is no profit. Majority of us have had an experience of a time they ordered something, and you were infuriated at how long it took to arrive and reach you. Environment sustainability? Urban environments are known for their high levels of traffic and commercial activities, already contributing to high levels of pollution and greenhouse gas emissions. Without optimized routes, delivery vehicles end up spending a lot more time idling in traffic and taking longer routes. This leads to

higher fuel consumption, increasing air pollution and contributing to climate change. By implementing optimized routing solution, businesses will be able to save on fuel costs and reduce emissions. This will also lead to fewer cars on the road, reducing congestion, contributing to liveability of the urban area. In extension with less vehicles on the road it will also reduce the strain on the roads helping in preserving infrastructure, safer roads due to less congestion, better public transport system. In summary, solving delivery inefficiencies benefits businesses, customers and cities, creating a more sustainable urban environment.

2. Can AI help solve this problem, and why? AI can play a critical role when it comes to solving such a problem. It can leverage both historical and real time data to enhance delivery efficiency. Businesses often collect large amounts of data, such as delivery times, routes taken, times of day and other relevant factors. How can AI come into play? AI can process this data to uncover patterns and train models capable of predicting outcomes, such as the travel cost and delivery time for each route given certain features. This predictive capability can be combined with optimization algorithms to find the shortest/optimal path that can be taken by the driver. For instance, it analyses trends to reduce delivery time and fuel

consumption while considering elevation and population density. It also can consider real time data meaning it can adjust to a dynamic environment as the conditions change throughout the day, so will the predicted outcomes and route adjustments will be made. These outcomes that will be predicted will be based of previous historical data and real time data that will continue to be collected as more orders are delivered. This will ensure that the model predictions keep on getting better as they is a wider range of data.

3. Is it a prediction problem, a search problem, or both? That's a good question. A prediction problem is one where models are trained on data to make predictions on what the value could be. In this situation to get the most optimal path it would require knowledge on the travel cost. With a prediction model considering all the factors that could influence the travel cost it would be able to give a prediction on what it thinks the travel cost would be from one city to another. This fits into the nature of prediction problems. After the travel costs have been obtained a search algorithm should be used to find the most optimal path the driver should take.

4. If it is a prediction problem, is it regression or a classification problem? Okay what makes you decide between using regression or classification to train the model is the output expected. Regression is

good for numeric or continuous value while Classification is good for discrete or categorical predictions. So, for example if I wanted to predict the weather for tomorrow if I wanted the output to be an exact value such as “25 °C”, regression would be better suited for this problem. But if I wanted the output to be “sunny” for example this problem would suit classification more. Travel cost is a numeric value so Regression would be the best way to handle this problem. After establishing that regression would be the best way to handle this problem then the next step it to decide between using linear regression or polynomial regression to train the model. Linear regression is best suited when the factors and the outcome are linearly dependent on one another, and it also works best when there is only one feature/factor that influences the outcome. Polynomial regression is much more suited for problems with many features/factors and the relationships are nonlinear. In this problem, features like traffic speed, elevation and weather conditions interact in complex ways, making polynomial regression a better choice. For example, traffic speed may have a non-linear relationship with travel cost, as the impact of congestion varies during the day for example during rush hours.

TASK TWO

Data preprocessing:

Handling missing values

- I loaded the csv file into my Jupyter notebook.
- I first checked the general information of the data from the number of columns to the rows and the specific data types of the columns.
- I checked for missing values; I did not find any missing value although I was incredibly confused because in Canvas when you look at the data manually some values have the value "########" so, I assumed these were missing values. However, when trying to check for them on notebook to replace them with the mean for the numerical values and mode for the three categorical values, I would print the number of missing values, and I would get 0 missing values in all columns.

Encoding values

- I then moved on onto encoding my data.
- First, I looked for non- numeric features in the data set, there are three non-numeric features:
    - Type of terrain
    - Zone classification

    - Time of day
- I encoded these categorical features using the One hot encoding technique over using Ordinal encoding because there is no ordinal relationship between these values meaning there is no logical ranking from low, medium, high for example. So, I thought this the best way to handle encoding.

Scaling

- I standardized the data because features like elevation and population density might have different ranges. Without scaling, the model might disproportionately weigh larger values as more important.
- I only standardized the features because, if I standardized the target value (travel_cost) as well it would lead to confusion when testing and having to revert the output back to its original scale.

Data Splitting

- A split of 80% for training data and 20% for test data was observed.
- Also, I introduced **random_state**, without this **train_test_split** will always select random rows to go into the training set or testing set. This will lead to slightly different model performance each time I train the models.

- It ensures that the splitting of data is reproducible selecting the same rows for training and testing every time the code is run.
- This will ensure a **fair comparison** is done between all the different models so that I can derive accurate conclusions.

Linear Regression

- Results breakdown:
    - Mean Absolute Error (MAE) – 124.38
    - Mean Square Error (MSE) – 27,449.40
    - Root Mean Squared Error (RMSE) – 165.68
- According to the MAE, on average the model's predictions are off by 124.38 units (same units as the travel_cost). A lower MAE shows better performance.
- The MSE represents the mean square error between actual and predicted values. Large errors will be penalized more because of squaring. The lower the MSE the better the model performance.
- The RMSE is the square root of the MSE. The difference between MAE and RMSE, suggests the presence of outliers or large prediction errors that cause the value of RMSE to spike.

Polynomial Regression

- Results breakdown:
    - Mean Absolute Error (MAE) – 0.4619
    - Mean Square Error (MSE) – 0.3375
    - Root Mean Squared Error (RMSE) – 0.5810
- The MAE, MSE and RMSE is much lower than that of the linear regression this shows that polynomial regression captures the relationship between features and output more accurately.
- The **degree- 4 polynomial** provides enough flexibility to capture non-linear relationships between features and the target variable while avoiding overfitting.

Basic Neural Network

- I designed a network with two hidden layers (32 and 16 neurons) and an output layer for regression. The architecture was chosen to balance learning capacity and simplicity for a basic neural network
- Activation functions (ReLU for hidden layers, Linear for the output layer) were chosen to enable the network to learn non-linear patterns while predicting continuous values for **travel_cost.**
- The Adam optimizer was chosen for its adaptive learning rate and efficient convergence.

Activation Functions

- ReLU was used in the hidden layers for its simplicity and efficiency, allowing the model to learn non-linear relationships without saturation issues like those of sigmoid and tanh.
- Linear activation function for the output layer to handle the regression task, as it allows the model to predict continuous values.

Optimizer

- The Adam optimizer was chosen for its adaptive learning rate and efficient convergence.

Training details

1. Epochs:

- The model was trained for 50 epochs with a batch size of 32 to allow sufficient learning while avoiding overfitting. This number was decided based on observed convergence during training where the **loss rapidly decreased initially and then stabilized**. It is also a good number for a basic neural net.

2. Validation Split:

- I used 20% validation split to monitor the model's performance as it trains.
- The validation loss closely followed the training loss, confirming that the model achieved a good

balance between learning from the training data and generalizing to unseen data. Showing that it was neither underfitting nor overfitting.

- Results breakdown:
    - Mean Absolute Error (MAE) – 67.60
    - Mean Square Error (MSE) – 7897.44
    - Root Mean Squared Error (RMSE) – 88.87

Model comparison and Analysis

- The best Model: Polynomial regression
    - This is the clear winner for this dataset. It captures the data's complexity with low error values across all metrics.
    - It successfully modelled the non-linear relationships in the dataset. Showing no signs of underfitting or overfitting.
- The worst Model: Linear Regression
    - It is the worst, it underfits the data unable to learn efficiently due to its simplicity and inability to handle non-linear relationships.
- Moderate performance: Basic neural network
    - It outperformed linear regression but did not achieve the accuracy of polynomial

regression. This may be due to limited data size or insufficient hyperparameter tuning.

Reason behind model performance

Why Polynomial Regression outperformed the rest:

- **Model complexity:**
  The degree-4 polynomial regression model has enough flexibility to learn the non-linear relationships, making it well suited for this dataset.
- **Data size:**
  Polynomial regression doesn't need that large of dataset to perform well
- **Feature importance:**
  Polynomial regression naturally captures interactions between features through its polynomial terms, enabling it to model the relationships effectively.

Why Linear regression Underperformed:

- **Model simplicity:**
  Linear regression assumes a linear relationship between the features and target. This makes it unsuitable for datasets with non-linear relationships like this one.
- **Underfitting:**

Unable to capture the complexity of the dataset, resulting in large errors and poor predictions.

Why the Neural Network Was Moderate:

- **Limitations:**
  1. **Data size –** Neural networks typically perform better with larger datasets due to their ability to model complex patterns.
  2. **Basic Architecture –** The two hidden layers of (32 neurons and 16 neurons) were sufficient for a "basic" model but it was unable to completely capture the relationships in the data.

Summary:

Polynomial regression performed the best, followed by the basic neural network and then finally linear regression.

TASK THREE

1. Predicting traversal Costs

- Using the **polynomial regression model** trained in Task 2, I predicted the traversal costs for each cell in the provided grid dataset (provided_grid.csv).

- The **best-performing model** (polynomial regression, degree 4) was applied to estimate traversal costs for the unseen data.
- The predictions were added as a new column, **predicted_traversal_cost**, to the dataset and saved as **Estimated_grid.csv**.

2. <u>Converting it into a graph</u>

- To enable advanced search algorithms like Dijkstra's, the dataset was transformed into a **2D grid structure**, where:
  - Each **cell** corresponds to a **node** in the graph.
  - Connections (edges) exist between **adjacent cells** (up, down, left, right).
  - The **weight** of each edge is defined by the **predicted_traversal_cost** of the destination cell.
- This transformation allows the traversal costs to be treated as weights in a graph, making the data suitable for pathfinding algorithms.

3. **<u>Implementing Dijkstra's Algorithm</u>**

- **Dijkstra's algorithm** was implemented to find the least-cost path in the weighted grid. The algorithm systematically explores nodes to guarantee the shortest path.
- **Start Node**: Top-left corner of the grid (cell **(0, 0)**).
- **End Node**: Bottom-right corner of the grid (cell **(grid_size-1, grid_size-1)**).
- **Results**:

- **Least-Cost Path**: The algorithm identified the sequence of nodes with the lowest traversal cost from (0, 0) to (19, 19).
- **Total Cost**: The traversal cost for this path was **13,105.76 units**.
- The path is as follows:
  - Least-cost path: [(0, 0), (1, 0), (1, 1), (1, 2), (2, 2), (2, 3), (3, 3), (4, 3), (4, 4), (4, 5), (5, 5), (6, 5), (7, 5), (7, 6), (8, 6), (9, 6), (9, 7), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (13, 9), (14, 9), (14, 10), (14, 11), (15, 11), (15, 12), (16, 12), (16, 13), (17, 13), (17, 14), (18, 14), (18, 15), (18, 16), (18, 17), (18, 18), (18, 19), (19, 19)]

Algorithm Comparison

1. Depth First Search

How it works:

- DFS explores as far as possible along one branch before back tracking.
- It uses a **stack to track visited nodes.**
- DFS does not consider weight and is not designed to find the **shortest or most optimal path.**

Advantages

- Simple and easy to implement.
- Memory-efficient

Disadvantages

- Does not guarantee the shortest path in a weighted path.

- Will explore unnecessary paths, especially in a grid with many branches, making it ineffective at optimal path finding.

DFS in weighted graphs

- In the context of finding the most optimal path in a weighted graph, it is not a good option because it doesn't account for weights.
- It might find a path, but it won't be the shortest or most optimal.

2. Breadth first search

How it works

- BFS will explore all nodes at the current depth before moving to the next level.
- It uses a **queue** to track visited nodes.
- BFS can work well in **unweighted graphs,** it can guarantee the shortest path in terms of number of edges.

Advantages

- Systematic exploration
- Guarantees the shortest path in terms of number of edges.

Disadvantages

- Memory intensive because it stores all nodes at the current level, which can be big in large grid.
- **Inefficient for weighted graphs** and would not be able find the optimal delivery path because it **doesn't account for weights.**

BFS in weighted graphs

- It treats all edges as having equal weight, it will not optimise for transversal costs.

3. Dijkstra's algorithm

How it works

- Dijkstra's algorithm finds the shortest path from a starting node to all other nodes in a graph.
- It uses a **priority queue** to explore nodes with the smallest known distance first.
- Accounts for **weights on edges** making it good for weighted graphs.

Advantages

- It is good because it guarantees the shortest path in weighted graphs.
- Works for both directed and undirected graphs.

Disadvantages

- Memory usage can be an issue as it stores distances and visited nodes, which will be large if the grid is large.
- Can be slow because it processes all nodes, even if they are not the optimal path.

Dijkstra in weighted graph

- It is highly suitable for this task as it considers the traversal cost(weights) of each cell and guarantees the most efficient delivery path.

4. **A* Search algorithm**

How it works

- It is a variation of Dijkstra's algorithm that uses a **heuristic** to guide the search. A heuristic is a function that estimates the cost to reach the goal from a given node.

Advantages

- The heuristic helps prioritize paths that are likely to lead to the destination faster making it more efficient than Dijkstra's.

Disadvantages

- Its effectiveness depends on the quality of the heuristic used. A poorly designed heuristic could lead to worse performance than Dijkstra's.

A* in weighted graph

- If a good heuristic can be designed, then it's a good option for this task.
- It combines Dijkstra's accuracy with added efficiency, making it well-suited for large grids or when faster computation is needed.

Conclusion: Best algorithm for this task – It is Dijkstra's because it guarantees the shortest path in weighted graph. A* could also be suitable but it depends on having a well-designed heuristic.

TASK FOUR

Implementation of Q-Learning

1. **Problem setup**
   - The grid world was represented as a 5x5 matrix:
     - **Start Position**: (1, 1) (adjusted to (0, 0) in 0-based indexing).
     - **Goal Position (Treasure)**: (5, 5) (adjusted to (4, 4)).
     - **Obstacles**: Located at (2, 2) and (4, 4) (adjusted to (1, 1) and (3, 3)).
     - Each cell represents a **state**, and the agent can take one of four **actions**: **up, down, left**, or **right**.
     - Rewards:
       - +1 for moving into an empty cell.
       - -10 for hitting an obstacle.
       - +100 for reaching the goal.
2. **Q-Learning algorithm implementation**
   - **Q-Table Initialization**:

    - Q-table of shape (**5, 5, 4**) is initialized with zeros. It stores the value of each action at every state.
- **Hyperparameters**:
    - **Learning Rate** : 0.1. It controls how much the agent learns from new experiences.
    - **Discount Factor** : 0.9 balances the importance of immediate and future rewards.
    - **Exploration Rate** : 0.1 ensures exploration by taking random actions 10% of the time.
- **Training Process**:
    - The agent was trained for 1000 episodes.
    - In each episode:
        - The agent started at (0, 0).
        - A greedy policy was used to select actions.
        - The Q-value for the current state-action pair was updated using the Q-learning formula.
        - The episode ended when the agent reached the goal or exceeded the maximum number of steps.
- **Evaluation**:
    - After training, the agent was tested on 100 episodes without exploration .
    - Average reward obtained: **100.0**, indicating that the agent consistently navigates to the goal efficiently.

3. Results

Learned Policy:
[['u' 'u' 'u' 'u' 'u']
['u' 'u' 'u' 'u' 'u']
['u' 'd' 'u' 'u' 'u']
['d' 'd' 'u' 'u' 'u']
['u' 'u' 'u' 'u' 'u']]

- **Key:**
  - **U – move up**
  - **d – move down**
  - **l – move left**
  - **r – move right**
- The agent successfully navigated reaching the goal while avoiding penalties, demonstrating it had learnt an efficient navigation strategy.

4. Analysis of hyperparameters

- **Learning rate**
  - A smaller learning rate(0.01) made learning slower as updates were minimal
  - A large learning rate(0.5) caused the agent to overwrite old knowledge too quickly, leading to inconsistent and unstable learning.
- **Discount factor**
  - Lower discount values(0.5) made the agent more focused on immediate rewards and miss long-term rewards.
  - Higher values(0.95) made the agent balance immediate and future rewards.
- **Exploration rate**

- Higher exploration rate(0.3) improved exploration but increased training time.
- Lower exploration rate(0.01) caused premature conclusions to non-optimal policies.

Conclusion:

- It was able to learn how to navigate the grid world, after being trained by the Q-learning algorithm.
- The learned policy achieved the goal while avoiding obstacles.
- Hyperparameter tuning was important to ensure it learnt effectively.