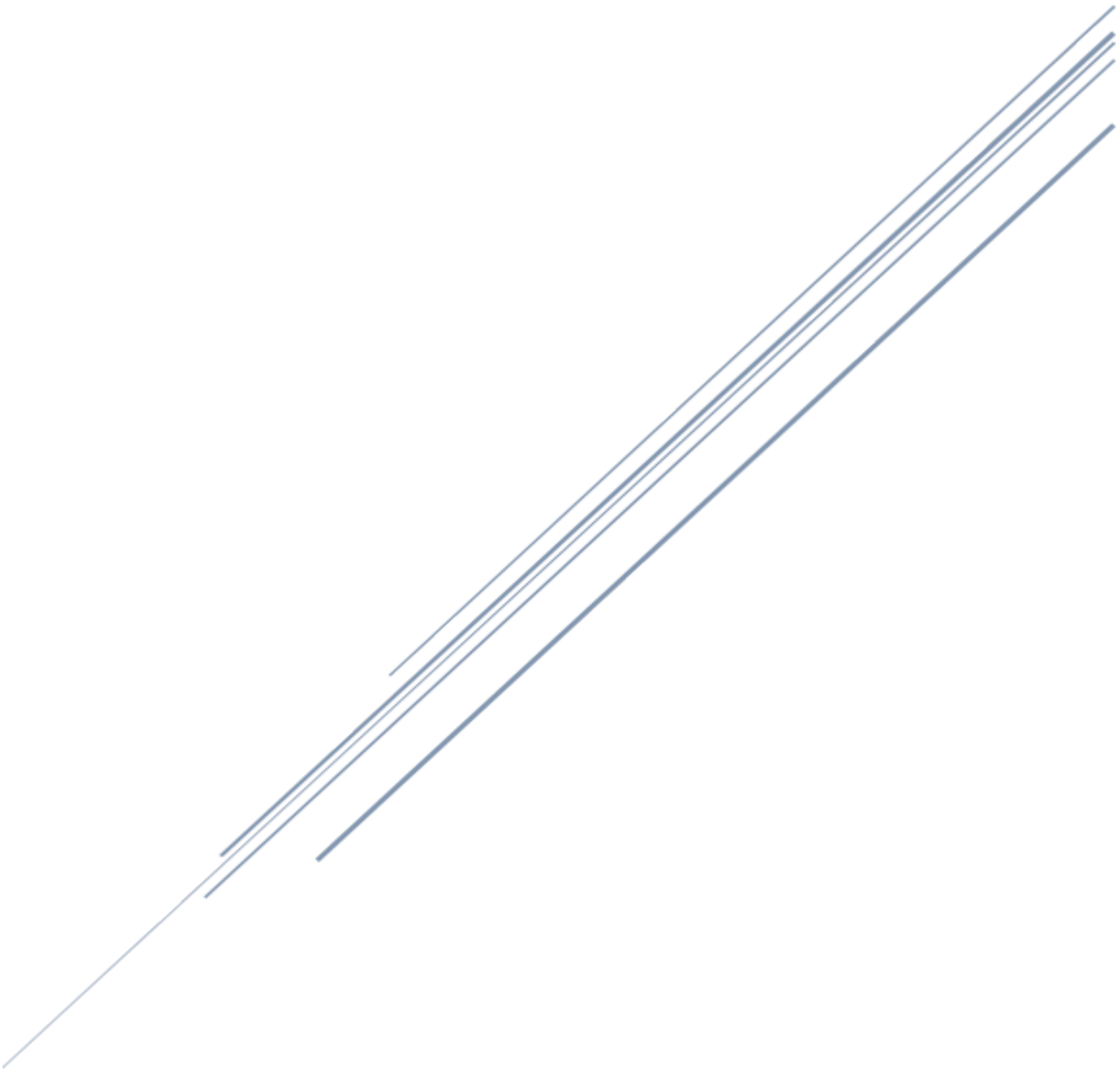
AI PROJECT 3

Name of students:
Chakradhar Reddy Punur (crp190),
Mohammed Najeebuddin Quadri (mq166).



Rutgers University-New Brunswick.
Introduction to AI (CS520).

Table of contents

Introduction

In project 1, the goal was to localize the bot on a ship. Initially, the bot could be in any open cell, which is represented as an initial belief. Various actions allowed the bot to move in the ship, and consequently, the goal is to shrink the belief to a single cell, essentially localizing the bot. This problem was solved using A* search optimally.

A* search guarantees optimality if the heuristic is admissible. However, the belief space for A* is expensive because belief states can be large up to all open cells, and many belief configurations have to be explored.

The heuristic used for project 1 is the Manhattan distance. This is simple to compute and is admissible, but it has limitations. The main limitation is that it ignores the details of the map structure, including walls, and doesn't account for the internal geometry of the belief sets.

In this project, the idea is to use a heuristic from a trained machine learning model that is trained on (belief state, number of moves). If this closely approximates the true cost, then it can reduce the search effort in producing optimal or sub-optimal solutions for the problem.

The key challenge is overfitting to one ship size, which happens if the model is trained on a fixed ship layout. To address this, we use a neural network to train different ship sizes along with the belief space. This allows the model to predict the number of moves across various ship layouts.

Data Generation Pipeline

1. Generating initial belief states using BFS:

For a particular ship layout, the belief states are generated using BFS over the belief space. Initially, the belief space consists of all open cells representing uncertainty.

From each belief state, actions that include UP, DOWN, LEFT, and RIGHT are applied to produce the next state. The visited set is also used to avoid duplicate beliefs.

This procedure results in a wide variety of belief states that are reachable with random motion.

2. Filtering the states from BFS with A*:

The BFS states generated previously may not include the belief in the later stages. For this, we run the A* strategy for each state from the BFS states, and all the states that are encountered in the optimal path are also considered for the dataset.

3. Constructing Input Output Pairs:

Each belief state L is paired with its true cost from A* strategy to form the training examples.

The input X encodes the belief state into fixed-size vectors, and the output Y is the optimal number of steps to localize for that belief state.

The final dataset consists of 4924 pairs of (X, Y) labels. In each label, the X length is 45, which denotes the total number of open cells, and Y is the number of moves, which ranges from 0 for full localization and 20, which is the maximum number of moves.

Machine Learning Models

1. Problem Formulation:

The learning task is formulated as a supervised regression model:

$F(X) = Y$

The goal is to learn a function that can accurately predict the number of moves and also be sufficient to be used as a heuristic for A* search.

2. Model choice based on Input and Output Structure:

The output in this case is a single scalar value representing the optimal number of steps to localize the bot in a given belief state. As a result, the task becomes a supervised regression problem rather than classification.

3. Linear Regression:

Linear Regression is perhaps the best model that can be used for this problem. It is simple to use, easy to interpret, and provides a baseline, but it can not capture complex interactions between belief configurations.

4. XGBoost:

Other possible choices include, but are not limited to, tree-based regression models. Gradient boosted decision trees (XGBoost), Random Forests can efficiently model non-linear interactions between features.

XGBoost works really well in this problem, achieving high accuracy. It works well because the heuristic problem is low-dimensional after encoding, and the target function is piecewise simple.

5. Loss function for evaluating the model:

The model is predicting a single numerical value; this is a regression problem. So the natural choice is MSE (Mean Squared Error).

In our evaluation, we have also computed MAE and **R^2,** which are very useful to our report along with MSE. MAE is more human interpretable, while **R^2** gives us an overall goodness-of-fit score.

6. How the model is trained and possible issues:

The training data is generated by exploration of belief states and labelling them using the A* algorithm. Each belief state is paired with its true remaining cost, computed by running A* from that particular state.

There were several challenges that arise during the training. First, label generation is expensive, since each label requires the A* to be running, so the dataset size must be very carefully bounded. Secondly, the dataset may be biased towards easier or optimal path states because belief states are collected mainly from successful A* trajectories. Third, bounding A* expansions can exclude very difficult states, which may limit robustness. These factors must be acknowledged when interpreting the results.

7. Avoiding Overfitting:

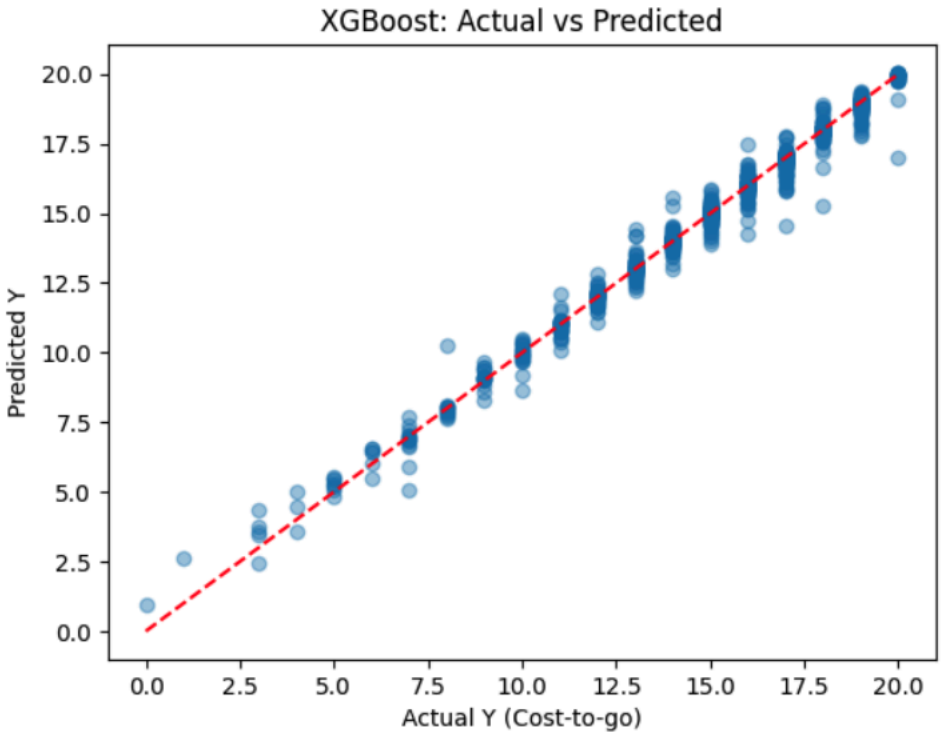
Linear regression exhibits a low risk of overfitting in the single-ship setting. This is because, due to its limited expressive capacity, the model can only learn a global linear relationship between belief occupancy features and the remaining localization cost, which prevents it from memorizing specific belief configurations. As a result, the training and validation errors are similar, indicating stable but constrained learning behavior. However, this limited capacity can also lead to underfitting. Linear regression fails to capture the spatial

structure and geometry of belief states, such as connectivity, shape, and interactions between belief regions.

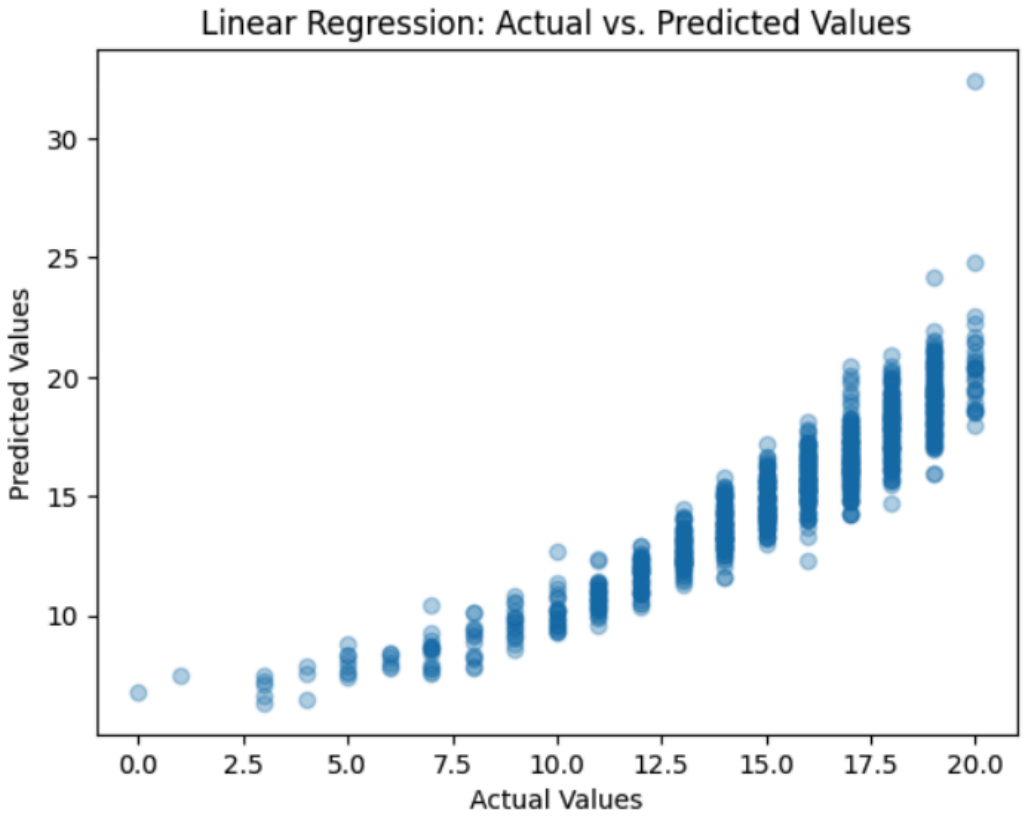
In contrast, XGBoost demonstrates very strong performance on a single ship layout, achieving extremely low error and a high R^2. This is due to its high capacity and ability to learn complex, non-linear decision rules based on combinations of belief features. In the single-ship setting, these features have fixed semantic meaning, which allows the model to accurately predict the cost-to-go function.

At the same time, this expressive power makes XGBoost prone to overfitting. The model can learn layout-specific rules that are tied to absolute cell identities rather than general properties of belief states. This overfitting is not immediately apparent when evaluation is performed on held-out data from the same ship, but it becomes evident when the model is applied to unseen ship layouts, where performance degrades significantly.

8. Analysis:



Each point represents one belief state with the x-axis showing the true optimal cost and the y-axis showing the model's predicted cost. The red diagonal represents perfect predictions. Most points lie close to this line, indicating that XGBoost predicts the optimal cost with high accuracy and low error.



The linear regression plot above, on the other hand, shows more scatter when compared to XGBoost, especially at higher cost values. So, linear regression tends to struggle in complex belief configurations. The model learns a coarse linear relationship but can not fully capture the non-linear structure of the cost function.

A* search using ML heuristic

1. Decrease in node expansions:

The number of belief states expanded by A* is the most straightforward and significant metric. Both the learned ML heuristic from XGBoost and a conventional distance-based heuristic are used to run the A*. We can determine whether the model directs the search more successfully by comparing the number of expansions needed in each scenario. An effective heuristic should drastically cut down on expansions, so that A* investigates fewer states before achieving the objective.

2. Runtime and efficiency of searches:

Runtime is closely associated with node expansions. The learned heuristic is more beneficial if it shortens the search's overall computation time, even if both heuristics identify the best answers. Both fewer expansions and improved priority queue prioritization are part of this. A useful evaluation of real-world benefit can be obtained by measuring wall-clock runtime or average planning time over several runs.

3. Correctness and quality of the solution:

It is crucial to confirm that applying the learned heuristic does not result in a lower-quality solution. The optimal solution discovered using a typical admissible heuristic should coincide with the final solution length (number of actions). By comparing path lengths between heuristics, the model is guaranteed to increase efficiency without compromising accuracy.

4. Search performance versus heuristic accuracy:

Although useful, prediction metrics like MSE and MAE are insufficient on their own. If a heuristic ranks states well, it may still improve

search performance even with a slightly higher prediction error. Therefore, rather than depending only on regression metrics, utility should be assessed by linking prediction quality with real reductions in expansions and runtime.

5. Sturdiness in a variety of belief states:

It is important to test the model's usefulness in a range of belief states, from large, highly uncertain belief sets to smaller, more constrained ones. A good heuristic should guide search well in all of these situations, not just close to the objective or in simple situations.

6. Analysis:

The XGBoost model trained previously is used as a pickle file as the heuristic for A*. It is then compared with A* with Manhattan Distance for different belief states, and the results show that both give the same optimal solutions, proving that this heuristic is good enough for A* to provide optimal cost.

The main difference comes from the search efficiency. Compared to the Manhattan distance heuristic, A* search with ML-based heuristic expands far fewer states while still achieving optimal or near-optimal solutions. The model has learned how the belief states shrink over time and how the ship’s layout affects the process.

The utility of the model is evaluated by how much it reduces A* node expansions, improves runtime, and generalizes across environments discussed in the next section.

Generalization

1. Data Generation:

For a single ship, the belief states are encoded as vectors indexed by open_cells, but for multiple ship layouts, the belief states are encoded along with the ship size as a 2-channel spatial grid. Channel 1 consists of the binary map of open and closed cells, and channel 2 consists of a binary belief map of possible bot locations.

The training dataset is constructed by combining the belief states from multiple ship layouts. For each ship, a diverse set of initial belief states is generated using BFS. Each belief state is then processed using bounded A* search to compute the optimal path from that belief state. An expansion limit is also added to reduce run time.

All belief states that are encountered on the optimal path are also added to the dataset. Each belief state is then encoded jointly with its corresponding ship layout as a 2-channel spatial grid.

2. CNN Architecture, Training, and Evaluation:

The dataset is converted into Pytorch tensors and reshaped into 2-channel spatial grids. The data is then split into training and validation sets and loaded using min-batch DataLoaders for CNN training.

The CNN architecture we created consists of two convolutional blocks, each containing 2 (3 * 3) convolutional layers with ReLU activations, max pooling and fully connected layers.

The CNN is trained using the Adam optimizer with a learning rate of 0.001 and Mean Squared Error (MSE) as the loss function. MSE is appropriate for this task since the model performs regression, predicting the remaining number of localization steps from a belief state. Training is conducted for 20 epochs, and the network is

updated using mini-batch gradient descent, allowing for stable and efficient optimization of model parameters.

During each epoch, the model processes the training data in shuffled mini-batches. For each batch, predictions are computed through a forward pass, the loss is evaluated against the ground-truth cost-to-go values, and gradients are backpropagated to update the model weights. The average training loss per epoch is reported to monitor convergence, with the observed decreasing trend indicating successful learning of the belief-state heuristic.

After training, the model is evaluated on a held-out validation set to assess its predictive accuracy and generalization performance. The CNN achieves an MAE of 0.594 and an R^2 of 0.9741, indicating that predicted localization costs are typically within less than one action of the optimal value and that the model explains over 97% of the variance in the true cost-to-go values. These results demonstrate that the learned heuristic provides an accurate and reliable approximation of the belief-space cost function across multiple ship layouts.

3. Overfitting:

Overfitting happens when the network learns to memorize the training examples instead of learning general patterns. The model performs very well on the ship layouts it was trained on, but its predictions become inaccurate when it is tested on new, unseen layouts. This can be prevented by making sure the model learns general patterns and using validation data to monitor performance on unseen data.

Overall, rather than learning layout-specific patterns, the learned heuristic can capture environment-invariant properties of belief states by training across multiple ship layouts. The model learns how belief structure and environmental constraints affect localization

difficulty by simultaneously encoding belief configurations and map geometry. According to experimental results, this method consistently directs belief-space search effectively and produces stable performance across a variety of ship layouts, exhibiting successful generalization to previously unseen environments.

Collaboration Note

This work was done collaboratively by both Chakradhar Reddy Punur and Mohammed Najeebuddin Quadri. We jointly developed the code, performed the analysis, ran experiments, and wrote the report.