MLDM Coursework: *Group Alchemy*

Fin Campbell-Brady FC00617@SURREY.AC.UK
Harsh Marthak HM01626@SURREY.AC.UK
Harshita Wardhan HW01458@SURREY.AC.UK
Luke Menezes LM01906@SURREY.AC.UK
Meenu Madhavan Pillai Sasidharan Nair MM03943@SURREY.AC.UK

Abstract

This study explores two datasets: the first one relates to real estate data, including features like bathrooms, bedrooms, square footage, latitude, longitude, and various amenities, to predict property prices. The second dataset involves socio-economic and demographic data, such as age, education, work hours, and income, to classify income levels. Various machine learning algorithms, including XGBoost, KNN, SVM, ILP, MLP, decision tree, and Q-learning, were employed to analyse these datasets. The models were evaluated using metrics such as Mean Squared Error (MSE), R², Mean Absolute Error (MAE), accuracy, precision, recall, F1-score, and ROC curve.

1. Project Definition

The main objective of our project was to compare the performance of machine learning algorithms from different paradigms on the same datasets. For a complete understanding, two datasets were chosen, one for classification and one for regression. By the conclusion, we hope to have a greater understanding of which types of learning algorithms are suitable for each kind of task.

To explore our objectives, we have run two experiments, where we ran our seven different learning algorithms on each of the datasets and compared the results. To ensure fair results and comparisons, the pre-processing has been kept as similar as possible for all the implementations and the evaluation metrics are the same.

Datasets: The UCI Adult data set, also known as "Census Income," is derived from a 1994 Census database and has information regarding age, work class, education, marital status, occupation, relationship, race, sex, capital gain, capital loss, hours per week, and native country. This dataset has a good mix of categorical, discrete, and continuous data. This data set contains 48,842 instances. The target is the binary income level, either less than or equal to 50K or more than 50K (binary classification).

The UCI "Apartment for Rent Classified" dataset contains information about apartment rental listings in the USA. It includes 10,000 rows with 22 features, it can be used for various machine learning tasks such as classification, regression, and clustering. Most useful features are numerical with a few categorical. The target variable is the price of rent for each apartment, which makes this a regression task.

The seven algorithms we have chosen are: decision trees, support vector machines (SVM), Inductive logic programming (ILP) with Pygol, XGBoost, K-nearest neighbours (K-NN), Multi-layer perceptron and reinforcement learning.

The evaluation metrics for classification are confusion matrix, F1 score and receiver operating characteristic (ROC) curve. For regression, we have chosen R^2 score, mean squared error (MSE) and actual vs predicted plots.

2. Data Preparation

2.1 Data cleaning and integration

Rent Dataset:

Data dictionary description: There are 22 features in the dataset, with 7 being integer features and 15 categorical. The categorical features included information about each apartment, such as the amenities, city and square feet. The integer features held information about the listing of the apartments, like price and location. The dataset does not come with a pre-set target variable.

During initial exploration, missing values were found in amenities (3549 instances), bathrooms (34 instances), bedrooms (7 instances), pets_allowed (4163 instances), address (3327 instances), city name (77 instances), state (77 instances), latitude (10 instances), longitude (10 instances).

Features body and title were dropped because information about them was contained in others. Features Currency, Price type, Category, and Fee were dropped because they all contributed just the one value. ‘id’ was dropped as it does not offer useful information, and address was dropped in favour of latitude and longitude, as they are easier for ML to use as they are just numbers and makes it easier to compare different locations. The address column was also dropped in favour of using latitude and longitude, as numerical coordinates simplify the processing for machine learning applications and facilitate location comparisons by providing a uniform data format. Dropping those columns helped with the missing values problem, for bedrooms and bathrooms we used the square feet and number of the other to predict what they maybe then impute the values. For longitude and latitude there were only 10 missing columns so we decided it would just be easier to drop them.

Adult Dataset:

Initial exploration revealed that the work class (963), occupation (966), and native-country (274) columns had missing values. It was imputed with logistic regression. Unique values and their counts were checked for each categorical column; therefore, the income column had 4 instead of 2 values in that column, as the full stops were not removed. So, this column was put through a cleaning operation to remove the full stops and make it two values.

2.2 Variable transformation

Rent Database:

Outliers were dealt with by calculating z-scores of price, square feet, and time columns and applying them. The features that were scaled included price, square feet, latitude, longitude, and time. After removing the outliers, the data quality was then checked using histograms Figure. The column amenities, which listed the amenities of each apartment, was cleaned and tokenized. The frequency of each of those amenities was counted, and logistic regression was used to rank their importance; finally, only the top 10 were taken into consideration. These features were then one-hot encoded. The columns has_photo, pets allowed, and source were one-hot encoded to convert categorical data into numerical format.

Adult Data Set:

Data dictionary description: There are 13 features and one target variable in the dataset. The target variable is stored as binary, while the features are a mix of integer, categorical and binary. There are several features with missing values. Most of the features are typical of censuses, with the categorical features describing things like education level and working class level, while the integer features describe hours worked.

Work class, marital-status, occupation, relationship, and race features are one-hot encoded. Sex, native-country, and income features are encoded as labels. The education feature is grouped with the feature relationship, to reduce the number of columns. Outlier capping of numerical columns (age, fnlgwt, capital-gain, capital-loss, hours-per-week) using Z-scores, with mean of 0 and standard deviation of 1.

Exploration showed significant class imbalances, with 37,155 instances of <=50K and 11,687 instances of >50K in the income column. This imbalance was addressed by down sampling the majority class (<=50K) to match the number of instances in the minority class (>50K), resulting in an equal representation of both classes. Down sampling was chosen over oversampling to maintain data quality and integrity and ensure faster training times with lower computational costs which is important as resources are limited. The <=50K distribution was then visualised before and after down sampling to make sure its structure hadn't been affected.

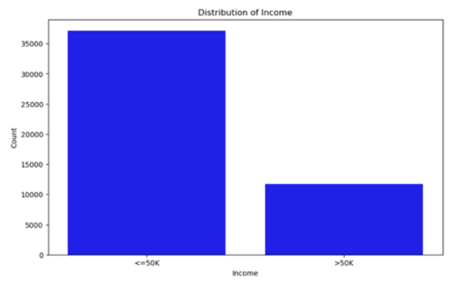
2.3 Data visualisation

Various graphs were produced to visualise the various relationships between variables in the datasets. The Python libraries Matplotlib and Seaborn were used to produce these visuals.

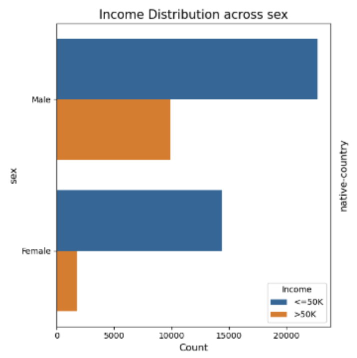
Adult dataset

Bar graphs were produced for this dataset's 10 categorical variables. With the space limit, the most interesting findings will be shown.

Income (target variable)

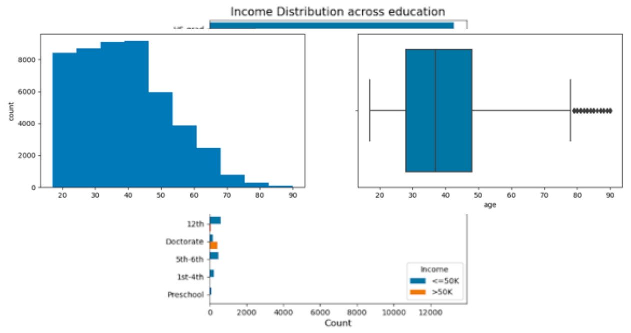


As seen, the target variable is unbalanced, with about 20,000 more entries above $50,000 than below it.

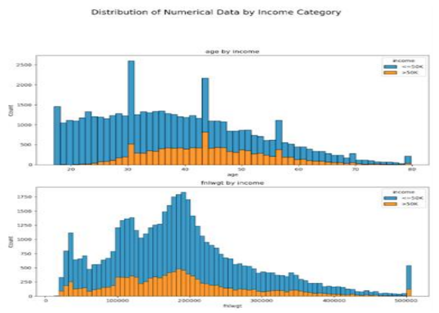


This shows that the male census responders were much more likely to earn over $50,000.

As expected, census responders with higher education levels tended to earn above $50,000 more often.



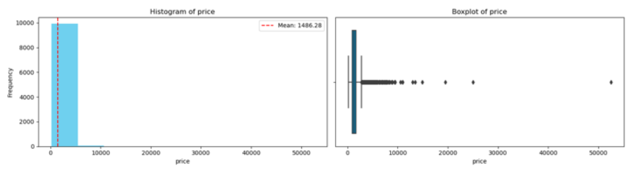
As expected from a general census, most respondents' ages are between 30 and 50, with the outliers above 75.



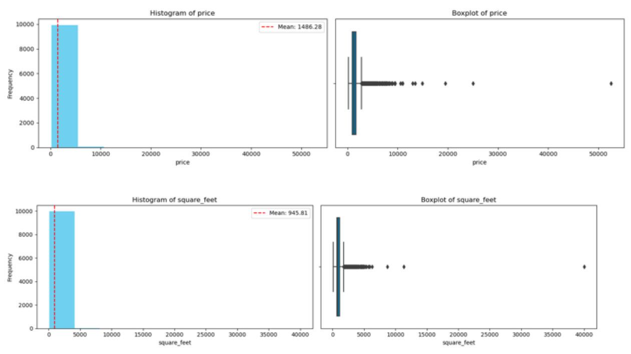
Most responders earning over $50,000 are middle-aged, between 30 and 60, and tend to be more common when their area's population is around 200,000.

As seen, this dataset is not very co-correlated, with no correlation ratios over 0.15 found.

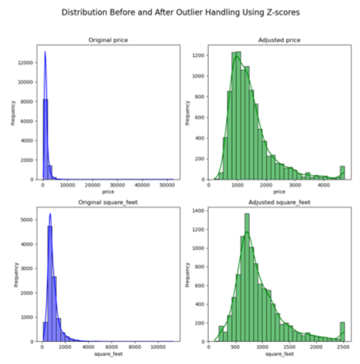
Rent dataset.



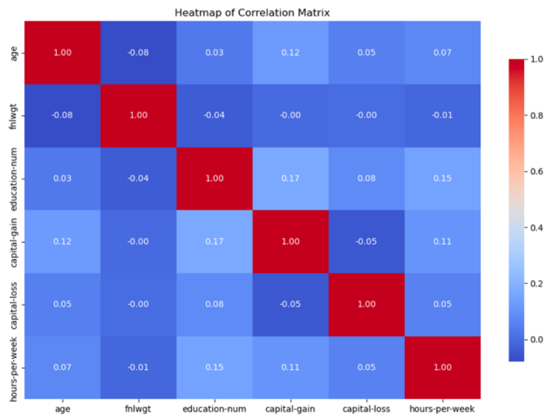
Both price and square feet are heavily concentrated at lower values, suggesting there is not much distribution in these variables.



To address this, outlier handling was used, which resulted in much better distributions for price and square feet.



Some variables do have high levels of co-correlation, like



bedroom and bathrooms, but this is to be expected and should not impact their predictive power.

3.Model development.

We used several models and compared their performances. Here, we describe each method, provide justification for their use, and discuss the pros and cons.

Model 1: Decision Tree

A Decision Tree is a non-linear predictive model that splits the data into subsets based on the value of input features. It forms a tree-like structure where each node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome. In this analysis, we have employed several techniques to optimize and evaluate the performance of a Decision Tree Regressor, including pre-pruning, post-pruning, regularization, GridSearchCV, and k-fold cross-validation.

Pre-Pruning (Early Stopping)

Pre-pruning stops the tree from growing beyond a certain point. This helps prevent overfitting by imposing constraints during the growth of the tree.

Parameters:
max_depth=10: Limits the maximum depth of the tree. A deeper tree can capture more details from the data, but it also risks overfitting. Setting a maximum depth helps to generalize better. **min_samples_split=10**: The minimum number of samples required to split an internal node. Ensures that splits occur only when there are enough samples to make meaningful decisions. **min_samples_leaf=5**: The minimum number of samples required to be at a leaf node. Prevents the tree from creating nodes with very few samples, which would be overly specific to the training data. **min_impurity_decrease=0.01**: A node will be split if this split induces a decrease in impurity greater than or equal to this value. It ensures that only splits that significantly improve the model are made.
Justification:
These parameters are chosen to balance the trade-off between bias and variance. Limiting the depth and requiring a minimum number of samples for splits and leaf nodes ensures the model is not too complex and overfitting is minimized.
Post-Pruning (Cost Complexity Pruning)

Post-pruning involves growing a full tree and then pruning it back to a smaller size. This is done using cost complexity pruning where we minimize the cost complexity measure for pruning.
Parameters:
ccp_alpha: Complexity parameter used for Minimal Cost-Complexity Pruning. Larger values of ccp_alpha will prune more of the tree, thus simplifying the model.
Justification:
By pruning the tree after it has been fully grown, we remove branches that have little importance, thereby reducing overfitting and improving generalization.

Regularization Parameters

Regularization involves adding constraints to the tree to avoid overfitting.

Parameters:

max_depth=10: Limits the depth of the tree. **min_samples_split=10**: Minimum number of samples required to split an internal node. **min_samples_leaf=5**: Minimum number of samples required to be at a leaf node.

max_leaf_nodes=20: Maximum number of leaf nodes in the tree. Limits the number of end points, which controls the complexity. **min_impurity_decrease=0.01**: Ensures splits that provide a meaningful decrease in impurity.

Justification:
These parameters help in controlling the complexity of the tree, ensuring that it does not become too detailed and overfit the training data.

Grid Search with Cross-Validation

Grid Search is used to perform hyperparameter optimization by exhaustively searching over a specified parameter grid. Cross-validation is used to evaluate the performance of the model for different hyperparameter combinations.
Parameters:
param_grid: A dictionary specifying the parameters and their respective ranges to search. In this case: **'max_depth'**: [None, 10, 20, 30],**'min_samples_split'**: [2, 10, 20],**'min_samples_leaf'**: [1, 5, 10],**'max_features'**: ['auto', 'sqrt', 'log2']

Justification:
Grid search with cross-validation helps in identifying the best combination of hyperparameters that provide the best model performance. This method ensures that all combinations are tried, and the best one is chosen based on cross-validation performance.
K-Fold Cross-Validation
K-fold cross-validation splits the dataset into K subsets, trains the model K times, each time using a different subset as the validation set and the remaining K-1 subsets as the training set.
Justification:
K-fold cross-validation provides a reliable estimate of the model performance by ensuring that every data point is used for both training and validation. This reduces the risk of overfitting and provides a more accurate measure of model performance.
Decision Tree: Pros and Cons
Pros:
Decision trees are easy to understand and interpret. They mimic human decision-making processes, making them accessible to non-experts.
Decision trees can handle both numerical and categorical data without requiring extensive pre-processing. This flexibility is beneficial in datasets with mixed types of features.

Cons

Decision trees can easily overfit the training data, especially if the tree grows too deep. Overfitting occurs when the model captures noise in the training data, leading to poor generalization to unseen data.
Decision trees can be sensitive to small changes in the training data, leading to different splits and variations in the resulting trees (high variance).
The concept of overfitting and how it affects decision trees underscores the high variance issue.
If some classes dominate, the decision tree might be biased towards those classes, potentially ignoring minority classes.
This is a known drawback of decision trees, especially in imbalanced datasets.
Model 2: XGBoost Algorithm

XGBoost is an implementation of gradient boosting machines designed for speed and performance. It is known for its efficiency at handling large-scale data.

Gradient Boosting Framework: XGBoost iteratively refines its predictions through an ensemble of decision trees. Each tree corrects errors made by previous ones in the series.

Regularization: It includes both L1 and L2 regularization, which helps in reducing overfitting and improves model generalization.

Handling Sparse Data: XGBoost automatically handles missing data and maintains sparsity through its DMatrix data structure, which is optimized for both memory efficiency and training speed.

Hyperparameter Tuning and Application:

Initial Parameter Setting: Started with generic parameter values based on typical use cases.

Advanced Tuning: Utilized GridSearchCV and Bayesian Optimization to systematically explore and optimize parameters.

Hyperparameter Tuning Techniques:

GridSearchCV is a comprehensive approach that performs an exhaustive search over a specified parameter grid and is particularly useful for smaller datasets where the computational expense is manageable.

•*Application:* For the rent_apartments_dataset, GridSearchCV was employed to optimize the XGBoost regressor, exploring parameters such as max_depth, n_estimators, and learning_rate.

•*Outcome:* This method identified the optimal parameters that minimized the MSE, delivering insights into the most effective settings for the regression task.

Bayesian Optimization:

Bayesian Optimization uses a probabilistic model to predict the performance of the model and efficiently finds the optimal parameters by focusing on areas promising improvement.

Integration: Implemented using scikit-optimize, this method was applied to refine the hyperparameters further, improving upon results obtained from GridSearchCV by efficiently navigating the parameter space.

Pros and Cons of XGBoost:

Pros:

Efficiency and Scalability: Handles large data sets with a relatively low memory footprint.

Model Accuracy: Consistently outperforms other algorithms on structured data.

Flexibility: Supports various loss functions and customizations in the model building process.

Robust to Overfitting: Especially with small datasets, thanks to its regularization and cross-validation capabilities.

Cons:

Complexity: Requires careful tuning of parameters which can be time-consuming.

Computationally Intensive: Especially with large datasets and a large number of trees.

Less Effective on Non-structured Data: Such as image, text, or time-series data.

Model 3: Multi-layer Perceptron (MLP)

The multi-layer perceptron is one of the most used types of neural network models. They are built from perceptrons, which are the smallest type of artificial neurons, used to build all types of neural networks. Perceptrons use activation functions to produce an output, which take in inputs, weights and biases. The MLP model is built as a network of perceptrons in at least three main layers, each of which feeds forward to the next.

The input layer is the layer that receives the input data and transmits it to the next layer. Typically, this layer is made up of a neuron for each feature.

There is at least one hidden layer, each of which consist of perceptrons that apply non-linear activation functions to the information received from the previous layer. For my implementation, various hidden layer sizes were experimented with.

The output layer of the MLP uses a different function for classification or regression to output the results of the previous calculation. This layer also uses backpropagation to feed results back to the previous layers so their weights and biases update.

Most implementations of MLP involve other components, which will be explained next.

As there are no cycles of information involved, only feedforward, the MLP is the simplest form of artificial neural network. Despite this, they still provide strong performances on most machine learning tasks, compared to other algorithms.

For these experiments, scikit-learn's implementation of MLPClassifier and MLPRegressor was used. Scikit-learn offers many parameters which can alter performance of the models. To find the optimal model performance, a grid search was implemented for both the regressor and classifier versions of this model, using the same parameters. Grid search is another scikit-learn method that trains and tests the model using k-fold cross validation, using every permutation of hyperparameter input passed into it, and returns the best performing parameters.

The parameter choices for MLP were:

Hidden layer sizes: As explained previously, MLP models have at least one hidden layer made up of a certain number of perceptrons. This parameter defines both the number of hidden layers and the number of perceptrons in each layer. The choices for the model were three hidden layers of 50 perceptrons each, three hidden layers of 50, 100 and 50 layers each, and one hidden layer with 100 perceptrons.

Activation function: The hidden layers can use different activation functions for calculations. There are various options available here, but I chose two of the popular functions for comparison: the hyperbolic tangent function (tanh), and the rectified linear unit function (relu).

Solver: This parameter chooses the weight optimisation function. The choices are either the stochastic gradient

descent (sgd), the simpler gradient descent algorithm, or 'adam', a more advanced gradient descent algorithm widely used in modern machine learning algorithms.
Alpha: This parameter is the L2 regularisation term, which is the value of the loss function, also called ridge regression, which is used to control overfitting. The values compared are 0.0001 and 0.05.
Learning rate: This parameter controls how the learning rate changes over time while the model trains. The 'constant' option keeps the learning rate at the initial value, which is automatically set to 0.001. 'Adaptive' keeps the learning rate equal to the initial value as long as training loss is occurring. If not, it divides the current rate by 5.

While not included in the grid search, the maximum number of iterations (max_iter) is an important hyperparameter that defines how many times each data point is iterated in the model. While it was originally included in the grid search, it was taking up too much time, so it was removed and was set to 1000 for the model.
The grid search found the following combination of hyper-parameters to be the most effective:

For both the classifier and regressor, the hidden layer size of (50,100,50) was found to be the best. The classifier found the relu activation function to work better, while tanh worked better for the regressor. Both models found Adam to be the better solver, and the constant learning rate worked better for both. The last difference was in the alpha regularisation, where the classifier found 0.0001 to work better while the regressor found 0.05 to be better.

Model 4: Inductive Logic Programming (ILP)
As explained in the lecture notes, inductive logic programming (ILP) can be defined as the intersection of machine learning, computational logic and programming. Using logic programming, traditional datasets are represented in the form of hypotheses and background knowledge. Using background knowledge, including positive and negative examples, an ILP model can generate hypotheses and perform testing on them to explain the positive examples and avoid the negative ones.
There are various frameworks for implementing ILP in Python, and we decided to use Pygol, a meta inverse entailment implementation of ILP in Python. Meta inverse entailment is an extension of inverse entailment, a common ILP concept, where hypotheses are derived by reversing the normal entailment process. This extension introduces meta reasoning, which the process can reason about itself, improving the search capabilities for hypotheses.
Pygol uses Python data structures to handle typical logical programming expressions, like background knowledge and examples. It can use either Metagol or Aleph as Progol systems for implementation. We decided to use Aleph. Due to the time taken by the model, we decided to randomly sample 1500 rows from both datasets to use with Pygol. Our background knowledge, positive examples and negative examples were generated directly from the datasets. Pygol then generates the bottom clause by saturating the positive examples with respect to the background knowledge, which are then used to generate the final hypotheses.

In conclusion, ILP models offer a different machine learning approach to the other algorithms used in this coursework, being able to use a stronger hypothesis language and background knowledge. The downsides are the much larger hypothesis spaces required for even simple problems, and the much steeper learning curve compared to other machine learning methodologies.

Model 5:
The model Deep Q-Learning (DQL) was chosen as one of the models for this project. DQNs are a type of reinforcement learning model that combines Q-learning with deep neural networks. This is not a common choice for this kind of task, but it was included to explore its potential in handling the binary classification and regression problems on tabular data.

How DQL Works
Deep Q-Learning (DQL) is an advanced reinforcement learning algorithm for learning optimal policies for sequential decision-making tasks. It combines the Q-learning algorithm with deep neural networks to handle environments with large state and action spaces. The main idea is to use a neural network, which is referred to as the Q-network, to approximate the Q-value function at a given state, which provides an estimate of the future expected rewards for taking a given action in that state.

Model Architecture
In the architecture chosen for this project, the Q-network is made up of three fully connected (dense) layers. The input layer has the same feature dimensionality of the dataset. The first hidden layer is of 128 neurons each having ReLU activation functions, and hence, the layer is also introducing non-linearity to the model. The next hidden layer contains 64 neurons with ReLU activations as well, abstracting the feature representations. The output layer is the Q-values shown for each of the actions, the number being the same as the two possible outcomes in this binary classification problem (Paszke and Towers, 2023).

Learning and the Decision-Making Process
The DQN agent powers the learning and decision-making process itself. It builds the Q-network and a target network with the target network updated regularly to stabilise training. It maintains an epsilon-greedy policy to handle the trade-off of exploration and exploitation. High epsilon value at the start makes the decision-making random to explore the space of action, which decays towards a final point of exploration to then exploit the learned Q-values with training.

The experience replay is an important part of the DQN algorithm. The agent holds its experiences (state, action, reward, next state, done) in a replay buffer. During learning, randomly selected experiences from the buffer are taken for giving an update to the Q-network, which breaks the correlation between consecutive experiences which stabilises the learning (TensorFlow, 2023)..

Hyperparameters

Several hyperparameters play a key role in the performance of the DQL model. The learning rate is used to keep the size of the step for the gradient descent so that the model converges at an appropriate pace without overshooting. The gamma, or discount factor, is used to calculate the weight for upcoming rewards, giving importance to long-term rewards when the factor is close to one. The epsilon value being used in the epsilon-greedy policy starts high to ensure exploration, and as the training goes on, the epsilon decay is used to reduce the exploration, with a minimum epsilon value to ensure some exploration. The memory size is the size of the replay buffer, which provides varied set of experiences to learn from. The batch size is the size of the experiences that are sampled from the replay buffer to update the Q-network, allowing a trade-off between computation efficiency and stability of the learning. The number of episodes is used to control how many episodes of experience are sampled from the environment within the training process, and the target update frequency is how often the target network and the Q-network are synchronised to stabilise the training process. (Paszke and Towers, 2023).

The DQL agent learns from the environment through multiple episodes. In each episode, the agent chooses its actions according to the epsilon-greedy policy so that there is a trade-off between exploration and exploitation. The chosen actions are compared to the actual labels to calculate rewards where right predictions are rewarded by positive rewards and wrong ones by negative rewards. The experiences are inserted into the replay buffer, and the agent will sample a minibatch of experiences stored in the replay buffer to update the Q-network by gradient descent. The target network is also updated occasionally so as to dampen oscillations and stabilise training (Wang, 2020).

DQL was chosen for the case study to explore its strengths in solving the given binary classification problem of identifying whether an individual’s income is more than $50K/year and the regression problem of determining the rent of an apartment. Although not conventional for such problems, DQL’s capacity to learn from high-dimensional inputs and sequential decision making provides an interesting approach for attacking the problem. The exploration also facilitates gaining an insight into the flexibility and weakness of DQL in the context of various machine learning problems, which in turn provides some insight into its application that extends beyond conventional reinforcement learning settings. Although computationally expensive and hard to tune its hyperparameters, DQL is capable of scaling to large state spaces effectively and taking advantage of experience replay and target networks in stabilising the training.

Model 6: Support Vector Machine (SVM)

Support Vector Machine is a supervised learning model commonly used for classification and regression. The model creates a hyperplane, which is a line in a two-dimensional space, that attempts to separate data between classes. Once the data points are separated, support vectors are created from the data points closest to the hyperplane.

As with previous models, the grid search was used to find the optimal hyperparameters.

For the regression task, the following parameters were optimised:

‘C’: the regularisation parameter that defines the margin for classifying results. The options were 0.1, 1 and 10, and grid search found 10 to be the most optimal.

‘epsilon’: specifies the penalty in the training loss function. The options were 0.01, 0.1 and 0.2. 0.2 was the best performing for regression, and 1 for the classifier.

‘kernel’: specifies the algorithm’s choice of kernel type. The choice was between linear and ‘rbf’, and ‘rbf’ was more successful for both.

‘gamma’: this parameter effects how influential individual training examples are, with lower values meaning less influence and higher values meaning more influence. The choices were between ‘scale’ and ‘auto’ and ‘auto’ was chosen.

While SVM is a powerful algorithm, notable effective in when working with data with many dimensions, it is very computationally intensive, and so can struggle with larger datasets. Despite this, the customisability of the hyperparameters means it can be tuned easily for better results.

Model 7: Algorithm: K-Nearest Neighbors (KNN) Regression

KNN is a non-parametric, instance-based learning algorithm used for regression and classification tasks. For regression, it predicts the target value based on the average of the nearest neighbours.

Parameters and Justifications:

n_neighbors: [3, 5, 7, 9, 11]

Testing various neighbour counts helps identify the optimal number of neighbours that minimize prediction error. Too few neighbours might lead to a noisy model, while too many can over smooth the predictions.

weights: ['uniform', 'distance']

Justification: 'Uniform' assigns equal weight to all neighbours, while 'distance' gives closer neighbours more influence. Testing both allows determining which approach better captures the relationship between features and the target variable.

algorithm: ['auto', 'ball_tree', 'kd_tree', 'brute']

Justification: Different algorithms handle the nearest neighbor search differently. 'Auto' selects the best

algorithm based on the data. Including others ensures that the grid search tests all possible approaches for efficiency and accuracy.
leaf_size: [20, 30, 40]
Justification: Leaf size affects the speed and memory requirements of tree-based algorithms (ball_tree and kd_tree). Testing different sizes helps find the best trade-off between speed and accuracy.
p: [1, 2]
Justification: Defines the distance metric to use: 1 for Manhattan distance and 2 for Euclidean distance. Including both ensures the most appropriate distance metric for the data is selected.
Experimental Results:
Best Parameters: {'n_neighbors': 5, 'weights': 'distance', 'algorithm': 'auto', 'leaf_size': 30, 'p': 2}
Mean Squared Error (MSE): A measure of the average squared difference between the observed actual outcomes and the predictions.
Pros and Cons of K-Nearest Neighbors (KNN) Regression
Pros:
KNN is easy to understand and implement, making it accessible for beginners. It doesn't require a training phase, allowing for quick initial deployment. It is versatile, applicable for both classification and regression tasks, and does not assume any underlying data distribution, which makes it flexible.
Cons:
KNN is computationally intensive, especially with large datasets, due to the need to compute distances for each test instance against all training instances. It is sensitive to noise and irrelevant features, can be challenging to choose the optimal K, and struggles with high-dimensional data, leading to poorer performance.

4. Model evaluation / Experiments

Based on our dataset and the machine learning techniques we have used, here's an explanation of the choice of using F1 score, confusion matrix, and ROC curve for model evaluation for classification.

1. F1 Score

The F1 score is the harmonic mean of precision and recall. It is a single metric that combines both precision (the accuracy of the positive predictions) and recall (the ability to find all the positive samples).

2. Confusion Matrix

A confusion matrix is a table that shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). It provides a comprehensive breakdown of model performance.

3. ROC Curve and AUC

The ROC (Receiver Operating Characteristic) curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The area under the ROC curve (AUC) is a single value summary of the ROC curve, representing the model's ability to distinguish between classes.

Application to our Dataset

Our dataset involves predicting income levels based on various features. Given the potential class imbalance (more people earning <= 50K than > 50K), the F1 score, confusion matrix, and ROC curve provide a comprehensive evaluation framework:

The F1 score balances precision and recall, ensuring that both false positives and false negatives are considered.

The confusion matrix gives a detailed breakdown of correct and incorrect classifications, allowing for detailed error analysis. The ROC curve and AUC provide an overview of the model's performance across all thresholds, helping to choose the best threshold for classification**.**
Based on our second dataset and the regression techniques we have used, here's an explanation of the choice of using Mean Squared Error (MSE), R-squared (R^2), and Actual vs Predicted graph for model evaluation.

1. Mean Squared Error (MSE)

Mean Squared Error (MSE) is the average of the squares of the differences between the predicted and actual values. It is a measure of the quality of an estimator; in this case, how well the regression model predicts the target variable.

2. R-squared (R^2)

R-squared (R^2) is a statistical measure that represents the proportion of the variance for the dependent variable (target) that's explained by the independent variables (features) in the model. It provides an indication of goodness of fit.

3. Actual vs Predicted Graph

An Actual vs Predicted graph plots the actual values of the target variable against the predicted values from the model. Ideally, the points should lie on or near the 45-degree line that represents perfect predictions.

4.1 Experiment 1

4.1.1 NULL HYPOTHESIS 1

The null hypothesis (H_0) is that the nature of the binary classification task, the size of the data, and the nature of the data types in the Adult dataset do not provide hugely different results in the predictive power of the provided machine learning models.

4.1.2 MATERIAL & METHODS 1

The nature of this experiment is that it is a binary classification task that predicts the people whose income exceeds a value of $50K/year based on the Adult dataset, which contains a total of 23,374 records and 14 features. The dataset has been cleaned and consists of standardised numerical and encoded categorical features. The models were evaluated based on three performance scores: ROC AUC, Test Accuracy, and Cross-Validation Accuracy.

ROC AUC is suitable for the Adult dataset as it tells how well the model can distinguish between the target variable classes, income, at different thresholds. As the classes can be unbalanced, ROC AUC is a balanced measure of performance, which is one of the critical considerations when checking the performance of models on imbalanced data.

Test Accuracy directly gives the idea of how well the model generalises to unseen data. In the Adult dataset, as there are a large number of training examples and a diverse mixture of numerical and categorical features, a high test accuracy score tells that the model thinks very well and robustly on diverse and unseen inputs so a reliable prediction of income can be ensured.

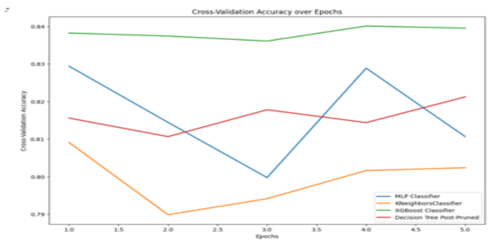
These performance scores after each epoch are used to ensure that performance can be checked and overfitting or underfitting can be detected initially. The generalisation abilities and accuracy of the model are thereby optimised as the training goes on. This method is critical in dealing with the binary classification task and diverse features of the Adult data.

Each of the models is expected to perform differently given the strength of these characteristics. The MLP Classifier is expected to perform with success based on the expectations of stable training of the neural network as a result of the handling of the standardised feature. However, the KNeighborsClassifier may not perform as well since it uses distance metrics in the high-dimensional space and hence may express relatively lower performance even in the presence of the standardised data. The XGBoost Classifier is expected to perform very well as a result of its robustness and proper handling of the data types at hand. The Decision Tree Post-Pruned model is expected to perform fine with strong interpretability, though it may not perform as well as the XGBoost in light of the risks of overfitting that the pruning factor is built to mitigate. The DQN Classifier is expected to perform widely from case to case, as it is less popularly used for tabular data and may not perform as well as conventional classifiers for this particular task. The SVM Classifier is said to achieve good performance in binary classification problems with standardised data, but how good it will perform would greatly depend on how you select your regularisation and kernel.

ILP Approach

As explained in the lecture notes, inductive logic programming (ILP) intersects machine learning, computational logic, and programming. Traditional datasets are represented in the form of hypotheses and background knowledge. Using background knowledge, including positive and negative examples, an ILP model can generate hypotheses and perform testing to explain the positive examples and avoid the negative ones. We used Pygol, a meta inverse entailment implementation of ILP in Python. Meta inverse entailment extends inverse entailment by introducing meta reasoning, enhancing the search capabilities for hypotheses. Pygol uses Python data structures to handle logical programming expressions, like background knowledge and examples. It can use either Metagol or Aleph as Progol systems; we chose Aleph. Due to the model's processing time, we randomly sampled 1500 rows from the dataset for use with Pygol. Background knowledge, positive examples, and negative examples were generated directly from the dataset. Pygol then generates the bottom clause by saturating the positive examples with respect to the background knowledge, which are then used to generate the final hypotheses. In conclusion, ILP models offer a different machine learning approach compared to the other algorithms used in this coursework. They can use a stronger hypothesis language and background knowledge. However, they require much larger hypothesis spaces for even simple problems and have a steeper learning curve compared to other machine learning methodologies.

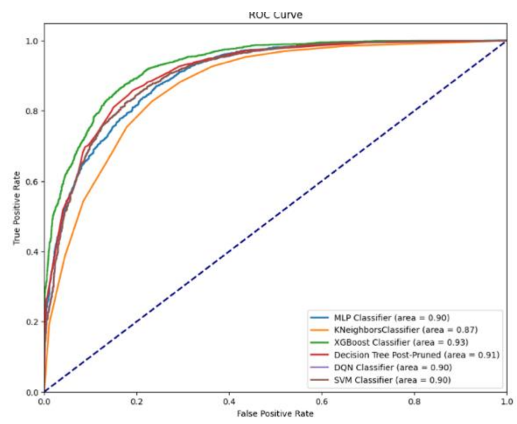
4.1.3 RESULTS & DISCUSSION 1

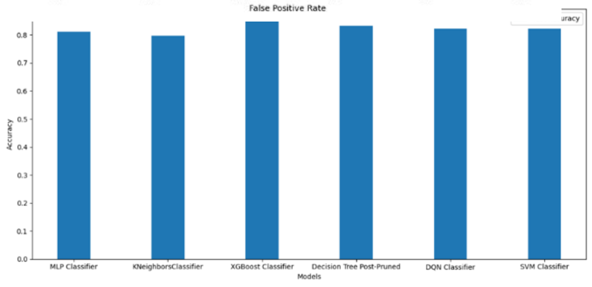


Mean Cross-Validation Accuracy

The MLP Classifier achieved a mean cross-validation accuracy of 0.8166, indicating consistent performance across different subsets of the data. The KNeighborsClassifier had a mean cross-validation accuracy of 0.7994, slightly lower than the MLP Classifier, indicating it may struggle more with the high-dimensional space and distance metrics despite standardised features. The XGBoost Classifier had a mean cross-validation accuracy of 0.8383, better than other models, indicating its robustness and efficiency in the face of complicated datasets. The Decision Tree Post-Pruned had a mean cross-validation accuracy of 0.8159. The DQN Classifier and SVM didn’t have cross validation as they took too long.

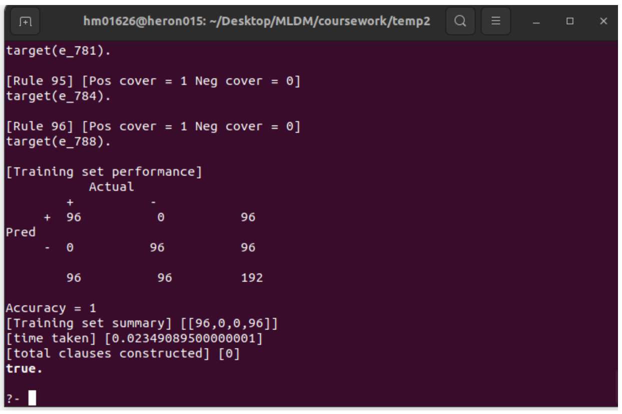
Test Accuracy





The MLP Classifier had a test accuracy of 0.8111, indicating good generalisation to unseen data. The KNeighborsClassifier had a test accuracy of 0.7970, slightly lower than its cross-validation accuracy, slightly lower then MLP but not by too much. The XGBoost Classifier achieved the highest test accuracy at 0.8496, confirming its effectiveness and robustness. The Decision Tree Post-Pruned had a test accuracy of 0.8332, slightly higher than its cross-validation accuracy, thus successfully demonstrating the process of pruning. The DQN Classifier had a test accuracy of 0.8225, strong performance on the dataset. The SVM Classifier also had a test accuracy of 0.8225, indicating good generalisation from the training data, as indicated by the similar cross-validation performance.

ROC AUC: The MLP Classifier had a ROC AUC of 0.9016, indicating good discriminative performance between income classes. The KNeighborsClassifier yielded a ROC AUC of 0.8715, indicating strong discriminative power but less so than other best models. The XGBoost Classifier yielded the highest ROC AUC at


```
hm01626@heron015: ~/Desktop/MLDM/coursework/temp2
target(e_781).

[Rule 95] [Pos cover = 1 Neg cover = 0]
target(e_784).

[Rule 96] [Pos cover = 1 Neg cover = 0]
target(e_788).

[Training set performance]
           Actual
        +          -
     +  96          0          96
Pred
     -  0          96          96

        96          96         192

Accuracy = 1
[Training set summary] [[96,0,0,96]]
[time taken] [0.023490895000000001]
[total clauses constructed] [0]
true.

?- 
```

0.9292, indicating superior capability of distinguishing between classes. The Decision Tree Post-Pruned yielded a ROC AUC of 0.9089, indicating very strong performance. The DQN Classifier yielded a ROC AUC of 0.9039, indicating strong discriminative power, only being beaten by XGBoost and matched by SVM. Finally, the SVM Classifier likewise yielded a ROC AUC of 0.9039, indicating strong discriminative power that agrees with its test accuracy.

While the ILP approach was implemented successfully the results were not very satisfactory as we only used one split to train and test. this can be improved using a proper split to test the data and hence the reason why of such high accuracy.

4.2 Experiment 2

4.2.1 NULL HYPOTHESIS 2

Null Hypothesis (H_0): The various features in the apartment rental dataset, such as the number of bathrooms, number of bedrooms, source of listing, square footage, and amenities, do not significantly affect the rental price of the apartments.

4.2.2 MATERIAL & METHODS 2

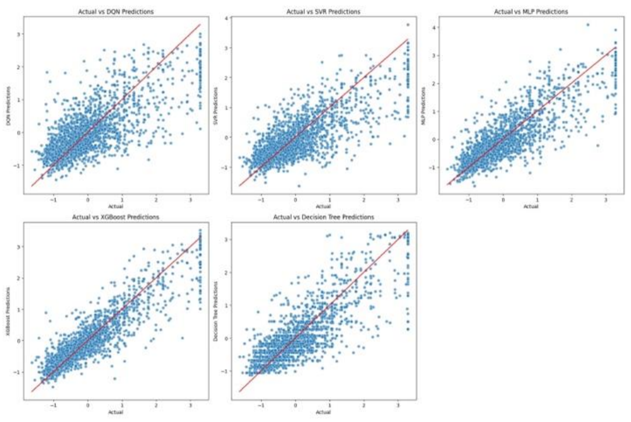
The nature of this experiment is to predict apartment rental prices based on a dataset that contains various features such as the number of bathrooms, number of bedrooms, source of listing, square footage, and amenities. The dataset has been cleaned and consists of standardised numerical and encoded categorical features.

The models were evaluated based on three performance scores: Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2).

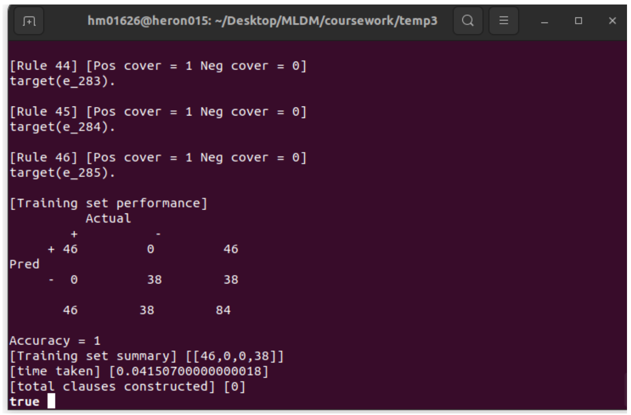
Cross-Validation is important because the dataset has features in standardised form. Cross-validation divides the dataset into folds to give a robust estimate of performance, which helps in tuning the hyperparameters.

Each of the models is expected to perform differently given the strength of these characteristics. The MLP Regressor is expected to perform with success based on the expectations of stable training of the neural network because of the handling of the standardised feature. However, the KNeighbors Regressor may not perform as well since it uses distance metrics in the high-dimensional space and hence may express relatively lower performance even in the presence of the standardised data. The XGBoost Regressor is expected to perform very well because of its resilience and proper handling of the data types at hand. The Decision Tree Regressor is expected to perform fine with reasonable power of interpretability, though it may not perform as well as the XGBoost considering the risks of overfitting that the pruning factor is built to mitigate. The DQN Regressor is expected to perform widely from case to case, as it is less popularly used for tabular data and may not perform as well as conventional regressors for this task. The SVR is said to achieve good performance in regression problems with standardised data, but how well it will perform would greatly depend on the selection of regularisation and kernel.

As explained in the lecture notes, inductive logic programming (ILP) intersects machine learning, computational logic, and programming. Traditional datasets are represented in the form of hypotheses and background knowledge. Using background knowledge, including positive and negative examples, an ILP model can generate hypotheses and perform testing to explain the positive examples and avoid the negative ones. We used Pygol, a meta inverse entailment implementation of ILP in Python. Meta inverse entailment extends inverse entailment by introducing meta reasoning, enhancing the search capabilities for hypotheses. Pygol uses Python data structures to handle logical programming expressions, like background knowledge and examples. It can use either Metagol or Aleph as Progol systems; we chose Aleph. Due to the model's processing time, we randomly sampled 1500 rows from the dataset for use with Pygol. Background

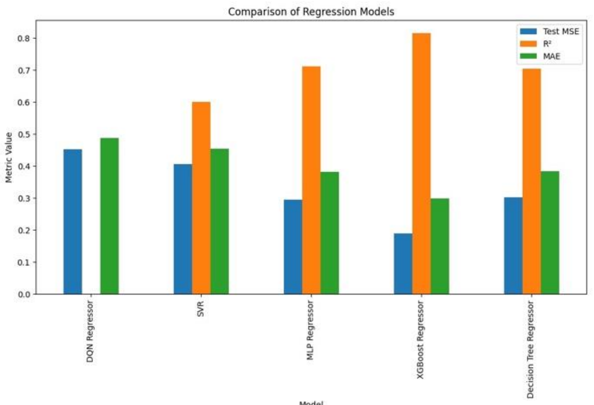


knowledge, positive examples, and negative examples



```
[Rule 44] [Pos cover = 1 Neg cover = 0]
target(e_283).

[Rule 45] [Pos cover = 1 Neg cover = 0]
target(e_284).

[Rule 46] [Pos cover = 1 Neg cover = 0]
target(e_285).

[Training set performance]
          Actual
        +         -
     + 46         0         46
Pred
     -  0        38         38

       46        38         84

Accuracy = 1
[Training set summary] [[46,0,0,38]]
[time taken] [0.04150700000000018]
[total clauses constructed] [0]
true
```

were generated directly from the dataset. Pygol then generates the bottom clause by saturating the positive examples with respect to the background knowledge, which are then used to generate the final hypotheses. In conclusion, ILP models offer a different machine learning approach compared to the other algorithms used in this coursework. They can use a stronger hypothesis language and background knowledge. However, they require much larger hypothesis spaces for even simple problems and have a steeper learning curve compared to other machine learning methodologies.



4.2.3 RESULTS & DISCUSSION 2

while interpretable, exhibited higher errors and a tendency to overfit compared to XGBoost and MLP. The DQN Regressor's higher errors suggest it may not be well-suited for this type of data or requires further optimization. Overall, the experiment highlights the importance of model selection and data handling in achieving accurate predictions for rental prices.While the ILP approach was implemented successfully the results were not very satisfactory as we only used one split to train and test. This can be improved using a proper split to test the data and hence the reason it has such high accuracy.

Algorithms	Test MSE	R^2	MAE
DQN Regressor	0.4526	nan	0.4876
SVR	0.4062	0.6008	0.4540
MLP	0.2946	0.7105	0.3813
XGBoost	0.1888	0.8145	0.2989
Decision Tree Regressor	0.3012	0.7040	0.3836

The results indicate that the XGBoost Regressor performed the best, achieving the lowest MSE and MAE, and the highest R², making it the most effective model for predicting rental prices in this dataset. The MLP Regressor also showed strong performance, suggesting that neural networks can capture complex patterns effectively when the data is properly standardized. SVR demonstrated moderate performance, indicating its capability in handling high-dimensional data, though its effectiveness depends on the choice of kernel and regularisation parameters. The Decision Tree Regressor showed middling performance when compared to the other models, while the DQN and SVR regressors showed poor performance.

5. Conclusion

The study is based on seven machine learning algorithms: decision tree, XGBoost, MLP, ILP, KNN, and SVM, with one reinforcement model, Q-learning, and consideration of two classes. The first one is the classification by income, and the second one is regression-based price prediction. The comparative study between the models identifies XGBoost as the best-performing classification model, with accuracy and AUC-ROC, hence both its robustness and effectiveness, respectively. Both the MLP and the decision tree were acceptable, but the KNN and the ILP were only slight fits into the moderate category. Both the SVM and the DQN had discrimination power when the parameters and computational resource settings became proper. Once again, having the lowest MSE and MAE, the highest R^2, XGBoost was the one to really fit best for the regression task. After it, the MLP followed it closely, hence indicating it could really catch complex patterns. For SVR, moderately successful results were given; Decision Tree was interpretable but had a serious problem of overfitting, while KNN was suboptimal. On the other hand, due to its sensitivity to high-dimensional data and large computational complexity, the performance of ILP was low. In other words, from the study experience, it was of utmost importance among model selection and hyperparameter tuning to beget not excellent but reliable predictions. More specifically, XGBoost was noted to hold state-of-the-art performance in many competitions on these tasks.

Contributions:
Fin: Reinforcement learning and ILP research and explanations. Got dataset, Data-preprocessing, EDA and Data visualisations, report, combined code for comparisons
Harsh: SVM, K-NN, complete implementation of ILP models and explanations. EDA contributions and report writing.
Meenu: Decision trees and ILP research and explanations, Data Preprocessing
Harshita: XGboost model and explanations. EDA contributions and report writing.
Luke: MLP and researching ILP and explanations. EDA changes, report writing

Reference list

archive.ics.uci.edu. (n.d.). *Rent dataset*. [online] Available at: https://archive.ics.uci.edu/dataset/555/apartment+for+rent+classified.

Becker, B. and Kohavi, R. (1996). *Adult Dataset*. [online] archive.ics.uci.edu. Available at: https://archive.ics.uci.edu/dataset/2/adult.

Guo, G., Wang, H., Bell, D., Bi, Y. and Greer, K. (2003). KNN Model-Based Approach in Classification. *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, 2888, pp.986–996. doi:https://doi.org/10.1007/978-3-540-39964-3_62.

Inductive Logic Programming. (2008). *Lecture notes in computer science*. Springer Science+Business Media. doi:https://doi.org/10.1007/978-3-540-85928-4.

Jang, B., Kim, M., Harerimana, G. and Kim, J.W. (2019). Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access*, 7, pp.133653–133667. doi:https://doi.org/10.1109/access.2019.2941229.

Patil, D.D., Wadhai, V.M. and Gokhale, J.A. (2010). Evaluation of Decision Tree Pruning Algorithms for Complexity and Classification Accuracy. *International Journal of Computer Applications*, 11(2), pp.23–30. doi:https://doi.org/10.5120/1554-2074.

Ramraj Santhanam, Nishant Uzir, Sunil Raman and Shatadeep Banerjee (2017). *Experimenting XGBoost Algorithm for Prediction and Classification of Different Datasets*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/318132203_Experimenting_XGBoost_Algorithm_for_Prediction_and_Classification_of_Different_Datasets.

ResearchGate. (n.d.). *(PDF) Multilayer perceptron and neural networks*. [online] Available at: https://www.researchgate.net/publication/228340819_Multilayer_perceptron_and_neural_networks.

scikit-learn. (n.d.). *sklearn.neural_network*. [online] Available at: https://scikit-learn.org/stable/api/sklearn.neural_network.html [Accessed 29 May 2024].

Varghese, D. (2024). *danyvarghese/PyGol*. [online] GitHub. Available at: https://github.com/danyvarghese/PyGol.

Varghese, D., Didac Barroso-Bergada, Bohan, D.A. and Alireza Tamaddoni-Nezhad (2024). Efficient Abductive Learning of Microbial Interactions Using Meta Inverse Entailment. *Lecture notes in computer science*, pp.127–141. doi:https://doi.org/10.1007/978-3-031-55630-2_10.

TensorFlow. (2023). Introduction to RL and Deep Q Networks. Available at: https://www.tensorflow.org/agents/tutorials/0_intro_rl (Accessed: 29 May 2024).

Wang, M. (2020) 'Deep Q-Learning Tutorial: minDQN - A Practical Guide to Deep Q-Networks', Towards Data Science, 18 November. Available at: https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-a-practical-guide-to-deep-q-networks-47d1f3e32ecb (Accessed: 29 May 2024).

Paszke, A., & Towers, M. (2023). Reinforcement Learning (DQN) Tutorial. PyTorch Tutorials 2.3.0+cu121 documentation. Available at: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html (Accessed: 29 May 2024).