Machine learning and data mining experiments across datasets

James Prestwich jp01702@surrey.ac.uk
Pauline Schouten ps00711@surrey.ac.uk
Rudra Someshwar rs01922@surrey.ac.uk
Steryios Nicolaides sn00999@surrey.ac.uk

Abstract

This report compares machine learning algorithms across 4 datasets and 2 reinforcement learning environments. The first 3 datasets are used to compare Decision Tree (DT), Support Vector Machine (SVM) and Neural Network (NN) based algorithms across different types of tasks and data: regression, multi-class classification and a classification task using complex image data. The next dataset is a relational learning task used to compare rule-learning time for different Inductive Logic Programming (ILP) systems on the UW-CSE dataset. Finally, two types of reinforcement learning algorithm Q-learning and Sarsa are compared across two environments from the Gymnasium toolbox. On the regression task SVM performed best, on both classification tasks DT performed best. On ILP, PyGol performed better than Aleph with regards to both learning time and accuracy. Q-learning outperformed Sarsa in both environments.

1. Project Definition

The first part of this report looks to compare DT, SVM and NN based algorithms across different types of tasks and data to see which algorithms are best suited to which. A regression task on predicting house prices in Boston, a multiclass classification task on a room occupancy and a binary classification task using image data to determine the health of leaves were used. Further detail on each of these datasets as well as the metrics used to assess performance are discussed in Section 2. It is expected that the NN and SVM algorithms will perform better than the DT on the Boston house price dataset due to the complexity of the data, whilst the DT algorithm will outperform the other on the room occupancy dataset due to smaller number of features and imbalance in the classes. On the leaf classification task, it is predicted the NN will outperform the other two algorithms as it can handle the complex image dataset and it can use convolutional layers as part of its architecture to identify spatial features.

The next part looks at Inductive Logic Programming (ILP) applied to a relational learning dataset (selected from the CVUT Prague Relational Dataset Repository [1]). The lecture slides (Week 7 pg27) were followed to define the problem for inductive learning:

1. A set of positive and negative training examples
e.g. Positive Examples (pos_example.f):
advisedBy(person265, person168).
e.g. Negative Examples (neg_example.n):
advisedBy(person292, person342).
2. Background theory as a Prolog file
e.g Background knowledge (bk_train.pl):
taughtBy(course44, person171, autumn_0001).
student(person284).
3. Hypothesis, which is the **advisedby(A, B)** relation
4. Evaluation function, chosen to be **accuracy**

Two ILP systems, Aleph and PyGol, are used to compare learning times and accuracies for this relational learning task. It is expected that learning times will be mainly impacted by the number of training examples as the hypothesis space grows. PyGol, with its novel approach of Meta Inverse Entailment (MIE) [2], might be more efficient in handling complex relational patterns, leading to faster convergence and potentially higher accuracies.

Finally, two methods of updating Q-values in a model free reinforcement algorithm, Q-learning and Sarsa, are compared across two simple environments from the Gymnasium Toy Text project [3]. Both these environments involve navigating a grid, in the case of Taxi to transport a "passenger" from one location to another and in the case of the Cliff Walking to reach a goal without falling off a cliff. More detail on these can be found in Section 2.5. The purpose of this is to better understand the differences in training and final policies produced by these algorithms so the correct algorithm can be selected depending on the objectives of a reinforcement learning task. It is expected that Q-learning will produce a better final policy whilst Sarsa will perform better in training.

2. Data Preparation

2.1 Dataset 1 – Boston Housing Prices

Dataset 1 [4] is a regression task with non-linear dependencies between features. The dataset contains information on the various factors that influence house prices in Boston, Massachusetts, USA. There are 13 attributes being used to predict the target variable: MEDV (house price). The following steps are taken to prepare the data for analysis and model development:

1. Plotting data correlation between variables to help identify relationships or dependencies between features as can be seen in Figure 1.
2. Plotting data distribution per variable to identify any skew or outliers in each feature as can be seen in Figure 2 (additional scatterplots present in the notebook show the skew in each feature)
3. Data cleaning to ensure data validity. There are no null values, and all variables are numerical. However,

Figure 2 identifies major outliers in the RM (average number of rooms per dwelling) feature that have been removed to not influence the models.

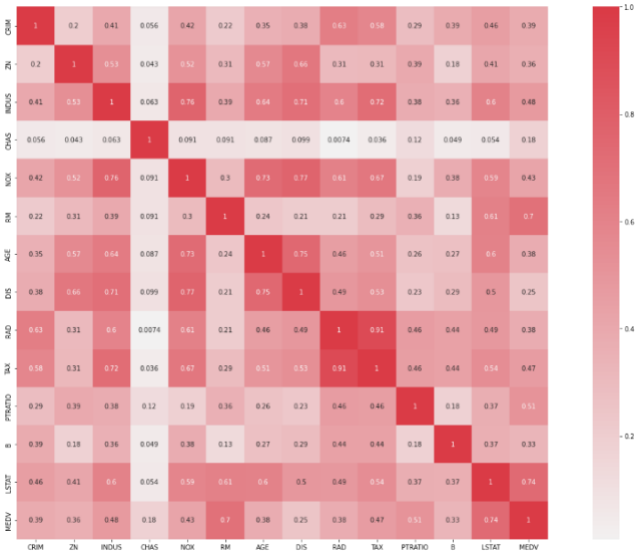

Figure 1: Correlation heatmap for the attributes in dataset 1

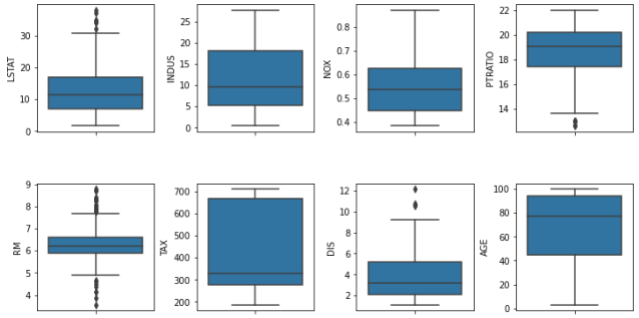

Figure 2: Boxplot distribution of data in each variable

2.2 Dataset 2 – Room Occupancy

Dataset 2 [5] is a multiclass classification task for predicting the number of occupants in a room, ranging from 0 to 3. There are 16 attributes to the data and 10129 instances, the attributes correspond to 4 sensors for sound, light and temperature, a CO2 sensor measures both ppm as well as slope from a sliding window and 2 motion detectors. The data is all numerical, with no null values so minimal data cleaning is required.

In Figure 3 the imbalance in the target classes is evident, with a strong majority of the instances having an occupancy of 0. Different class balancing techniques were compared during training and F1 score was chosen as the primary metric to assess performance as it gives a better indication of how well the model sorts the data into each class; accuracy is then used as a secondary metric to give an overall view of the performance. Figure 4 shows all attributes are correlated to the room occupancy, so it would not be worthwhile removing any. Yet, the 4 temperature sensors are highly correlated to one another (0.7 to 0.95) as well as the light sensors being somewhat correlated.

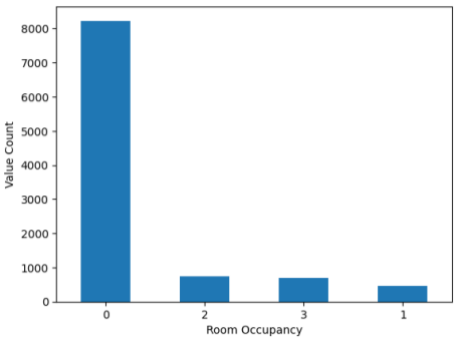

Figure 3: Bar chart of the number of instances belong to each target class in dataset 2.

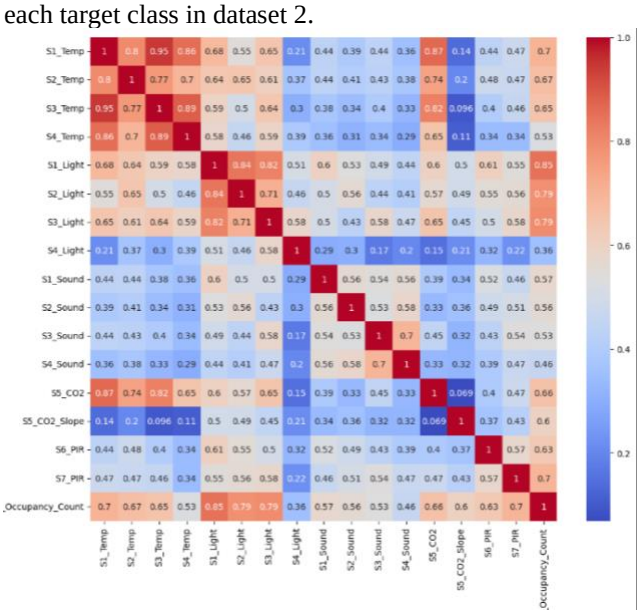

Figure 4: Correlation heatmap for the attributes in dataset 2.

The data was scaled using min-max scaling to vary between 0 and 1 to ensure that all features had equal importance as features with large scales could bias a learning algorithm. Next a dataset was made that merged the 4 temperature sensors and 4 light sensors to test the correlations discussed earlier; this dataset was initially compared with the unmerged version to see if this boosted performance. The datasets were divided randomly into training and testing splits in a 75:25 ratio. As the target classes were imbalanced, four more versions of the training data were generated using random over sampler (ROS), SMOTE (Synthetic Minority Oversampling Technique), random under sampler (RUS) and near miss under sampling.

2.3 Dataset 3 – Healthy/Diseased Leaf Images

Dataset 3 [6] is an image dataset for a binary classification task for classifying images of leaves as healthy or diseased. The dataset is well structured with 10 categories of leaves. Upon exploring this dataset, valuable insights are gained.

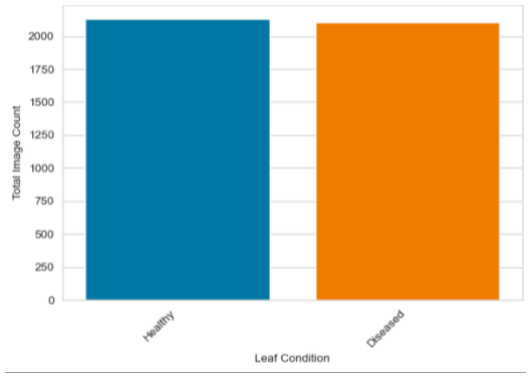

Figure 5: Bar chart showing image counts by leaf category.

As the task is binary classification, Figure 5 shows that both healthy and diseased classes are balanced, and no further balancing is required. There are a total of 4236 images in this dataset and all of them are the same size of 6000 x 4000 pixels. Sample images from each category were displayed at random to get a better idea of the dataset. Upon inspection, it looks like all the images are taken under similar lighting conditions and background, which improves the quality of the dataset vastly, and there is less noise during training. Next, the colour distribution below also shows differences between the healthy and diseased leaves. This makes it easier for the model to differentiate.

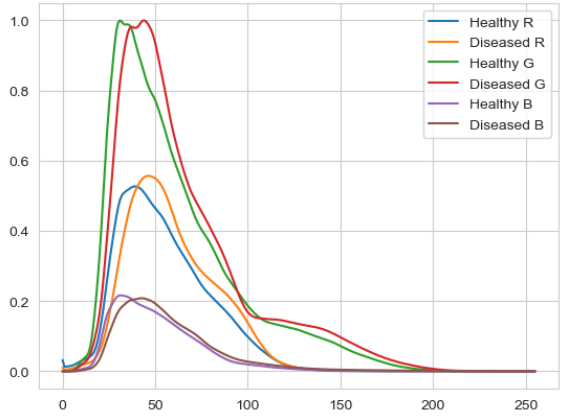

Figure 6: Colour distribution comparison for healthy vs diseased leaves

The Kolmogorov-Smirnov (KS) statistic and P-values were also obtained for these 2 categories for each leaf type, and the entire dataset. The KS test is a nonparametric statistical test that can be used to determine whether two samples are drawn from the same distribution. A larger KS statistic indicates a larger difference between the two distributions. Similarly, the p-value is a measure of the evidence against the null hypothesis that the two samples are drawn from the same distribution. A small p-value (typically less than 0.05) indicates that there is strong evidence against the null hypothesis and that the two samples are likely drawn from different distributions. For the entire dataset, **KS statistic = 0.1042, and P-value = 0.00048.** This indicates that there is evidence of samples being obtained from different distributions, which should strengthen the models' performances and make it easier to classify the images.

After ensuring that the dataset was clean and well-structured, the images were resized to 600x400 pixels to save time and computing resources. This smaller size would be sufficient for most purposes, such as visualizing the images or performing simple analysis. For the CNN experiments, the images were further reduced to one channel using different techniques and further resized to 224x224 pixels. This was done to ensure that the images would fit in the GPU RAM while training on Google Colab, which has limited resources. To create input for the DT and SVC classifiers, once the images were loaded another function was employed that calculated and created colour histograms from those images. The parameters needed as input to calculate said histograms were 'image' and 'bins'. 'bins' represented a tuple that specified the number of bins ((8,8,8) - 8 bins) for each color channel (in this case, [0, 1, 2] - blue, green, and red). The function then normalized the histogram so that it could be displayed as an image and returned a flattened histogram array.

2.4 Dataset 4 – ILP

The dataset consists of a set of Prolog clauses that encode relationships between entities within the UW-CSE (University of Washington Computer Science and Engineering) domain. These clauses represent aspects such as student-advisor relationships, courses, publications, and other relevant information related to the academic domain.

The Prolog files were sourced from a public Github repository [7]. The training examples were extracted from the "exs_train.pl" file and split into positive and negative examples. To prevent excessively long training times, the number of examples was significantly reduced. Additionally, mode declarations were extracted from the "aleph-modes.pl" Prolog file, providing additional information about the predicates and their argument types to aid in the learning process. This last step is undertaken to prepare the dataset for effective rule generation using Aleph. Pygol does not require this mode declaration file.

2.5 Reinforcement Learning Environments

To compare reinforcement learning algorithms two environments were used from the Gymnasium Toy Text project [3]. The first environment used was Taxi, this problem requires the algorithm to pick up passengers and drop them at the correct location. There are 4 drop off locations in a 5x5 grid and in each square, there are 6 possible actions: moving in one of 4 directions, picking up a passenger or dropping off a passenger. Rewards were:

- -1 per step unless other reward is triggered.
- +20 for delivering a passenger correctly.
- -10 executing "pickup" and "drop-off" actions illegally.

The second environment explored was Cliff Walking from the same environment library as before. In this case, the Cliff Walking problem requires the algorithm to avoid falling off a cliff when crossing a gridworld from start to goal. There is a starting location at [3,0] and a goal location at [3,11] in a 4x12 grid. There are 4 possible actions: move up, right, down and left. Rewards were:

- -1 per step unless other reward is triggered.

- -100 if the player stepped into the cliff.

To compare how the models performed, the average reward obtained by the optimal policy at the end of training, and the number of training episodes required to converge on an optimal policy were looked at. The primary objective of reinforcement learning is to maximise cumulative reward. Comparing the average reward of the final policy allows us to compare which of the two algorithms achieved this goal better. The second metric allows us to assess how well the algorithms might perform when training time is limited or when quick learning is required. The best algorithm will strike a balance between these two metrics.

3. Model development

3.1 Decision Tree Algorithms

DTs are structured as a flowchart-like tree that splits the data at each node using a decision based on one of the features in the data. The feature used to split the data is usually chosen as one that produces the greatest separation between classes in classification or reduction in variance in regression tasks. The data is repeatedly split into smaller subsets until the final 'leaf' node which will give a class label or predicted value.

DTs are popular machine learning algorithms because they are interpretable, as rules learnt can be visualized. They can capture non-linear relationships and are fast to train and infer. However, they are prone to overfitting and are unable to capture as complex features as some other algorithms.

For Dataset 1, the DT regressor from sk-learn was used as this a regression task. The regression DT algorithm uses a different splitting criterion (usually MSE or MAE) to determine best feature and threshold for splitting. For Dataset 2, random forest from sk-learn was used, this is where a collection of smaller DTs each trained on subsets of the data and features are used to vote on a class label. This was selected over a traditional DT as it is better at handling minority classes in imbalanced data like this dataset. It can also handle more complex relationships as may be the case with the number of different sensor inputs. For Dataset 3 a standard DT classifier from sk-learn was used due to its ability to handle high dimensionality datasets whilst retaining a shorter training time.

3.2 SVM Algorithms

SVM algorithms transform the data into a higher-dimensional feature space using a kernel function before finding an optimal hyperplane boundary to separate the classes in classification or a function to capture the relationship in regression. SVM algorithms perform well in complex tasks with large numbers of features and can capture complex non-linear relationships. However, they require careful hyperparameter tuning, struggle with noisy data and are less interpretable than other algorithms.

As Dataset 1 was a regression problem, the support vector regression (SVR) algorithm from sk-learn was used and as Datasets 2 and 3 were classification tasks, the support vector classifier (SVC) from sk-learn was used.

3.3 Neural Network Algorithms

NN algorithms use interconnected artificial neurons or nodes which are grouped into layers. These nodes are connected using weights which determine the strength of the connection. Inputs pass through the network and weighted sums of these are received by the neurons where they apply an activation function to determine an output which is passed onto the next layer, until the output layer is reached where either a class label is determine or regression prediction made. During training the weights are adjusted to minimize the error between the predicted output and the calculated output from the network.

NNs are capable of modelling very complex relationships and have a good deal of customization in their architecture for specific problems. However, they are prone to overfitting, require lots of training data and time to adjust the weights as well as often being described as 'Black Box' due to them having near to no explainability.

For Datasets 1 and 2 a standard type of NN architecture as described earlier was used called a multi-layer perceptron (MLP) from the sk-learn library. For Dataset 3 a specialized type of NN called a Convolutional Neural Network (CNN) was implemented using the PyTorch library. Since it is an image dataset which can have complex features to capture, a heavier model than the MLP was chosen for this task.

3.4 ILP Algorithms

There are several ILP options that are integrated with pyswip (Python interface to SWI-Prolog enabling running of Prolog programs within Python). We decided to focus on the well-established Aleph system and the novel PyGol system to compare learning time performance.

Aleph uses mode declarations, which specify the input and output modes for predicate arguments in the background knowledge and examples [8]. Mode declarations provide information about the expected types and modes of the arguments, guiding Aleph in constructing appropriate clauses. Aleph starts the rule generation process by exploring the hypothesis space, which consists of all possible rules that can be generated given background knowledge and mode declarations. It employs a top-down search strategy, iteratively refining and expanding the initial set of hypotheses. The best rule or set of rules is finally selected based on the evaluation function.

PyGol generates rules similarly to Aleph's mode-directed inverse entailment system, however it does not require any mode declarations. It uses Meta Inverse Entailment to generate hypothesis clauses from a 'meta theory' which is automatically derived from background knowledge. No user intervention or parameter setting is needed.

3.5 Reinforcement Learning Algorithms

Reinforcement learning involves learning a policy of how to act in different states in an environment to maximise a reward. For both environments used, it was assumed that

there was no known model of the environment. Therefore, two algorithms were chosen that used Monte Carlo methods to sample the environment and use Temporal-Difference methods to learn an optimal policy. Two of the most popular methods for doing this are Sarsa and Q-learning. These algorithms are similar in that they calculate Q-values for all possible actions in every state, and then decide on an optimal policy by selecting the action with the maximum Q value. They also both use ε-greedy exploration to try different actions in each state, whereby the action selected will be taken according to the maximum Q-value with a probability 1 - ε or a random action taken with a probability ε. However, they differ in how they use a target policy when updating the Q-values. Sarsa is "on-policy" which means the Q-value for the successor state is selecting using the current policy, whereas Q-learning is "off-policy", so it just uses the maximum Q-value from the successor state. This difference can be seen in the update equations below:

Q update equation for Sarsa:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

Q update equation for Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Where Q represents the Q-value of the state and action combination, α is the step size taken for when updating Q, R is the reward and γ (gamma) is the weighting placed on the future state's Q-value.

This difference can result in the algorithms converging on different solutions. Sarsa converges on Q-values that are optimal when the same policy used in training will be used in future, whereas Q-learning converges on Q-values that are optimal for a policy which always selects the maximum of these. This means that Sarsa algorithms are less exploratory and learn more about the policy being followed whereas Q-learning is more exploratory and learns more about an optimal policy. Often this means that Sarsa performs better during the training, but Q-learning can perform better in testing. It also can result in Q-learning performing better in environments with sparse rewards where stringing together combinations of consecutive actions is important, but Sarsa can converge on an optimal policy faster in others.

These algorithms were implemented by initially setting all policy actions as 0 and then using these algorithms to update the Q-values every step each episode. The action taken at each step was determined using an epsilon greedy policy as described earlier, and epsilon was decayed using an exponential decay show in the equation below:

$$\varepsilon = 0.1 + 0.8e^{-(decay\ rate \times episode)}$$

4. Model evaluation / Experiments

4.1 Experiment 1

4.1.1 Null Hypothesis 1

Comparing the relative performance, effectiveness, and interpretability of the SVR, DT, and MLP models for a regression task of predicting housing prices in the Boston area. SVR and MLP methods are expected to have better generalization given the high dimensionality of the dataset.

4.1.2 Material & Methods 1

Dataset 1 is used to demonstrate the strengths and weaknesses of each model on a regression task. The following models, that are suited for non-linear relationships (as supported in Sec. 3), are selected: SVM, MLP and DT. The parameters for each of these algorithms are systematically set by using GridSearchCV for tuned hyper-parameters.

4.1.3 Results & Discussion 1

The models are evaluated firstly with MSE which measures the average squared difference between the predicted and actual values in a regression model. A lower MSE indicates better model performance, as it signifies smaller errors between the predicted and actual values. Secondly with R-squared which measures the proportion of variance in the dependent variable (target) that is explained by the independent variables (features) in a regression model. R-squared ranges from 0 to 1, where 0 indicates that the model explains none of the variance, and 1 indicates that the model explains all the variance in the target variable.

Results are collected and plotted with scatterplots demonstrating model fit.

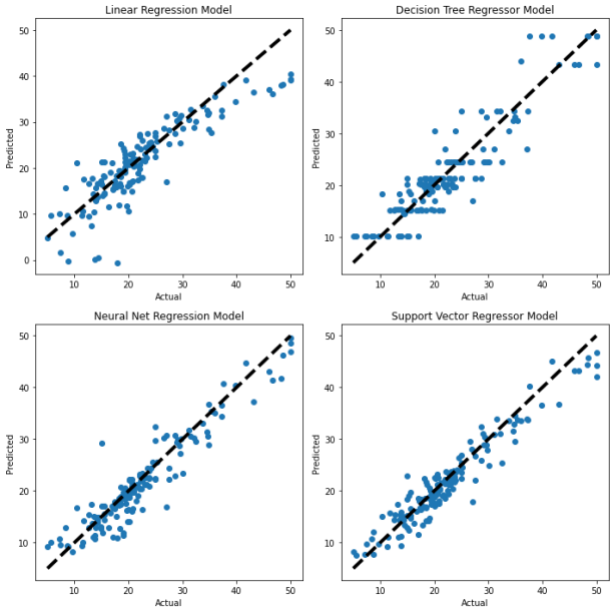

Figure 7: Model fit for each algorithm (including baseline linear regression model for comparison)

Model	Best hyper parameters	MSE	R-squared
DT Regressor	{'max_depth': 5, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 2}	13.83	0.83
MLP	{'hidden_layer_sizes': (128, 64), 'max_iter': 1000}	10.07	0.88
SVR	{'C': 10, 'epsilon': 0.5, 'gamma': 'scale'}	7.54	0.91

Table 1: Table summarizing generalization performance across each model.

In terms of **interpretability**, the DT model does much better than the MLP and SVM as it consists of a hierarchical structure of nodes and splits, making it easy to understand and explain the decision-making process. It allows us to identify important features and their impact on predicting housing prices. The complex architecture of the MLP model and high dimensional feature space built by the SVM model reduces the interpretability of these models.

The SVR achieves the highest R-squared score of 0.9122, indicating a strong ability to explain the variance in the target variable. It captures approximately 91.22% of the underlying patterns, suggesting excellent **generalization** performance. The MLP also gets a high R-squared score of 0.8827, only slightly less than SVR. SVR uses a kernel function to transform the data into a higher-dimensional space, while MLP has multiple layers of interconnected neurons. This flexibility allows them to capture more intricate relationships between features and the target variable. The DT model makes predictions based on simple rules according to its hierarchical structure. This structure may not capture complex relationships in the data, limiting its ability to explain the variance.

Additionally, the DT model provided **significant** insights into the Boston Housing dataset through the feature importances. They indicate the relative importance of each feature in the model's decision-making process. In this case, the top three most important features were "CRIM" (per capita crime rate by town), "NOX" (nitric oxides concentration), and "AGE" (proportion of owner-occupied units built prior to 1940). The high importance assigned to "CRIM" suggests that the crime rate has a significant impact on housing prices. This is supported by research on the topic [9].

4.2 Experiment 2

4.2.1 Null Hypothesis 2

This experiment looked to verify the null hypothesis that the DT based algorithm, random forest, would outperform SVM and MLP on a less complex dataset for a multiclass classification task where the data is imbalanced.

4.2.2 Material & Methods 2

As described in Section 2.2, the training data was experimented on to find out if feature merging or class balancing would improve model performance. As can be seen in Table 2, merging columns did not improve performance, although it had a very minor impact (<0.001) on the random forest model, it decreased the performance of the MLP model and significantly decreased the F1 score of the SVM model. This is likely because reducing the number of features decreased the complexity of the solution found by the algorithms, so they were worse at differentiating precisely between the number of occupants in the room. As simplifying the data did not improve the models, the original dataset was used for further training.

As per Table 3, both under sampling techniques decreased the accuracies and F1 scores of all the models but oversampling improved performance, most noticeably in the SVM models F1 score. This is likely because models overfit to the majority class during training on an imbalanced dataset, but with under sampling the decreased number of training instances hinders MLP and SVM. Oversampling helps the models optimize for all classes equally and increases the number of training examples. Random oversampling (ROS) and SMOTE both produced similar model performances. However, SMOTE has some advantages as it generates new training examples that aren't just copies of existing ones; this increases the variety of data which can help the models generalize. Therefore, SMOTE was selected as the class balancing approach for this dataset.

	Original Dataset		**Merged Temperature and Light Features**	
	Accuracy	F1 score	Accuracy	F1 score
Random Forest	0.996± 0.001	0.988±0.002	0.996± 0.001	0.987± 0.003
MLP	0.991± 0.001	0.969±0.005	0.987± 0.001	0.952±0.005
SVM	0.871± 0.003	0.394±0.005	0.826± 0.002	0.284± 0.008

Table 2: The average 5-fold stratified cross-validation accuracy and macro F1 score on the training data for the original dataset and one with merged features.

	Random Forest		**MLP**		**SVM**	
Technique	**Accuracy**	**F1**	**Accuracy**	**F1**	**Accuracy**	**F1**
None	0.996 ±0.001	0.988 ±0.002	0.991 ±0.001	0.969 ±0.005	0.871 ±0.003	0.394 ±0.005
ROS	1.000 ±0.000	1.000 ±0.000	0.988 ±0.001	0.987 ±0.002	0.906 ±0.008	0.905 ±0.008
SMOTE	0.999 ±0.001	0.999 ±0.000	0.989 ±0.002	0.990 ±0.002	0.905 ±0.008	0.904 ±0.008
RUS	0.990 ±0.006	0.990 ±0.006	0.972 ±0.013	0.973 ±0.012	0.558 ±0.098	0.483 ±0.114
Near miss	0.963 ±0.055	0.937 ±0.105	0.949 ±0.049	0.883 ±0.113	0.613 ±0.004	0.190 ±0.001

Table 3: The average 5-fold stratified cross-validation accuracy and F1 score on the training data for each of the algorithms with different class balancing techniques used.

For each of the 3 algorithms, a grid search with a 5-fold cross validation split was used to find hyperparameters that produced the best mean F1 cross-validation score as this was the most important metric for determining the models' performance across all classes.

For Random Forest the number of estimators, features used, max depth and minimum number of samples at each leaf and split were varied. This varied how complex each tree was as well as the variance between trees and the features used. The optimal hyperparameters were: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}. For MLP different hidden layer sizes, activation functions, optimization parameters and regression were tested. More complex architecture allows for more complex patterns to be found in the data but MLPs are prone to overfitting, different activation functions and regression can help with this. Finally different optimizers and learning rates, ensure the algorithm converges on an optimal solution during

training. The best parameters found were: {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (10, 10), 'learning_rate': 'adaptive', 'solver': 'adam'}. For SVM the kernel type, kernel coefficient and L2 regularization parameter were varied as different kernel transformations work better with different data and changing regularization allows the model to find the correct complexity of boundary for the data. The best parameters found were: {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}.

4.2.3 Results & Discussion 2

Table 4 shows the accuracy and macro averaged F1 scores achieved by each model with its optimal hyperparameters on the test data. All models achieve very good accuracies and F1 scores (over 98%), with random forest achieving the best, followed by SVM with MLP having a marginally worse F1 score. Random forest achieved 100% recall when classifying cases of 0 or 1 room occupant, and then made a small number (5) of misclassification between 2 and 3 people, which is a harder task and may reflect challenges faced when the size of people varies. This is an excellent performance and validates the possibility of accurately determining the number of occupants in a room using these sensors. Random forest needed the least tuning to achieve a high performance and had an F1 score of 99%, even when trained on the imbalanced data. Whereas SVM performed very poorly on the imbalanced data with an F1 score of 39%, showing that data preparation and optimization are very important for this algorithm. Along with handling imbalanced data better, random forest also offers more explainability, particularly than MLP, as the importances of each feature can be calculated using impurity-based feature importances. When this was done for the fully trained model it could be seen that the first 3 light sensors were the most important features followed by the CO_2 sensor. It is for these reasons that the null hypothesis was correct as random forest is the best algorithm for this dataset. In future studies it would be worthwhile testing removing features to see if a high accuracy could still be achieved with a lower number of sensors.

Model	**Test Accuracy**	**Test F1 score**
Random Forest	0.998	0.993
MLP	0.994	0.980
SVM	0.994	0.982

Table 4: Test accuracy and macro averaged F1 score for each model on the test dataset.

4.3 Experiment 3

4.3.1 Null Hypothesis 3

Comparing the performance of DTs, SVC, and CNNs for a binary classification task on an image dataset. Null hypothesis being “CNNs are more accurate and efficient at classifying images than DT and SVC due to their ability to capture complex image information”.

4.3.2 Material & Methods 3

Hypothesis 3 was put to test in the following series of sub-experiments. Various pre-processing techniques - Colour Histogram (described in section 2.3), Pixel Intensity, and Texture Features - were tested on a dataset with DTs and SVC. GridSearchCV was applied to the data from the technique that yielded the best results with a 5-fold cross validation. That was done to obtain the hyperparameters that produced the optimal performance for DT and SVC, that were `{'criterion': 'entropy', 'max_depth': 8, 'min_samples_leaf': 2, 'min_samples_split': 5}` and `{'C': 100, 'gamma': 1, 'kernel': 'rbf'}` accordingly.

Followingly, three experiments conducted to improve the performance of the CNN model for binary image classification of the leaf dataset. First, a baseline model was defined which consists of three convolutional layers with max pooling, followed by two fully connected layers. The model is trained using the Adam optimizer and the cross-entropy loss function, for 25 epochs. The training and validation losses and accuracies are recorded and plotted to visualize the training process. The results show that the model achieves a test accuracy of 0.8462 and an F1 score of 0.8512, with a training time of 155.41 seconds. In the first experiment, changes were made to the CNN architecture by increasing the number of filters in the convolutional layers, adding more convolutional layers, increasing the size of the fully connected layers, and adding dropout layers to prevent overfitting. However, the model's performance was erratic, and it failed to classify healthy leaves, leading to the conclusion that the original CNN model was better suited for the task. In the second experiment, colour channels were used instead of grayscale images, and the model performed best when using the isolated red channel. The third experiment involved hyperparameter tuning, where the optimizer was changed from Adam to SGD, and different learning rates were tested. However, the results showed that the Adam optimizer outperformed SGD in terms of accuracy and F1 score.

4.3.3 Results & Discussion 3

The Colour Histogram technique had good performance metrics with DT, demonstrating quick training time, high F1-Score, and accuracy, likely due to its simple, compact image representations. The Pixel Intensity function had a long training time and worse performance metrics with DT. However, its performance improved with SVC after reducing the image size, as SVC handles high-dimensional data well. Texture Features offered faster processing times but worse performance metrics, possibly due to its reliance on local image patterns. As DT performed better with Colour Histograms and SVC's performance was only marginally better with Pixel Intensity, the researcher decided to use Colour Histograms for further tests.

	DT			**SVC**		
Technique	Acc	F1 Score	Time (s)	Acc	F1 Score	Time (s)
Colour Histogram	0.813	0.812	0.147	0.754	0.753	2.621
Pixel Intensity	0.712	0.711	10.134	0.783	0.783	25.984
Texture Features	0.666	0.666	0.057	0.616	0.608	0.057

Table 5: Summary of Image processing experiments

The results of the CNN sub-experiments are summarized in the table below. Only the best result from Experiment 3 is displayed here. The best CNN model works when the images are reduced to only the red colour channel using the Adam optimizer.

Model	Accuracy	F1 Score	Runtime (s)
Baseline	0.8462	0.8512	155.41
Exp. 1	0.5204	0.6846	452.83
Exp. 2 R	**0.863**	**0.8665**	**153.04**
Exp. 2 G	0.857	0.8615	152.97
Exp. 2 B	0.8329	0.8386	153.04
Exp 3.	0.8474	0.849	556.02

Table 6: Summary of CNN performance experiments

Model	Accuracy	F1 Score	Runtime (ms)
DT	0.8431	0.8429	30.58
SVC	0.8679	0.8678	121.73
CNN	0.863	0.8665	15304

Table 7: Best results from the 3 different models

Based on these experiment results, the null hypothesis is rejected. SVC performed the best at image classification on the leaf dataset with the highest accuracy and F1 scores. It was also more efficient than the CNNs which took relatively longer to train. SVCs, can often perform very well on certain types of datasets and have shorter training times due to their algorithmic efficiencies.

Additionally, the CNNs did not perform to their expected potential due to the hardware limitations of Google Colab. Given more GPU RAM, images with all 3 colour channels could be used with the original size of 600 x 400 pixels. This would challenge the rejection of the null hypothesis upon successful training. Pre-trained models like EfficientNet can also be used instead of defining a custom CNN which would vastly improve the accuracy.

4.4 Experiment 4

4.4.1 Null Hypothesis 4

Comparing rule-learning time for different Inductive Logic Programming systems. The null hypothesis being: "There is no significant difference in the convergence ability of Aleph and PyGol to generate a single rule that satisfies all the training examples."

4.4.2 Material & Methods 4

The materials extracted from the UW-CSE dataset as described in Sec.2 form the background knowledge (bk_train.pl), the training examples (pos_example.f and neg_example.n) and mode declarations (aleph-modes.pl) required for this experiment. The Aleph model was built using PyILP with 25 positive and 25 negative training examples, mode declarations extracted from aleph-modes.pl [ref] and a train-test split of 75:25. The PyGol model is built with the same background knowledge and training example split as with Aleph. The background knowledge, positive examples, and negative examples are first converted into bottom clauses to represent the information in a format suitable for the PyGol model. Bottom clauses are logical statements in first-order logic that are made up of atoms (predicates with arguments) and logical connectives (such as conjunctions or disjunctions). The PyGol system uses these to define the hypothesis space where it can explore and find the best ruleset that explains the data.

The following ruleset is generated by Aleph:

```
['advisedBy(person272,person7).advisedBy(person276,person407).advisedBy(person228,person342).advisedBy(person380,person79).advisedBy(person183,person5).advisedBy(person239,person171).advisedBy(person118,person5).
advisedBy(A,B) :-taughtBy(C,B,D), ta(C,A,E).
advisedBy(person263,person5).advisedBy(person142,person342).advisedBy(person300,person342).
advisedBy(A,B) :-tempAdvisedBy(C,B), inPhase(A,D).advisedBy(person113,person342).advisedBy(person362,person5).advisedBy(person435,person279).']
```

The PyGol model outputs a single ruleset instead:

```
['advisedBy(A,B):-inPhase(A,C),yearsInProgram(A,D),sameperson(B,B),professor(B),sameperson(A,A)']
```

Model	Accuracy	F1 Score	Runtime (s)
Aleph	0.643	0.444	9.75
PyGol	1.000	1.000	1.75

Table 8: Comparison of accuracy, f1 score and runtime across both ILP systems

The learning time for each model was compared using 5 different number of training examples: [10, 20, 30, 40, 50]

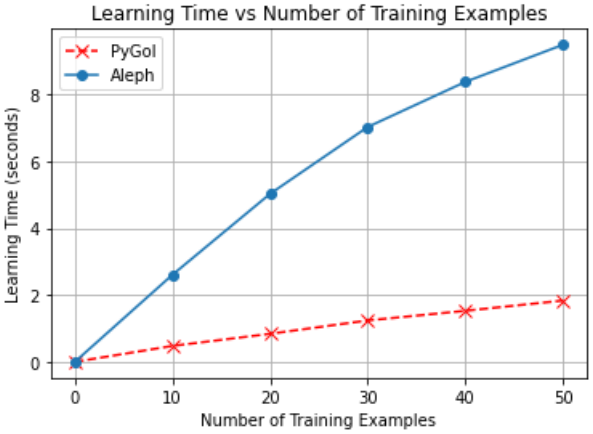

Figure 8: Timing of Aleph vs PyGol

Both models successfully generate rules with 100% accuracy during training but Aleph fails to converge to a single rule that satisfies all the training examples while PyGol does. The time taken for rule generation is heavily impacted by how many examples the system is trained on. Both models show a clear linear increase in learning time as the number of training examples increases. This shows that the search space for possible hypotheses grows linearly as the number of training examples is increased. PyGol converges quicker than Aleph, perhaps due to Aleph's extra processing of mode declarations which PyGol does not require. When evaluating the generalization of the models

on a test dataset, PyGol maintains 100% accuracy and F1 score whereas Aleph misses a lot of the positive examples resulting in an accuracy of 64% and F1 score of 44%. This demonstrates that PyGol's Meta Inverse Entailment may be more suited to the UW-CSE dataset rather than Aleph's mode-directed approach. The experiment also shows the PyGol model has better interpretability as it outputs a single rule whereas Aleph has outputted 2 different rulesets for different subsets of the data.

4.5 Experiment 5

4.5.1 Null Hypothesis 5

We expect Q-learning to converge on a better optimal policy but Sarsa to perform better in training as Q-learning optimises the Q-values using an "off-policy" method whist Sarsa uses "on-policy".

4.5.2 Material & Methods 5

For both environments 5000 episodes were selected as the training length after preliminary testing with default hyperparameters. Then step size, gamma and epsilon decay rate were systematically changed to increase convergence rate, whilst still achieving maximum accuracy from the final policy. A larger step size can help the algorithm converge on a solution faster as it updates the Q-values by larger amounts each visit but it can also cause issues if a bad policy updates the Q-value incorrectly. For example, in the Taxi environment dropping someone off in a correct zone has a positive reward of +20 but completing an incorrect action like trying to pick someone who is absent has a negative reward of –10. Both algorithms worked best with high gamma values, as correct chains of subsequent actions are very important in both problems. Finally, the epsilon decay rate affects the amount of exploration that takes place as time goes on. Higher values of epsilon result in more exploration but if there is too much exploration the algorithm will not settle on an optimal policy. The right balance between this is required to converge quickly whilst still finding an optimal policy. The optimal hyperparameters for each algorithm on each environment can be seen in Table 9. The algorithms were then trained for 5000 episodes and the optimal policy from each tested.

Environment	Q-Learning			Sarsa		
	Step size	Gamma	Epsilon Decay Rate	Step size	Gamma	Epsilon Decay Rate
Taxi	0.9	0.95	0.1	0.2	0.95	0.001
Cliff Walking	0.9	0.95	0.001	0.1	0.95	0.001

Table 9: Optimal hyper-parameters for each reinforcement learning algorithm.

4.5.3 Results & Discussion 5

For the Taxi environment the optimal policy from each algorithm was tested on the environment 10,000 times, as the environment varies between runs, to determine a success rate and average reward. Both policies achieved a 100% success rate and average rewards that are within their standard error as can be seen in Table 9. The difference between the two in this problem was the speed of convergence; Q-learning converged on an optimal solution in approximately 500 training episodes whereas Sarsa took over 3000, this can be seen in Figure 9.

As the Cliff Walking environment was deterministic the final policy for each algorithm gave the same reward every run. In contrast to the previous environment, the average reward for the optimal policy was better for Q-learning than for SARSA. Moreover, the speed of convergence of the Q-learning model was again larger than SARSA as the models converged at 750 and 1250 training episodes accordingly.

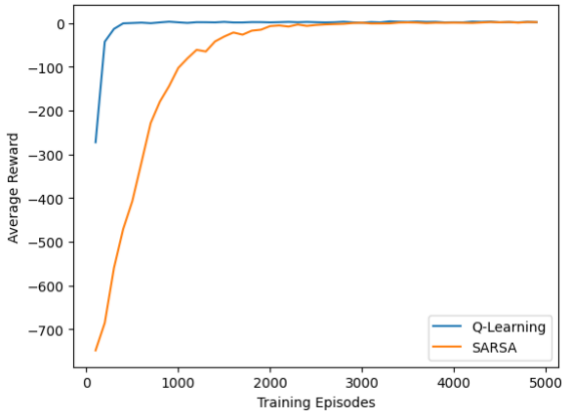

Figure 9: Average reward during training, calculated every 100 episodes on the Taxi environment.

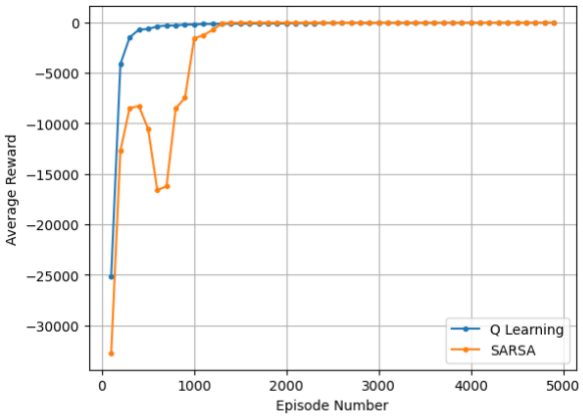

Figure 10: Average reward during training, calculated every 100 episodes on the Cliff walking environment.

Environment	**Q-Learning Average reward**	**Sarsa Average Reward**
Taxi	7.88±0.08	7.78±0.08
Cliff Walking	-13	-17

Table 10: Average reward for the two reinforcement learning algorithms' final polices on each environment.

The null hypothesis was rejected by the Taxi experiment as Q-learning did not find a substantially better final policy and Sarsa did not perform better in training. Sarsa performed worse on the Taxi environment as incorrect or illegal future actions chosen randomly due to the epsilon greedy policy would negatively impact the Q-value of a state. This state may have had a much higher Q-value if the correct future action was taken. To counter this effect, it is necessary to have a small step size. As explained in Section 3.5, Q-learning does not have this problem as the best Q-

value from the successor state is used when updating the current state's Q-value. The larger step size also meant Q-learning could use a higher decay rate, whereas Sarsa needed epsilon to decay slower as it required more exploration to converge on optimal solutions. As Sarsa could only converge with a small step size and decay rate, its convergence rate was much slower; as seen in Figure 9.

In the Cliff walking environment, walking near the edge of the cliff has a much lower Q-value when using the Sarsa algorithm as occasions when the epsilon greedy policy causes the algorithm to fall the cliff the successor state gives a reward of -100. This means that the final policy takes a longer route to avoid walking along the edge of the cliff. This does not happen for Q-learning as the final policy assumes that the best action is taken when near the cliff, which does not involve stepping off. So, a shorter but riskier route is taken. It also converges faster for the same reasons as the Taxi environment. The null hypothesis was supported somewhat in this experiment as the final policy was better for Q-learning, however, it was not supported in that Sarsa did not outperform Q-learning during training.

5. Discussion of the results, interpretation and critical assessment

The experiments involving datasets 1-3 showed the strengths and weaknesses of DT, SVM and NN models. DTs excelled in interpretability whereas SVM and NN demonstrated strong generalization overall. Using a bagging approach with DTs in experiment 2 allowed them to outperform the other models but only due to the imbalanced nature of the dataset which the Random Forest handles well. The SVM and NN, although performing strongly in both regression and classification, are held back by long training times and computational needs as shown with the image dataset experiments.

The experiment on ILP demonstrated that the novel Meta Inverse Entailment approach of PyGol is well-suited to learning the patterns in the (sampled) UW-CSE dataset. Research shows that Aleph is capable of achieving 84% accuracy on UW-CSE [10] so the poor performance shown in experiment 4 (64% accuracy) is most likely due to the much smaller size of the testing set.

The findings from the reinforcement learning experiment show that Q-learning converges faster than Sarsa and can in some environments find a better final policy, so is likely the better algorithm to use in most situations. However, the final policy in Cliff walking found by Sarsa was a safer solution that considered random variance in actions from the epsilon greedy policy. Therefore, although Q-learning may be better in most cases there is still an argument to use Sarsa in situations where the target policy needs to mimic some of the characteristics of the training policy. It also may produce solutions that are safer, for example if you wanted to train a robot to take a path through a dangerous environment, Sarsa could consider random events that may cause the robot to deviate from the path.

6. Conclusions

In summary, the experiments elucidate the strengths and weaknesses of different machine learning models, providing insights into their interpretability, generalization capabilities, training times, and computational needs. The ILP and reinforcement learning experiments demonstrated the suitability of specific approaches for specific datasets and scenarios, offering valuable considerations for future research and practical applications.

References

[1] CVUT. UW-CSE dataset. Retrieved from https://relational.fit.cvut.cz/dataset/UW-CSE

[2] Dany Varghese, Didac Barroso-Bergada, David A. Bohan and Alireza Tamaddoni-Nezhad,
Efficient Abductive Learning of Microbial Interactions using Meta Inverse Entailment, In Proceedings of the 31st International Conference on ILP, Springer, 2022(accepted).

[3] Farama Foundation. (2023). Gymnasium. GitHub Repository. Retrieved from https://github.com/Farama-Foundation/Gymnasium.

[4] Harrison, D. and Rubinfeld, D.L. `Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.

[5] Adarsh Pal Singh, Vivek Jain, Sachin Chaudhari, Frank Alexander Kraemer, Stefan Werner and Vishal Garg, Machine Learning-Based Occupancy Estimation Using Multivariate Sensor Nodes, in 2018 IEEE Globecom Workshops (GC Wkshps), 2018.

[6] Amanda, M. (2021). Healthy vs. Diseased Leaf Image Dataset. Kaggle. Retrieved from: https://www.kaggle.com/datasets/amandam1/healthy-vs-diseased-leaf-image-dataset

[7] UW-CSE dataset [GitHub repository]. Retrieved from https://github.com/logic-and-learning-lab/ilp-experiments/tree/main/ilpexp/problem/uwcse

[8] Srinivasan, A. (2001). The Aleph Manual (http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/)

[9] Raya, Josep Maria & Montolio, Daniel & Buonanno, Paolo. (2012). Housing Prices and Crime Perception. Empirical Economics. 45. 10.1007/s00181-012-0624-y.

[10] França, M.V.M., Zaverucha, G. & d'Avila Garcez, A.S. Fast relational learning using bottom clause propositionalization with artificial neural networks. Mach Learn 94, 81–104 (2014). https://doi.org/10.1007/s10994-013-5392-1