MLDM 2021 Coursework Group: *NSync*

Aditya Mohan Vichare av00550@surrey.ac.uk
Debjyoti Saha ds01290@surrey.ac.uk
Gururaj Nimbalkar gn00242@surrey.ac.uk
Kowshik Kesavarapu kk01002@surrey.ac.uk
Shweta Negi sn00890@surrey.ac.uk

Abstract

Over the years, we have developed many Machine Learning algorithms which are compared with one another for the purpose of analyzing which algorithm is better suited for which type of dataset/problem statement. Two datasets were taken for comparing different ML models. They were then analyzed using different algorithms including Decision Tree, Perceptron, Naive Bayes , SVM and DQN . For understanding how logic based algorithms work we used Metagol and Aleph and compared them with each other. The comparison between the algorithms is shown with the help of different metrics like , F1-score, Accuracy, Precision and Recall. It was observed that all the classic algorithms worked better with our chosen two data sets. Aleph took less time to run while metagol produced the same results but it took more time for metagol to run than aleph. DQN took far more time and produced byfar the less desirable results.

1. Project Definition

The first dataset is the Kidney Disease dataset. The dataset was taken over a period of 2 months in India. It has 25 features as shown below The objective is to classify Kidney Disease as Chronic(ckd) and not Chronic(notckd). Based on the 400 data values being fed to various Supervised ML algorithms, the model then can predict whether the disease is chronic or not when given a new set of values.

Second Data set is related to Pokemon . The main objective is to predict whether a pokemon is legendary or not . There are 12 features in this data set . This is binary classification .

2. Data Preparation

We have in total used 2 datasets to satisfy the above selected algorithms. The description of the datasets are as follows:

1. Chronic Kidney Disease

The chronic kidney disease dataset is the collection of 2-month kidney disorders over 25 different features. The main features include, (eg, RBC, WBC, etc.), the target column of the following dataset is ‘ckd’ or ‘notckd’ which are the abbreviations for ‘chronic kidney disease’ and ‘not chronic kidney disease’.

2. Pokemon

The Pokemon dataset has a record of 800 different pokemon and their details such as names , type and attack damage.There are 12 features in this data set including the index and target. The main objective is to predict whether the pokemon is legendary or not. This data set is chosen to evaluate how a model would react on skewed data. Here the positives are just 65 in 800 so it would be rather interesting to see how different models would react to this kind of data.

2.1 Data cleaning and exploration.

- **Chronic Kidney Disease**

1. It is observed while evaluating the dataset that the column names for short forms are quite confusing for the developers and can be renamed to make it more user-friendly.
2. After analyzing the data types, we observed that some columns like, “red_blood_cell_count”, “packed_cell_volume”,“white_blood_cell_count ” are of object type and the inserted data are numbers. We converted these columns into numerical data types.
3. The “id” column was found as a unique identifier for each row of the dataset and this won’t help us to discover any insights from the data.
4. The next step was to extract numerical and categorical features. After doing that we observed that two features and the target variable contain some illogical values.
5. Checking the feature distribution, we came to know that the data is distributed among all the factors like age, blood pressure, glucose, and

other specific content required to determine the disease.

6. Categorical data and label distribution graphs tell us about the normality and abnormality, present and absent status, yes and no factor for hypertension, and good and poor for appetite. Some of those are RBC, bacteria, coronary artery disease, anemia, etc.
7. Positive and negative correlation matrices are plotted with respect to classes.Here we can see there are few features that have higher correlation to the target variable. Mainly specific gravity, hemoglobin and packed cell volume.

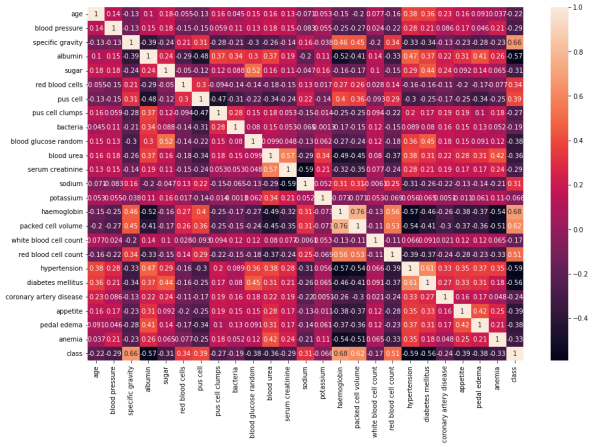


Fig 1: Correlation matrix on Dataset 1

- **Pokemon**

1. While evaluating the Pokemon dataset, we observed that it contains various columns of numbers. One of the columns from the dataset, “Type 2” was having 386 null values. This is because only a few pokemon belong to two types . so we replaced the empty data with NONE..
2. We plotted a pie chart to look at the distribution of class types. We can see that normal type pokemons are having 12.3% of the total dataset, and 14% water type pokemon.

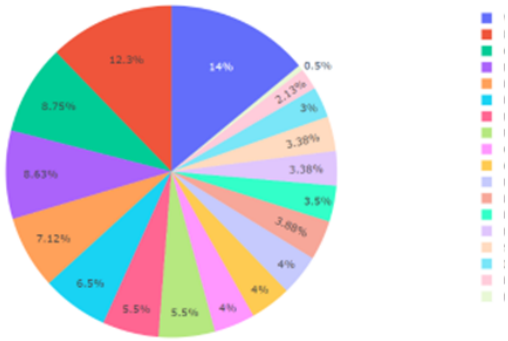


Fig 2: Class Distribution of Dataset 2

3. The observation of the dataset let us know that the data is very skewed, so taking a sample of 100 is taken.
4. We plotted a heatmap earlier to identify a better relation between features and target class. we can see there is only one feature(Total) has 0.5% and all the others have a good spread of correlation.

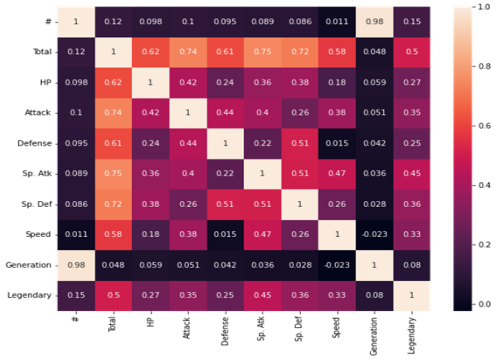


Fig 3: Correlation matrix on Dataset 2

2.2 Data Visualization

Dataset 1: Chronic Kidney Disease

1. Histogram was used to understand the distribution of data for each attribute

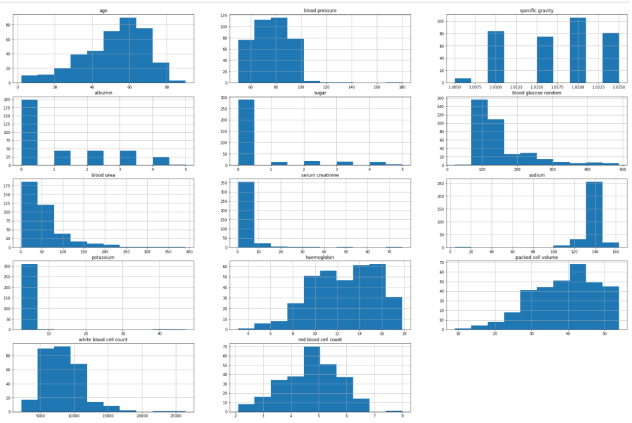


Fig 5: Attribute Histogram

2. Shown below is the target class distribution. Based on the graph below we observe that the dataset includes more entries of “ckd” in comparison to “notckd”

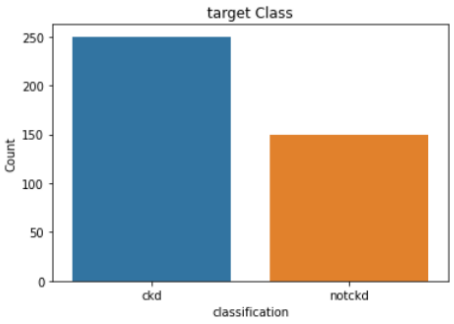


Fig 6: Chronic & Not Chronic comparison

3. Below graph represents the correlation of features and their impact on the target class. The figures are for 'red blood cell count' and 'Hemoglobin'.

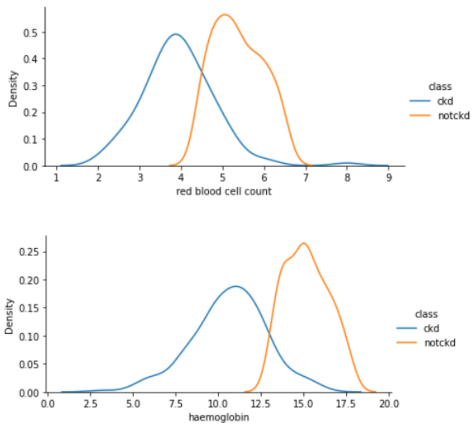


Fig 7: Hemoglobin and RBC graph

Dataset 2: Pokemon

- According to the dataset, the top level pokemons are rated as legendary. So to know the counts of legendary we plotted a bar graph to compare

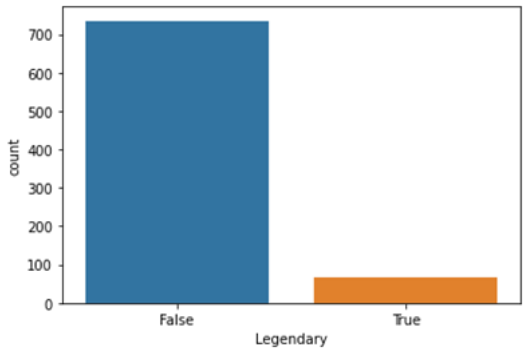


Fig 8: Type- Legendary Pokemons

Model development

Various algorithms have been implemented and tested for the 3 chosen datasets. The choice of algorithm and hyper parameter settings in the development are explained below.

2.1 Decision Tree

Library: sklearn

Module: sklearn.tree

Decision trees work by initiating a number of nodes which act as branches and entropy details as leaves. When a data is passed through the classifier it predicts the category of it and tries to locate it to the right node.

Decision tree classifiers are effective in the datasets which are classification based. The most efficient feature of the decision tree is you can handle the imbalanced dataset with the decrease in accuracy due to vacant values.

Similarly, changing in the training costs more, a minor change in the dataset leads to large amounts of dataset structure changes.

Hyperparameter settings:

1. Criterion: gini, entropy

The quality of split-ness is measured by the use of this parameter. The formulas for Gini and entropy are different from each other, entropy is a complex computational criterion in which it uses more resources compared to gini. Gini is relatively easier, the range specified for gini entropy is [0, 0.05] whereas the range of entropy is [0,1]

2. max_depth

This parameter is used to determine the maximum depth a decision tree can have in a classification problem. When a parameter like, "None" is passed, max_depth is more extensive without any limits.

3. min_samples_leaf

This parameter is used to decide the minimum number of leaves that are required to be present in each node of the decision tree. It is passed as per the following syntax: range(1, 5), range(0,5), range(10,20).

4. min_samples_split

The parameter refers to minimum samples that are required to split the tree.

2.2 Naive Bayes

Library: sklearn

Module: sklearn.naive_bayes

Naïve Bayes' methods are one of the sets for supervised learning. The algorithm works on conditional independence of features contained in every pair. One of the advantages of naïve bayes are, it predict the rate of probability very fast. It is recommended that when the training data is less naïve, Bayes performs better than any algorithm.

Naïve bayes also has limitations such as it works efficiently on categorical data, it jumbles when it is used in numerical data. It has a default assumption that it should assume that every feature is independent which is not in every case.

Hyperparameter settings:

1. alpha

This parameter works on the principle of Laplace transforms, which states the smoothening of parameters in naïve bayes. This is used for the explanation for the case of zero probability. Higher the value of alpha, probability will be pushed towards 0.5.

2. fit_prior

This is a Boolean parameter which is used to determine whether the algorithm should learn prior or not of the class probability.

2.3 **Support Vector Machines**

Library: sklearn

Module: sklearn.svm

SVM classifier is used when the dataset has N number of planes inside it. Planes refers to the number of features available in the dataset. So, if the dataset has 5 features, then the SVM classifier will create 5 planes to determine the accuracy. The objective of this classifier is to create hyper planes which are at the farthest distance from the data points.

The advantage of SVM is to make it highly effective on the datasets which are having large features. It works efficiently when a particular dataset has a margin level very clear.

SVM doesn't perform well in large datasets as it is requiring high computational power. If the dataset has a greater number of noise features, SVM will confuse itself and will not predict the correct value.

Hyperparameter setting:

1. C

This is a regularization parameter. The parameter C is inversely proportional to the data points available in the dataset. The greater the value of C the smaller is the hyperplane.

2. Gamma

This is the inverse radius of support vector samples that are selected.

3. Kernel

This parameter gives the option to the developer of which type of hyper plane is to be created. If linear is passed, a linear hyperplane is created else non-linear hyperplane is created.

2.4 **Perceptron**

Library: sklearn

Module: sklearn.linear_model

A neural network with a single layer of neuron is called a perceptron. Perceptron is a linearly classified neural network. The working follows by collecting the input data and predicting the output data. It initiates with the assignment of some weights to the input data and adding a bias to it.

Training the data with ease is the main advantage of perceptron. It is easy to implement in any programming language as it has no hidden layer of neurons. The limitation of perceptron is that it is only applicable to the data which are linearly classified. To overcome this limitation, MLP is used to get the desired output of the data which are not linear.

Hyperparameter settings:

1. Eta0

This is a learning parameter. Constant value which multiplies with any update feeded.

2. Random_state

Used to shuffle the training data

3. Max_iter

This defines the number of epochs needed. Epochs are the iterations which are needed to learn the training data for the algorithm.

2.5 DQN Algorithm

Library: imbDRL

Version:2021.1.26.1

Q-Learning is a Reinforcement Learning that focuses on maximizing rewards by training the agents to work on certain tasks and take actions on particular states in a given environment. Since, it becomes very complicated to derive Q-value for each state, it is always combined with neural networks. We predict that this would be the least performing model of the bunch as reinforcement learning is not good at classification and it can perform well in classification provided there is a large amount of data. As the datasets we choose are quite small we predict this model will produce less than ideal results

Hyperparameter setting:

Episodes -

Number of iterations to run . it can range between 100 to 1000000

Warmup -

This parameter specifies how many iterations to be run before training to collect some sample data.

Batch size -

This parameter specifies batch size for each iteration

Learning Rate -

This parameter specifies the learning rate of each episode

2.6 Logic Based and Relational learning.

For this part we choose to work with Michalski's trains problem and run it using ALeph and Metagol and compare them both but it gave almost same results then we decided to work with Mutagenesis data set and we extracted the facts using the lab sample code and compared them to classic algorithms decision tree and Naive Bayes .

3. Model evaluation / Experiments

Various experiments using different algorithms have been implemented in all the 3 datasets which are reflected in the output produced. We produced a null hypothesis which clearly explains the experimental setups and the results obtained. The results are discussed in the below sections.

In the first two experiments, a minimum of 4 algorithms is implemented. Each of the selected algorithms are trained again using the best parameters to improve the accuracy rate. The performance of each model is compared with respect to metrics such as, confusion, ROC, accuracy, F1 score, and precision.

Reinforcement Learning is the main objective of the third experiment as it was quite different and we want to evaluate it a little differently.

Final experiment is based on Logic based algorithms here we used different datasets for this as they require a dataset in different format..

3.1 Experiment 1

3.1.1 Null Hypothesis 1

Since chronic kidney disease (Dataset 1) is a supervised learning dataset,We predict that tree-based algorithms will be more effective than the other algorithms.

3.1.2 Material & Methods 1

The datasets were divided into train and test data with a split of 75/25. Hyperparameter tuning is also done on training data. Once the best parameters are inserted, the output accuracy was compared against all the performed algorithms. The process of comparing was carried out by stratified k fold technique. The algorithms and hyperparameters for the dataset with the desired output are described below:

Dataset 1: When no hyperparameters were passed

- **Decision Tree**

  Accuracy- 98%

  Criterion- entropy

  Max_depth- None

  Min_samples_split- None

  Min_samples_leaf- None
- **Multinomial Naive Bayes**

  Accuracy- 88%

  Alpha- None

  Fit_prior- False
- **Support Vector Machine**

  Accuracy- 97%

  C-100

  Gamma- default

  Kernel- None
- **Perceptron**

  Accuracy- 99%

Eta0-0.1

Random_state- 0

Max_iter- 0

While testing the dataset, we found 1 misclassified examples

Dataset 1: When hyperparameters were passed

- **Decision Tree**

  Accuracy- 97%

  Criterion- entropy

  Max_depth- 7

  Min_samples_leaf- 5

  Min_samples_split- 5

- **Multinomial Naive Bayes**

  Accuracy- 85%

  Alpha- 0.2

  Fit_prior- True

  Class_prior- None

- **Support Vector Machine**

  Accuracy- 97%

  C- 25

  Gamma- scale

  Kernel- rbf

- **Perceptron**

  Accuracy- 95%

  Eta0- 0.1

  Random_state- 1

  Max_iter- 100

  While testing the dataset, we found 5 misclassified examples

3.1.3 Results & Discussion 1

Varieties of evaluation metrics such as Accuracy, Precision, Recall, and F1 score were calculated for each algorithm performed on Dataset 1. These algorithms are performed according to two cases established as, “Performing algorithms without any hyperparameters” and “Performing algorithms with the hyperparameters' '.

Table 1 elaborates the values which are obtained by passing no hyperparameters and Table 2 elaborates the second case of our dataset, which is executing with the hyperparameters.

Table 1: Comparison of the different evaluation metrics without hyperparameters on dataset 1 for each algorithm

Algorithm	Precision	Recall	F1
Decision Tree	95%	100%	98%
Naïve Bayes	77%	100%	87%
SVM	97%	95%	96%
Perceptron	99%	100%	99%

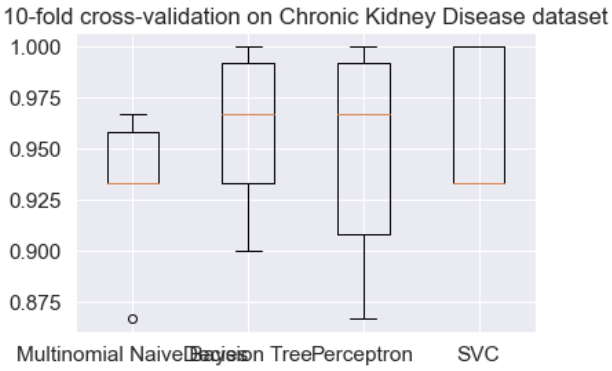


Fig 10: Comparison of metrics without hyperparameters on Dataset 1

Table 2: Comparison of the different evaluation metrics with hyperparameters on dataset 1 for each algorithm

Algorithm	Precision	Recall	F1
Decision Tree	95%	98%	96%
Naïve Bayes	73%	100%	85%
SVM	97%	95%	96%
Perceptron	91%	98%	94%

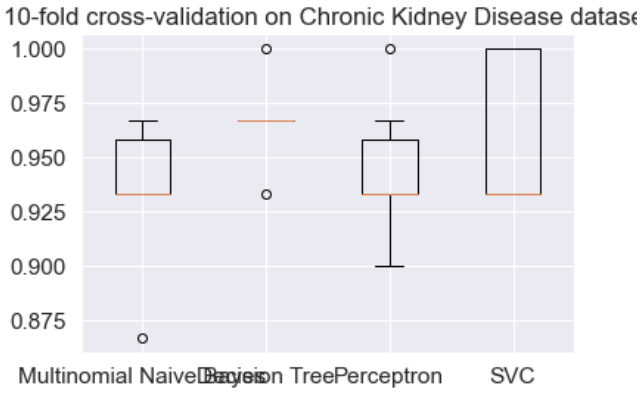


Fig 11: Comparison of metrics with hyperparameters on Dataset 1

After evaluating with and without the hyperparameters our hypothesis is proven to be false as the scores after passing the hyperparameters are considered to be less compared to the ones without hyperparameters.

3.2 Experiment 2

3.2.1 Null Hypothesis 2

For an imbalanced dataset of Pokemon if sampling is not done all the algorithms will be bad at evaluating when only positive samples are fed.

3.2.2 Material & Methods 2

The datasets were divided into train and test data with a split of 75/25. Hyperparameter tuning is also done on training data. Once the sampling was done successfully, the output accuracy and other required values are compared against all the performed algorithms. The process of comparing was carried out by stratified k fold technique. The algorithms and hyperparameters for the dataset with the desired output are described below:

Dataset 2: Trying without sampling the dataset and checking the overfitting issue

- **Decision Tree**

  Accuracy- 98.5%

  Criterion- entropy

  Max_depth- 7

  Min_samples_leaf- 5

  Min_samples_split- 5
- **Naive Bayes**

  Accuracy- 46.5%

  Alpha- None

  Fit_prior- False

  Class_prior- None
- **Support Vector Machine**

  Accuracy- 96%

  C- 100

  Gamma- default

  Kernel- non-linear
- **Perceptron**

  Accuracy- 93.5%

  Eta0- 0.1

  Random_state- 1

  Max_iter- 0

  While testing the dataset, we found 13 misclassified examples

Dataset 2: Predicting the legendary pokemons with our trained models without random sampling

- **Decision Tree**

  Accuracy- 96.9%

  Criterion- entropy

  Max_depth- None

  Min_samples_leaf- None

  Min_samples_split- None
- **Naive Bayes**

  Accuracy- 95.3%%

  Alpha- None

  Fit_prior- False

  Class_prior- None
- **Support Vector Machine**

  Accuracy- 0%

  C- 100

  Gamma- default

  Kernel- non-linear
- **Perceptron**

  Accuracy- 100%

  Eta0- 0.1

  Random_state- 0

  Max_iter- 0

Dataset 2: Using random sampling to avoid overfitting

- **Decision Tree**

Accuracy- 90.4%

Criterion- entropy

Ccp_alpha- 0.045

- **Naive Bayes**

  Accuracy- 59.5%

  Alpha- None

  Fit_prior- False

  Class_prior- None

- **Support Vector Machine**

  Accuracy- 80.95%

  C- 100

  Gamma- default

  Kernel- non-linear

- **Perceptron**

  Accuracy- 81%

  Eta0- 0.1

  Random_state- 1

  Max_iter- 0

  While testing the dataset, we found 8 misclassified examples

3.2.3 Results & Discussion 2

Varieties of evaluation metrics such as Accuracy, Precision, Recall, and F1 score were calculated for each algorithm performed on Dataset 2. These algorithms are performed according to two cases established as, "Trying without sampling the data and checking the overfitting issue", "Predicting the legendary pokemons with our trained models without random sampling", "Using random sampling to avoid overfitting issue".

Table 1, Table 2, and Table 3 elaborates the values of Precision, F1, Recall, and Accuracy with respect to the above mentioned criteria.

A separate table of ROC- AUC is also included for better clarification.

Table 1: Overfitting issue without sampling the data

Algorithm	**Precision**	**Recall**	**F1**
Decision Tree	99%	99%	99%
Naïve Bayes	96%	43%	59%
SVM	98%	98%	98%
Perceptron	97%	96%	96%

We can see that the accuracy is significantly higher due to the overfitting of false values in our dataset. So, we assume that the model is trained in such a way that it predicts false values even for the true ones. The number of legendary pokemons are significantly less so even if the predictions for those are wrong the model will still have better accuracy as it will still predict the non legendary pokemons correctly. This is an example of an overfitting issue. But such a model is not a good fit.

Table 2: Legendary Pokemons without random sampling

Algorithm	**Precision**	**Recall**	**F1**
Decision Tree	100%	97%	98%
Naïve Bayes	100%	95%	98%
SVM	0%	0%	0%
Perceptron	100%	100%	100%

As you see the accuracies are still better after taking just the legendary data set for testing. This can be because the model prediction depends upon different characters as the number of features are quite large. Our models were not over-fitting except SVM..

Table 3: Values after random sampling

Algorithm	**Precision**	**Recall**	**F1**
Decision Tree	95%	84%	89%
Naïve Bayes	56%	75%	64%
SVM	83%	75%	79%
Perceptron	83%	75%	79%

A table of ROC-AUC is given below which describes the dataset and its features in a better way.

Feature	AUC
Total	96.8%
Sp. Atk	89.3%
Sp.Def	84.7%
Attack	83.4%
Speed	83%

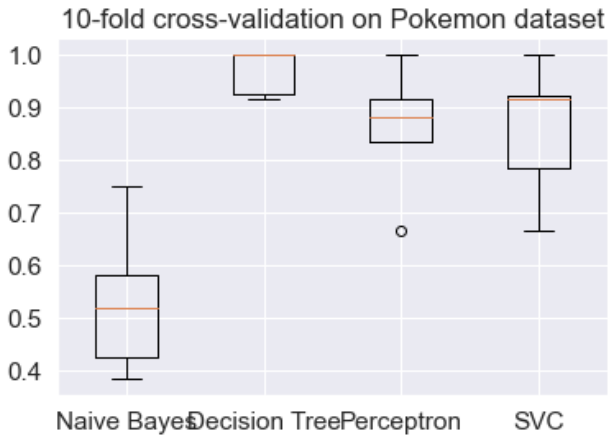


Fig 12: Comparison of metrics with sampling on Dataset 2

Experiment 3

3.3.1 Null Hypothesis 3

DQN will perform with very less accuracy when compared to other classic models and also it takes significantly more time to produce the results .

3.3.2 Materials and Methods 3

For this experiment we used Imbdrl which is a model developed by Thijs van den Berg. It is a classification model which uses Deep Q-Network to solve a classification problem . This is originally designed to perform classification on an imbalance data set but can also be used for any type of binary classification. We ran the two data sets with several hyper parameters and the results for those are described and evaluated below.

3.3.3 Results and Discussion

Scenario	Data set	Hyper Parameters	Results
1	Kidney	Episodes- 10000 Warmup - 10000, Batch Size- 10 Learn Rate - 0.005	Acc - 0.55 F1: 0.54, Presc:0.37 Recal': 1.0
2	Kidney	Episodes- 100000 Warmup - 100000, Batch Size- 25 Learn Rate - 0.05	Acc - 0.55 F1: 0.54, Presc:0.37 Recal': 1.0
3	Kidney	Episodes- 10000 Warmup - 10000, Batch Size- 50 Learn Rate - 0.005	Acc - 0.55 F1: 0.54, Presc:0.37 Recal': 1.0
4	Pokemon	Episodes- 10000 Warmup - 10000, Batch Size- 10 Learn Rate - 0.005	Acc - 0.77 F1: 0.89, Presc:0.81 Recal': 1
5	Pokemon	Episodes- 100000 Warmup - 100000, Batch Size- 10 Learn Rate - 0.005	Acc - 0.64 F1: 0.5, Presc:0.85 Recal': 0.35
6	Pokemon	Episodes- 10000 Warmup - 10000, Batch Size- 50 Learn Rate - 0.005	Acc - 0.55 F1: 0.54, Presc:0.37 Recal': 1.0

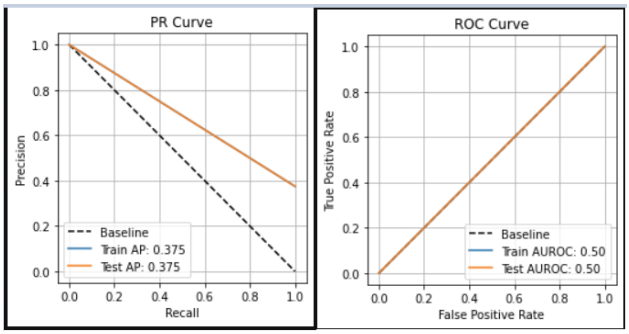


Fig 13 - AU-ROC curve for 1st scenario

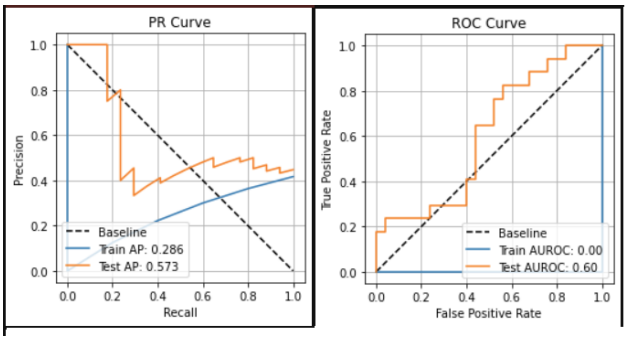


Fig 14 - AU-ROC curve for 4th scenario

After the experiments it is observed that the DQN model performed better with the pokemon data set compared to the kidney dataset. but it took significantly more time for the DQN model to run compared to other models . For scenario 2 and 5 the model took around 3:50 minutes to complete which is very high for classification problems. It also observed that the output from the model is highly irregular and the results varied significantly even with the same parameters. The model also took significantly more time to execute compared to other classic models.

With the experiments and analysis we can say that the hypothesis is assumed to be proved .

Experiment 4

3.4.1 Null Hypothesis 4

Relational models may produce better results than classic models when the data is highly relational but given enough data which is compatible with classic models ,Classic models will be getting the same results although after using much less time.

3.4.3 Materials and Methods 4

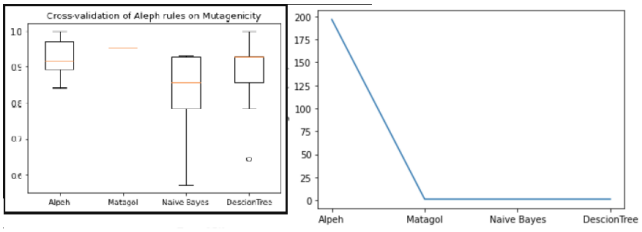
For this experiment we considered using decision tree and naive bayes to compare with aleph and metagol for a part of the Mutagenesis data set. We also captured time as it was a main part of our Hypothesis . The results are explained below.

3.4.4 Results and Discussion 4

After running the experiments it's found that metagol is surprisingly very fast compared to Aleph and also produced very good results even when compared to classic models like Decision tree and Naive Bayes

Accuracy for specific models are

Model	Accuracy
Aleph	92
Metagol	95
Naive Bayes	85
Decision Tree	81



With these results we can say that our hypothesis is only partially proven .Aleph and Metagol used large amounts of time but for almost same results given proper tuning classical algorithms would be better for most cases.

4. Discussion of the results, interpretation and critical assessment

After performing all the experiments we can see that for both the data sets we can observe that the Decision tree performed by far the best when compared to other algorithms. As it was a classification problem and the classification is a binary classification we assumed that the decision tree would be better initially when we started discussing this project. After running several experiments, we can say with certainty that the decision tree is very optimal for classification problems in general .

For Pokemon data set we did tried to explore the overfitting problem and our assumption is only true for SVM as it failed miserably when sampling is not done but surprisingly naive bayes performed well when sampling is not done , This is a very interesting observation , On further research it was found that as naive bayes is based on probability it might be able to better understand the relation between features and target when large data is present.

Reinforcement learning is something which we debated how to approach and we decided to go with classification using DQN. This is not what reinforcement learning is generally good at. In the research we did find reinforcement learning getting better results in classification problems but it was always when the data is very robust and also the number of data points were high. We assumed for our data set we will not be getting optimal results and our assumption is justified . DQN took 30X more time than other algorithms and produced not less than optimal results .

For relational models we decided to use a mutagenesis data set and run it with Aleph , Metagol ,Decision tree and naive bayes. Surprisingly aleph and Metagol produced very good results and classic algorithms did not perform as we expected.The relational models did took a lot of time but our assumption is that, as we are running swipl over python kernel and we did tried to run the same experiments natively on SWIPL and it was relatively fast to get to results than running through python.

5. Conclusions

As a conclusion we would say this project helps us to understand how and why different models perform differently and also how a simple task or hyper parameter tuning increases the performance of different models. We also observed interesting observations when we tested for over fitting.We understood the importance of relational learning and we can now see why they still have a place in the modern data science environment. when the data is less they often perform very well compared to classic models . We tried to convert the Train data set to make it compatible with classic algorithms and quickly realized the reason why no one tried doing that is that the amount of features a train has is very high and also its very difficult to represent them in a way a classic algorithm understands. We also found the limitations of reinforcement learning and why it's not the silver bullet everyone thought it to be as it was very good in some tasks but for classification classical algorithms are still better.

From the current setup we are able to understand so many things , but we also missed several interesting experiments like testing the effect of scaling. The one way we could think to improve this project is to have one more data set with a high number of data points as then we would be able to experiment much more on how DQN may perform better.

Overall this project helped us to understand how and why different models are important and also unique in their own way, And Reinforcement learning is really a great step forward but still the age olg relation based algorithms have their place even now.

Contributions

Kowshik Kesavarapu

kk01002 worked in research of different data sets and did the EDA , Data cleaning and feature engineering for the pokemon data set. He worked on Reinforcement learning , Logic based models which contributed to experiment 3 and 4, Worked on Initial implementation of decision tree. He helped in formulating the hypothesis for all the experiments and wrote the report for those both experiments.Also wrote the Section 4 and Section 5 of the report after discussing and gathering everyone's inputs.

Debjyothi Saha

ds01290 worked in research of different data sets and did the EDA , Data cleaning and feature engineering for the kidney data set. Worked on initial implementation of SVM , worked on experiment 1 Hyper parameter tuning and also worked on the report.

Gururaj Nimbalkar

gn00242 worked on Experiment 1 and Experiment 2 of the project and worked on the report . And also did the initial implementation of perceptron .

Aditya Mohan Vichare

av00550 worked on Experiment 2 and also worked on initial implementation of Naive Bayes. Hyper parameter tuning for Experiment 2 and combining different algorithms using K-fold.

Shweta Negi

sn00890 worked on initial implementation of SVM and also worked on report for section 1 and Section 2. Also helped in research of different data sets.

References

1. https://github.com/Denbergvanthijs/imbDRLAppendix
2. https://scikit-learn.org/stable/modules/tree.htm
3. https://www.brthor.com/blog/ai/reinforcement-learning/can-reinforcement-learning-be-used-for-classification/
4. https://analyticsindiamag.com/top-8-approaches-for-tuning-hyperparameters-of-machine-learning-models/
5. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python
6. https://dhirajkumarblog.medium.com/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a