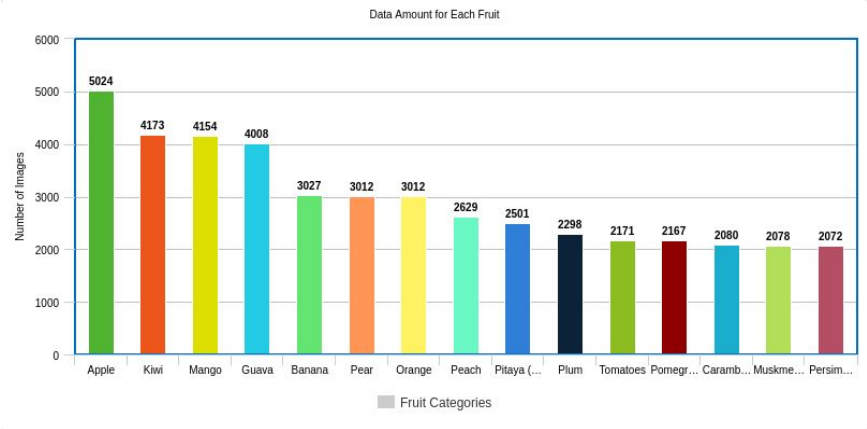
Statistical Learning for Engineers: Final Project

Adyota Gupta [Part 1: Machine Learning]
Adithya Rajagopalan [Part 2: Reinforcement Learning]

Task 1: Machine Learning

For the Machine Learning task, a large, labeled dataset of fruit images was obtained from Kaggle. There were folders for 15 fruits present, alongside the associated images of dimensions 320x258 pixels. Numerous images were taken in varying lighting conditions. Below is a distribution of the number of images in each of the folders. With respect to Apples, Kiwi, and Guava, subfolders existed for each variation of fruit (e.g. Red Apples, Green Apples, etc.), however for the purposes of this task, we did not care for the *variation* of the fruit, but rather that our model could identify the *correct* fruit. Below is a distribution of the labeled data set.

To perform classification of fruits, three methods were sought -- Polynomial Regression, Logistic Regression, and Artificial Neural Networks. For each model, a cross-validation was performed, varying a certain parameter to better choose the model.



NOTE: Each model was trained using a Quadro RTX 8000. Because the card has 48 GB of memory, the code was written to take full advantage of it.

1. Preprocessing

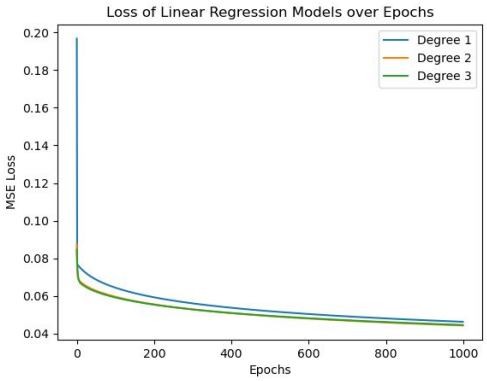
All the images were imported, and the corresponding label was created by using the folder name. Each image was subsequently resized from 320x258 pixels to 100x100 pixels to reduced the number of features of each image and cut noise. All three color channels were kept, thus yielding 100x100x3 = 30000 features from each image. All pixel values were already normalized between 0 and 1, so no extra steps were taken for scaling. In total, there were approximately 44000 images. For the label (y), two sets of labels were created -- one with categorical encoding, where a unique number corresponds to a fruit, and one-hot encoding, where an "indicator" array is created to denote the corresponding fruit. Finally, the data is shuffled, and then split into a training set (80%), a validation set (10%), and a test set (10%).

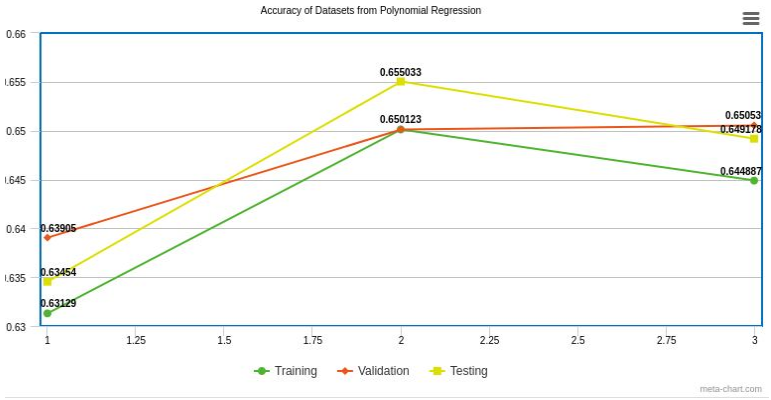
2. Looking at Different Models

Polynomial Regression

Polynomial regression follows the exact same methodology as that of Linear Regression to compute model parameters, $\hat{\beta} = (X^T X)^{-1} X^T Y$. Depending on the degree of the polynomial, $X = [X^* \ (X^*)^2 \ \dots \ ]$, where X^* represents the dataset. Because the dataset is very large, it unfortunately becomes infeasible to directly compute $\hat{\beta}$ and must be obtained through optimization. Models with degrees 1, 2, and 3 were tested, and the one-hot encoded output was used for y . The Mean-Squared Error function was used to obtain error, and backpropagation was used to adjust the weights and biases accordingly. 100 iterations of optimization was run. From the MSE Loss,

it is clear that convergence for the parameters was successfully obtained. The training, validation, and test set accuracies are plotted below. There is a clear increase in both the training, validation, and testing when using a degree of 2, peaking at approximately 65%. What is interesting to note is that there is a decrease in the training accuracy when the degree is 3, even though the loss is approximately the same as 2 at the end of 1000 epochs. This may be due to the fact that when evaluating the test set, the argmax function is used to return the corresponding category. If two elements are quite close in the length-15 output array, then it is possible that argmax will return the incorrect category. There is also the possibility that more iterations is needed to increase accuracy. The best model to pick of these three would be **Polynomial Regression with Degree=2**. Intuitively, this

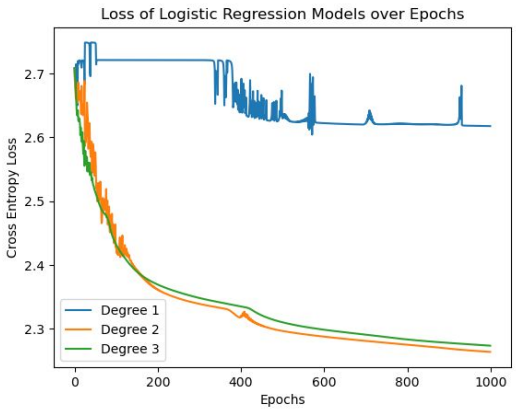




makes sense, as it would be able to capture some detail in this non-linear problem. Furthermore, with higher degree models, comes the cost of a larger parameter space, and thus the need for *even more data* due to curse of dimensionality.

Logistic Regression

Logistic regression follows a very similar methodology, but instead, we have a non-linear activation function. Logistic regressions are theoretically much better than linear regression in classification problems because the output vector from the Linear portion is rescaled so that they all sum to one, through use of the Softmax function. Then the class is determined using argmax, as this corresponds to the highest probability. Following a similar process as *Polynomial Regression*, the degree (and thus the number of features) were varied and cross entropy loss was used. Surprisingly, while convergence in the parameters was obtained (unreliably for degree 1), these models did not perform as well as those of the Linear Regression. This could be a result of a biased dataset -- because there significantly more samples of one type of class (Apples, Kiwi, and Mangoes in particular), and because a final scaling occurs such that all elements in the output array conform to the probability distribution, it is possible that

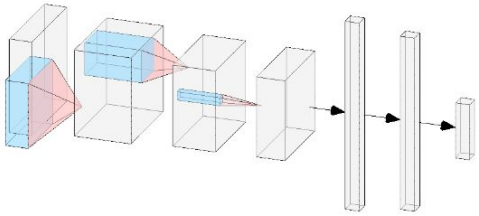


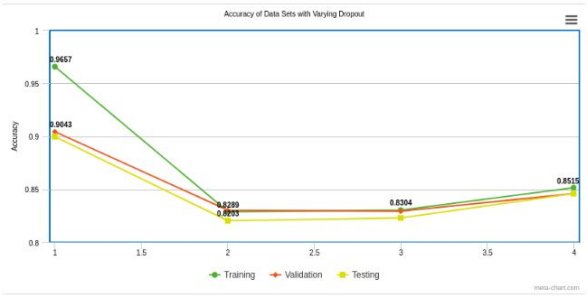
misclassification can occur as a result. Furthermore, it is interesting to see that the Degree 1 model performed so poorly compared to Degree 2 and 3 models. This may be due to the inherent non-linearity of the classification problem -- this can be seen in the loss function graphs. We obtain quick convergence with higher degree models, and barely any convergence in the degree 1 model, until much later in the training process.

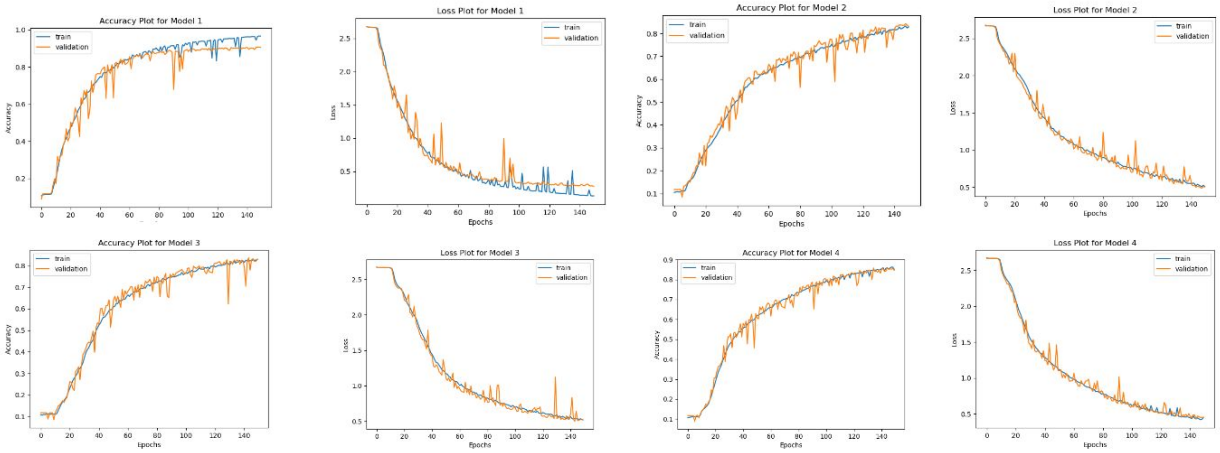
The best model to pick of these three would be **Logistic Regression with Degree=2**. Just as in the case of *Polynomial Regression*, we can capture non-linearity, but not have to suffer as much from curse of dimensionality.

Artificial Neural Network (ANN)

ANNs is the perfect choice for image classification as are extremely useful due to their versatility. Because we are performing image classification, convolutional layers, followed by fully connected units were used. To better perform a model evaluation, a Dropout layer was added in between the convolutional and fully connected layers. A dropout layer essentially "zeros-out" certain inputs at a specified rate, and in a way, loosely controls the number







of parameters being used in the ANN. Four models were created, each with dropout rates of 0, .05, .10, and 0.025, and accuracy in the testing, validation, and training sets were noted. As expected, the first model performed the best with a 96% accuracy on the training set, but has the largest deviation in performance from validation and testing sets due to overfitting. Increasing dropout did reduce overall accuracy, but better bridged the gap between testing, validation, and training sets. In a way, the fourth model performed quite well, with a training, validation, and testing accuracy of around 85%. All models converged well, based on the loss and accuracy plots. As tempting as it may be to pick Model 1, **Model 4 with a Dropout of 0.025** would be my model of choice due to its robustness. Perhaps I'd have to let it train for more epochs to further increase accuracy.

And of the three contending models, it is quite easy to determine that the **ANN with Dropout of 0.025** would be the ultimate model of choice.

Task 2 : Reinforcement Learning

1. Dynamic Programming:

This was performed in two ways - policy iteration and value iteration - listed below. Both algorithms made use of the same model for simulating the cart-pole system - detailed in the project description given to us. The parameters used in this model were :

M = 1 Kg; m = 0.1 Kg; g = -9.8 m/s^2; L = 0.5 m; mu_c = 0.0005; mu_p = 0.000002; del_t = 0.02s

For all algorithms used in part two the state space was divided based on the location, angle, speed and angular velocity as described in the project description. This divided the space in 162 states. Two actions were possible. For the dynamic programming algorithms the two actions had values 10N and -10N.

A discounting term (gamma) of 0.9 was considered in all cases [taken from example 3.14 in chapter 3 of Sutton and Barto's RL text book.

a. Policy Iteration:

To be able to perform the two steps - policy evaluation and policy improvement - required by this algorithm. One first needed to calculate the probabilities of all possible outcome states given an input state - *P(s'|s,a)*. This is 162 element vector for ever s and a combination. Resulting in a 162 X 324 matrix. This was calculated by randomly generating 10000 points in each state action pair (s,a) and passing these points through the cart pole model described in the project description with parameters as shown above to generate s'. From these results the probability distribution *P(s'|s,a)* could be estimated.

Once this was done, policy evaluation was performed and the best value function was converged upon. Convergence was said to have occurred if the delta (calculated as difference in value function over iterations) was less than a certain value theta. Theta was chosen to be 0.01.

This was followed by policy improvement to identify the best policy. If the newly produced policy differed from the previous policy, policy evaluation was repeated and this continued till a stable policy was identified.

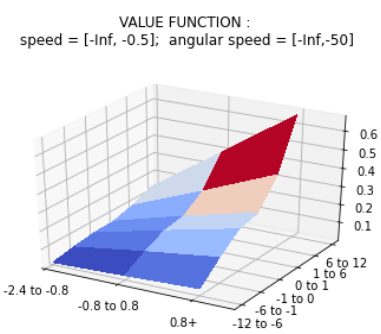
b. Value Iteration:

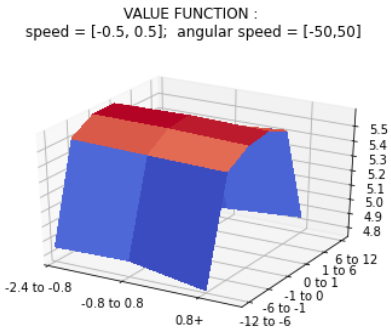
Similar to policy iteration, *P(s'|s,a)* was estimated. Once estimated, the best value function was converged upon. Convergence criterion here was the same as in the policy iteration case (theta = 0.01).

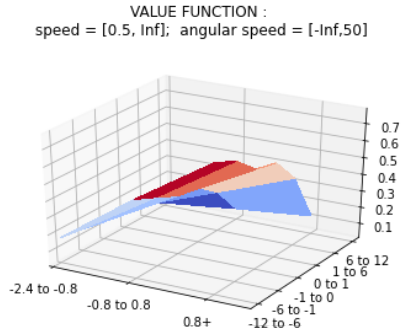
The value functions and policies arrived at by each of the algorithms can be seen below. They are plotted as a function of location and angle for given speed and angular velocity.

One can see that the value functions resulting from both algorithms look rather similar in shape but not in absolute value, suggesting that policy iteration produces a more robust value function

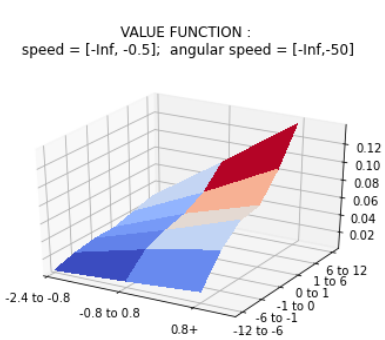
Value functions from - Policy Iteration

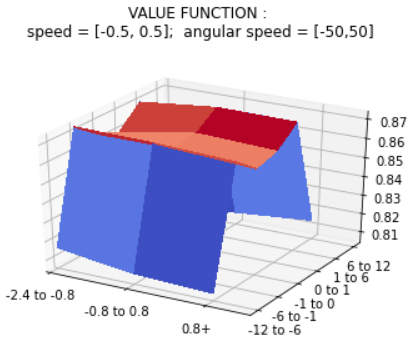


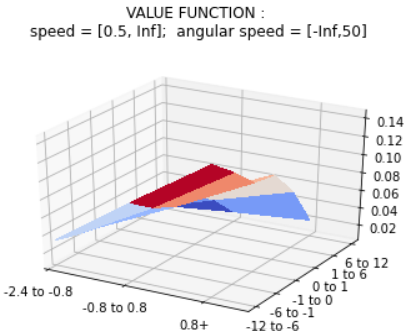




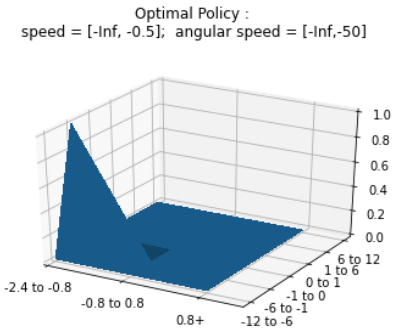
Value functions from - Value Iteration

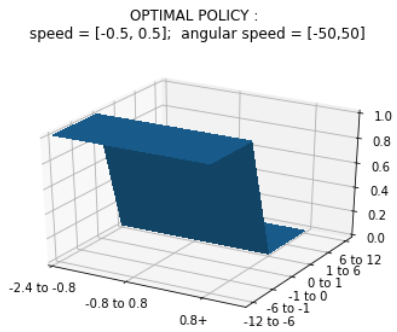


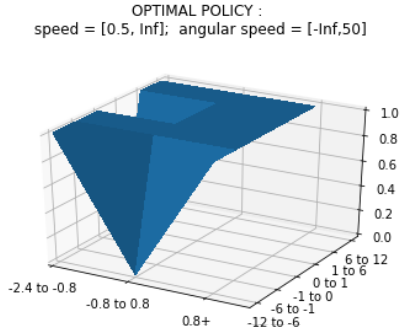




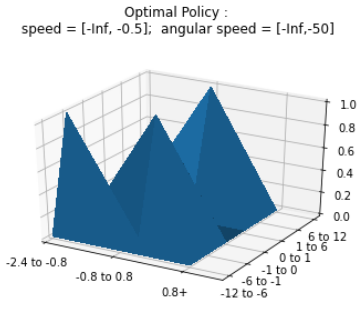
Policies from - Policy Iteration

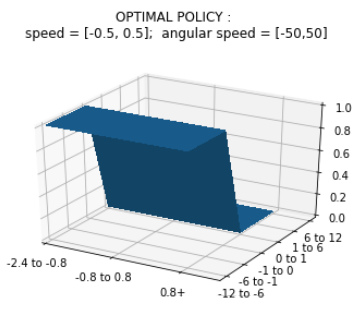


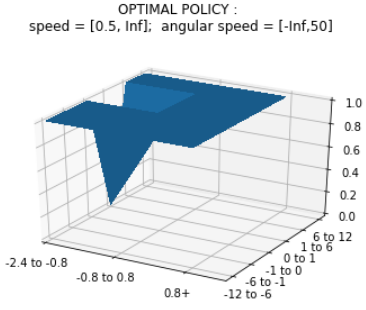




Policies from - Value Iteration







The policies arrived at by these two algorithms look similar (or basically the same) when speed and angular velocity are close to zero. But as these terms move further from zero there differences that can be observed. This fact,

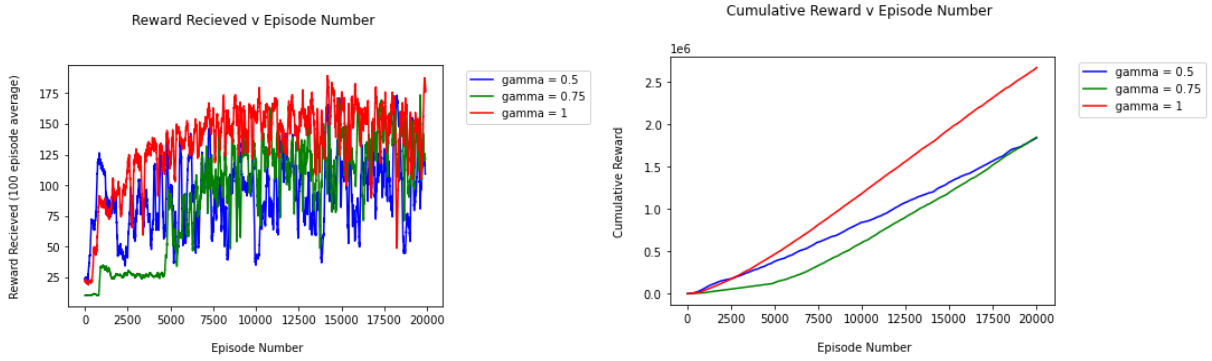
combined with the knowledge that the value function only increases with the number of iterations in policy iteration suggests that policy arrived at by policy iteration may be a more optimal one.

2. Temporal Difference with Q-learning:

The model used for the Q-learning section of the project was one suggested in the project description - openaigym's cartpole environment. The state space was divided in the same was as was done for the dynamic programming case. The default rule for termination of an episode of gym was used as it agreed with the requirements in the project description. The parameters required to be set for Q -learning were:

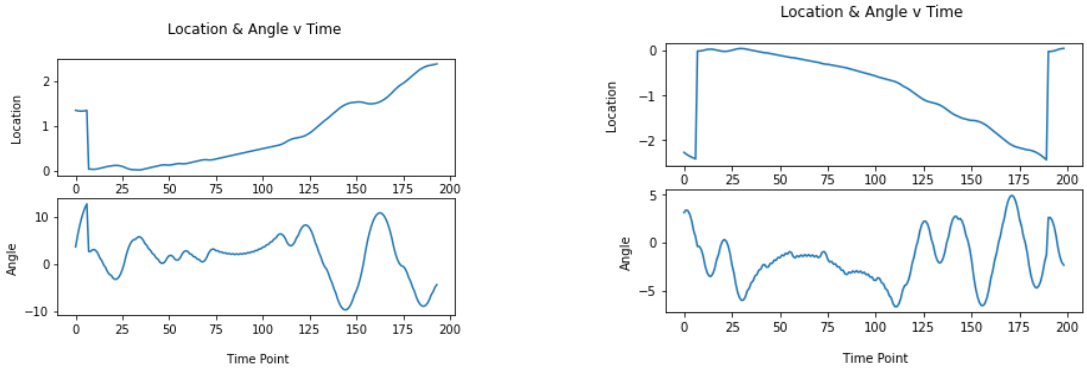
The policy - An epsilon greedy policy was used with epsilon = 0.05
Learning rate - a learning rate (alpha) of 0.05 was used
Decay rate - (gamma) as specified by the project description 3 different values - 0.5, 0.75 & 1 were used
Reward rule - every step was given a reward of 1 except the step that led to termination which was 0.

The rewards received can be seen (left) to increase as a function of episode number. While the project description required a 10 window moving average for the calculation of reward received, a 100 window average is used here for smoothness sake. The cumulative reward accumulated over episodes (right) shows an initial convexity and then becomes linear with episode number suggesting that the average reward received in an episode becomes somewhat constant. This agrees with the left plot.



.

Next, the speed and angle over the course of two example episodes are plotted below:



All code is available at: https://github.com/adyotag/SLE_Project