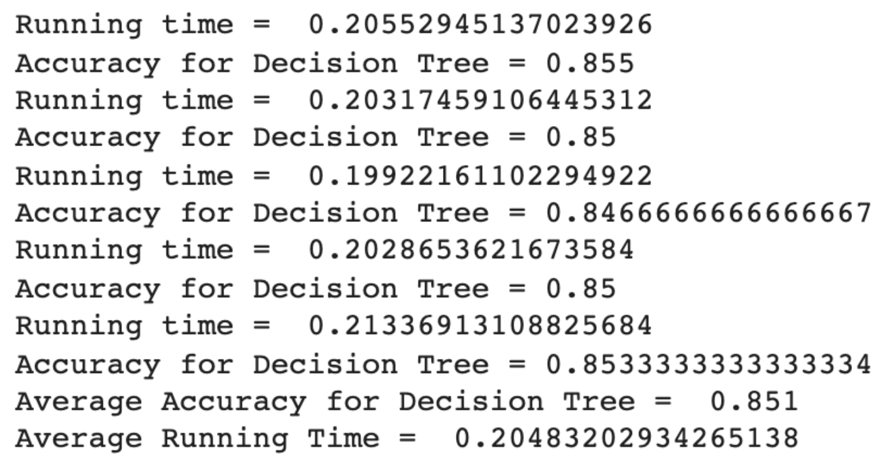
Name: Anar Rzayev **ID:20190788**

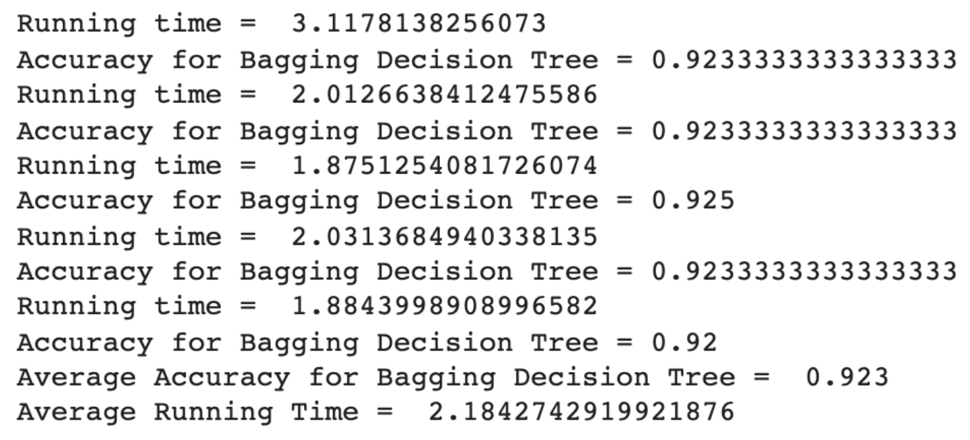
MAS473 - Introduction to Artificial Intelligence with Mathematics

Problem 1:



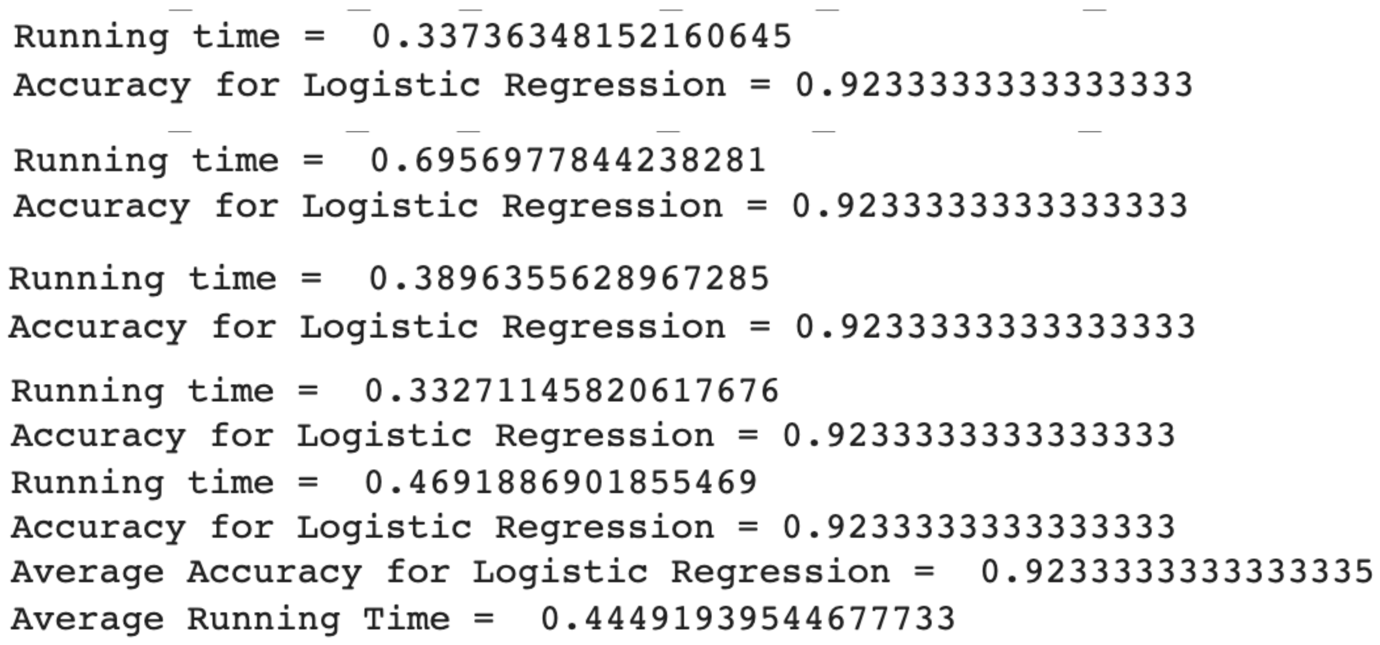
```
Running time =  0.20552945137023926
Accuracy for Decision Tree = 0.855
Running time =  0.20317459106445312
Accuracy for Decision Tree = 0.85
Running time =  0.19922161102294922
Accuracy for Decision Tree = 0.8466666666666667
Running time =  0.2028653621673584
Accuracy for Decision Tree = 0.85
Running time =  0.21336913108825684
Accuracy for Decision Tree = 0.8533333333333334
Average Accuracy for Decision Tree =  0.851
Average Running Time =  0.20483202934265138
```

Figure 1: Running Decision Tree with hyperparameter max_depth as 4



```
Running time =  3.1178138256073
Accuracy for Bagging Decision Tree = 0.9233333333333333
Running time =  2.0126638412475586
Accuracy for Bagging Decision Tree = 0.9233333333333333
Running time =  1.8751254081726074
Accuracy for Bagging Decision Tree = 0.925
Running time =  2.0313684940338135
Accuracy for Bagging Decision Tree = 0.9233333333333333
Running time =  1.8843998908996582
Accuracy for Bagging Decision Tree = 0.92
Average Accuracy for Bagging Decision Tree =  0.923
Average Running Time =  2.1842742919921876
```

Figure 2: Running Bagging Decision Tree with hyperparameter of Decision Tree in which splitter is 'random', maximum number of leaf nodes is 16, and hyperparameters of Bagging where estimator of n will be 500, maximum number of samples is 500, maximum number of features is 20, bootstrap is assigned to True, and n_jobs will be assigned to -1



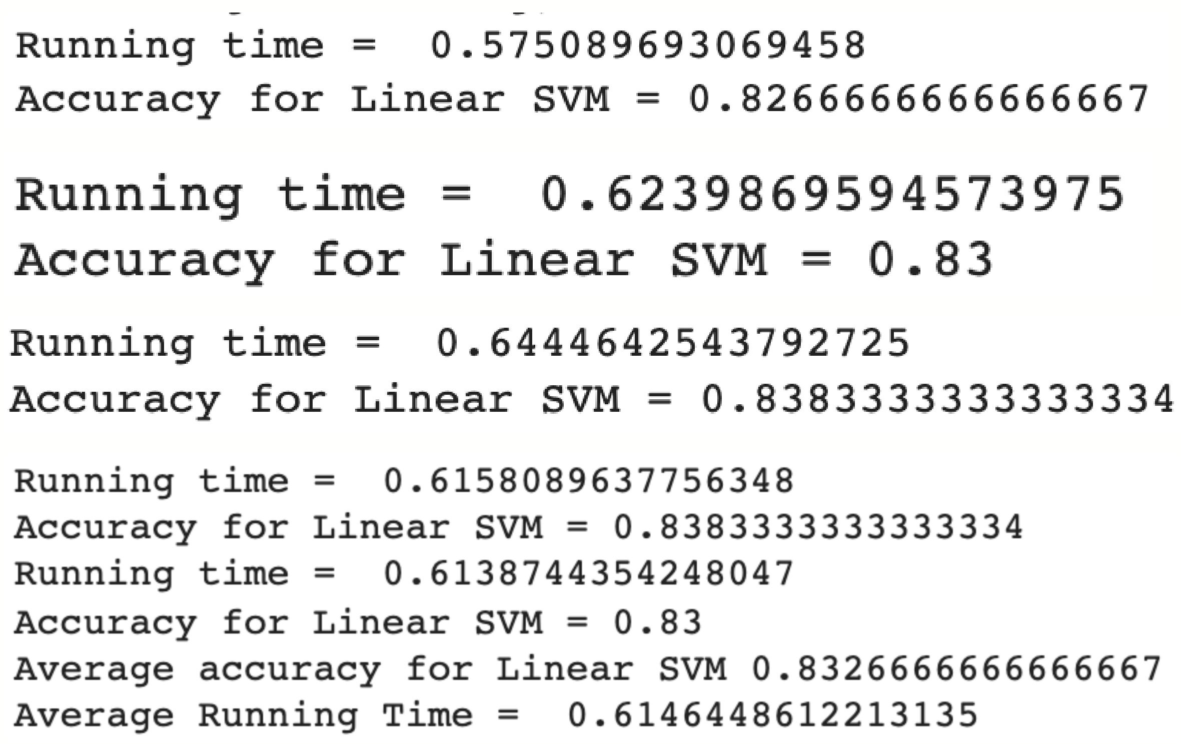
```
Running time =  0.33736348152160645
Accuracy for Logistic Regression = 0.9233333333333333

Running time =  0.6956977844238281
Accuracy for Logistic Regression = 0.9233333333333333

Running time =  0.3896355628967285
Accuracy for Logistic Regression = 0.9233333333333333

Running time =  0.33271145820617676
Accuracy for Logistic Regression = 0.9233333333333333
Running time =  0.4691886901855469
Accuracy for Logistic Regression = 0.9233333333333333
Average Accuracy for Logistic Regression =  0.9233333333333335
Average Running Time =  0.44491939544677733
```

Figure 3: Implementing Logistic Regression with hyperparameter multi_class as multinomial, solver as lbfgs, and C as 10



```
Running time =  0.575089693069458
Accuracy for Linear SVM = 0.8266666666666667

Running time =  0.6239869594573975
Accuracy for Linear SVM = 0.83

Running time =  0.6444642543792725
Accuracy for Linear SVM = 0.8383333333333334

Running time =  0.6158089637756348
Accuracy for Linear SVM = 0.8383333333333334
Running time =  0.6138744354248047
Accuracy for Linear SVM = 0.83
Average accuracy for Linear SVM 0.8326666666666667
Average Running Time =  0.6146448612213135
```

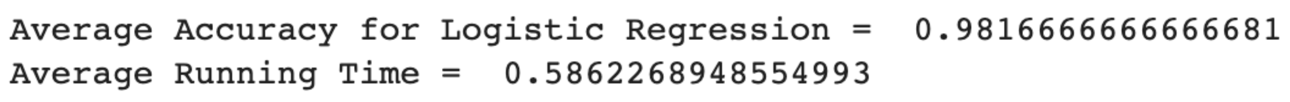
Figure 4: Applying Linear Support Vector Machine with hyperparameters C = 1, loss as hinge, and max_iter as 1000

As it can be seen from the submitted code implementations, we trained *Decision Tree*, *Bagging Decision Tree*, *Logistic Regression*, and *Linear Support Vector Machine* using the provided training set. We measured the training time for each of these models and evaluated the corresponding accuracy using the given testing set. From the above figures, one may imply that the fastest algorithms above those 4 approaches are Decision Tree and Logistic Regression, in which their average running time during the 5 repetitions of randomness are 0.2048 seconds, and 0.4449 seconds, respectively. However, Bagging Decision Tree & Linear Support Vector Machine were the slowest among them, with corresponding average running time of 2.1843 seconds, and 0.614645 seconds, respectively.

Subsequently, we also calculated the (average) accuracy of each of the trained models using the provided testing set. According to figures with accuracy values, Logistic Regression with hyperparameters multinomial multi-class, "lbfgs" solver and C = 10 achieved the highest accuracy measurement: 92.33 %. Similarly, taking into account the given hyperparameter values and running the training 5 times using different random values of random_state provided in the code block, the average accuracy reached 92.3 %. In the subsequent manner, one can conclude from figures, the following model algorithms in terms of average accuracy were Decision Tree & Linear Support Vector Machine, with respective measurements of 85.1 % & 83.267 %.

In conclusion, we may encapsulate that the slowest algorithms such as Bagging Decision Tree produced higher average accuracies approximately equal to 92.3 %, whereas fastest models, particularly Decision Tree, achieved lower (average) accuracy measurements almost equal to 85.1 %.

Problem 2:



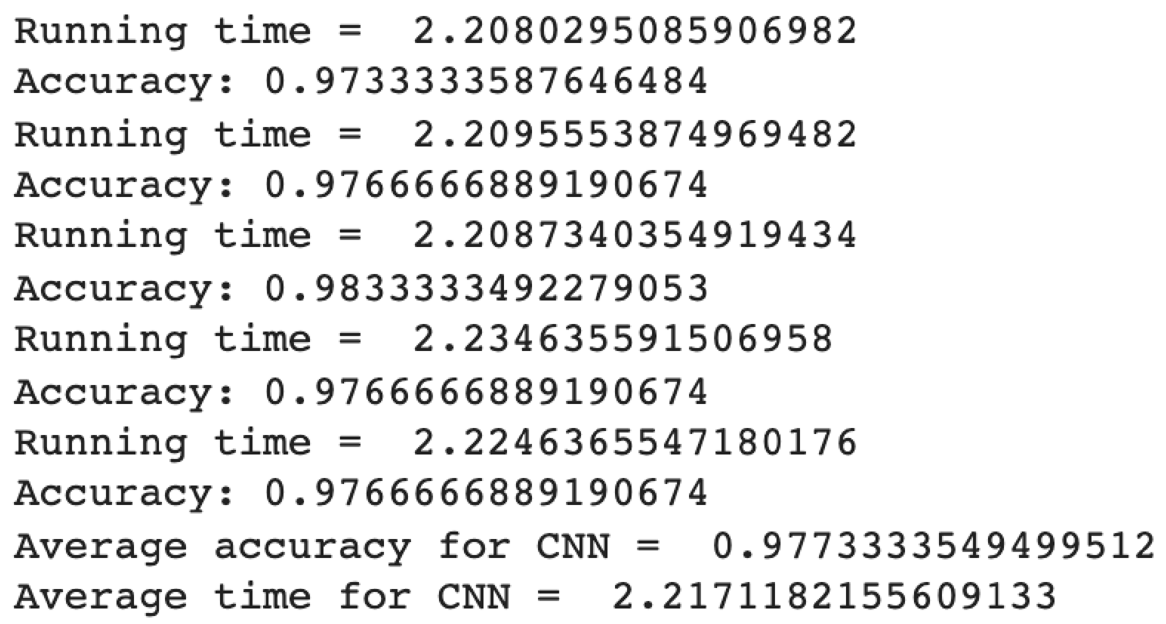
```
Average Accuracy for Logistic Regression =  0.9816666666666681
Average Running Time =  0.5862268948554993
```

Figure 5: Application of preprocessing method "QuantileTransformer" with hypertuned Logistic Regression model, given the trained model of chosen hyperparameter values n_quantiles = 100

As seen from the submitted code implementations, we trained the *Logistic Regression* model using the provided training set, taking into account the preprocessing tool and efficient hyperparameter values in this question. Particularly, our choice of preprocessing mechanism for better data input representations was *QuantileTransformer.* As we discussed in the code blocks, this provides a unique non-parametric transformation for mapping the data into a uniform distribution where the values range between 0 and 1. More detailed information could be found in the following link: https://scikit-learn.org/stable/modules/preprocessing.html

After performing the preprocessing approach, it will be followed by the hypertuning technique in which choosing the right values for parameter n_quantiles in QuantileTransformer function and C value in LogisticRegression will be very critical to obtain higher accuracy (particularly, higher than 98 %). Moreover, our goal is also to construct a specific Logistic Regression method such that training time of this model needs to be less than or equal to 1 second. Therefore, experimenting with different combinations for n_quantiles & C, we came to the final conclusion of choosing *n_quantiles as 100*, and variable *C as 10* in this tuning procedure. Indeed, having in-depth look to the training process of code block and resultant figure above, we may conclude that n_quantiles = 100 for QuantileTransformer and C = 10 for LogisticRegression could be an ideal decided hyperparameter measurement for chosen Logistic Regression Model from problem 1.

Problem 3:



```
Running time =  2.2080295085906982
Accuracy: 0.9733333587646484
Running time =  2.2095553874969482
Accuracy: 0.9766666889190674
Running time =  2.2087340354919434
Accuracy: 0.9833333492279053
Running time =  2.234635591506958
Accuracy: 0.9766666889190674
Running time =  2.2246365547180176
Accuracy: 0.9766666889190674
Average accuracy for CNN =  0.9773333549499512
Average time for CNN =  2.2171182155609133
```

Figure 6: Running CNN architecture with defined training/evaluation models and reporting average time & accuracy for CNN

As it can be seen from the submitted code implementations, we trained *Convolutional Neural Network* using the provided training set. We measured the training time for a given architecture model in the code block and evaluated the corresponding accuracy of the trained model using the given testing set.
In the first phase, one should provide necessary macro values for batch_size, learning_rate, and epochs to run the DataLoader & Adam optimizer. Performing for-loop “epochs” number of times, we extract X & Y parts for training & testing data, respectively, in which forward propagation is executed using the CrossEntropyLoss as a loss function. In the end, we evaluate our predictions and compute the accuracy using the comparisons made between real values of test_y and predictions coming from model() operation. This will correspondingly reveal the average running time and accuracy for CNN approach, as indicated in Figure 6. It is inevitable such approach enables much efficient (average) accuracy of 97.73 % wherein average time required to perform randomly sampled epoch number of CNN algorithm is approximately 2.217 seconds.