Report

CE802 Assignment

Registration Number: 1804162

Word Count: 1514

Introduction

This report focuses on the comparative studies carried out for the two predictive tasks. The first predictive task was a binary classification task; where a system was to be developed to determine whether a new hotel will make profit given a set of historical data on existing hotels. The second task being a regression problem of the same context using a different set of data; where we were trying to calculate the exact amount of profit the hotel is expected to make.

Classification Task

Development Process

After importing the training data, I first had to check to make sure the data types were numerical as any categorical features would have to be converted to numeric features before proceeding. With this data however, there were no categorical features, so I moved on to the pre-processing stage. Here, I checked for any missing values within the dataset; where it was discovered that the feature 'F21' contained five hundred missing values. As half of the data in this feature was missing, there was only a couple of approaches that could have been made: removing the column completely or imputing the value within the column. As we did not know the context of this feature, we could risk losing valuable information if we were to remove this. For procedures such as the k-NN, this could also introduce false classifications if this column lessens similarity. Instead, I decided to take the mean of the feature values and replace any missing entries with this; as the values within the feature did not vary much (11.34 to 18.15, with a deviation of 1.06). I then performed the same steps within the testing data, as it was also discovered that the same number of missing values were located within the same feature.

The training data was then split, so that 70% of the data would be used to train the model, and 30% would be used as a validation test. The purpose of this was to ensure that the score we compare is made on data previously unseen to the model, and so we can check the model performance to see if it is overfitting or underfitting the data. These comparisons were made by predicting the classes of the train and validation sets, before comparing them to their expected class. Confusion matrixes were also made, so the number of false positives/negatives would be comparable.

Pipelines was used alongside 'GridSearchCV' to optimise the specified hyperparameters and scaler used. This creates a pre-defined number of folds to fit the values to, before the folds are cross validated to provide the best set of parameters. After the scores of each model was calculated, the model was then saved using the 'pickle' library so they could be reused. The models selected for experimentation were:

- Decision Tree Classifier
- SVM Classifier
- k-NN Classifier
- Random Forest Classifier

Experimentation

For all models I used, I included a standard scaler in the pipeline with 'True' or 'False' values. This is to allow models that do not rely on or require such scalers to reject them.

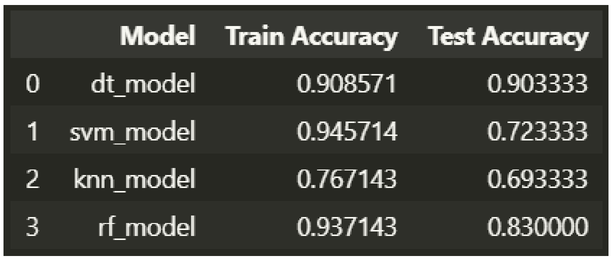
For my decision tree, I included three hyperparameters in my parameter grid: the criterion ('Gini' or 'Entropy'), the maximum depth of the tree, and the minimum required samples for a split to occur. After running this model numerous times, I was regularly having the optimal minimum samples to be 10, so this variable was then fixed in the parameter grid. For the tree depth, I first started with a larger maximum depth, however the optimal parameter value was never exceeding 10. For this reason, I then reduced this to improve training time. The best criterion that was used was 'Gini'.

For my SVM, I included two hyperparameters in my grid: the C and gamma coefficients. For each of these, I supplied a list of five values of different powers of 10: all of which being positive values. With my best accuracy, the best parameters were 100 for C, and 0.01 for gamma.

For my k-NN classifier, I also included two hyperparameters: the number of neighbours, and the P coefficient. The number of neighbours that produce the best accuracy was 25, and the best value of the P coefficient that was returned was 1. This value corresponds with the model using Manhattan Distance.

I also decided to experiment with random forest classifiers, as I wanted to see how they would perform in comparison to my decision tree classifier. I experimented with different numbers of estimators; with 130 being the best output I received.

From the results of my experimentation, I decided to use my decision tree model as my classifier for this prediction task. This is because when compared to my other models, it regularly achieved higher accuracies, and did not vary much between the training and validation testing sets. The accuracies of all models can be seen below.



	Model	Train Accuracy	Test Accuracy
0	dt_model	0.908571	0.903333
1	svm_model	0.945714	0.723333
2	knn_model	0.767143	0.693333
3	rf_model	0.937143	0.830000

Figure 1: Table of Model Comparisons

I then wanted to visualise this data, so that I could see a full comparison of my accuracies. Its clear here that there was some slight to moderate overfitting in some of the models, with all testing accuracies being lower than the decision tree model's.

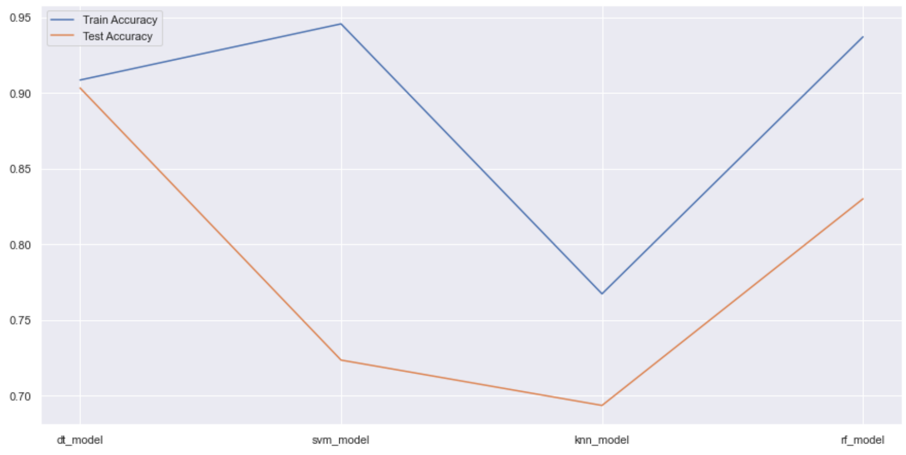


Figure 2: Visualizing Train and Test Accuracies

Regression Task

Development Process

As with the classification task, I first had to check to make sure that the data types of the imported training data were numerical. After investigating, I found that two of my features were not numerical: features 'F3' and 'F11'.

Feature 'F11' was the simplest to convert, as it contained an ordinal set of values ('Very Low' to 'Very High'). This allowed me to replace each of these values by using arbitrary values (1 to 5) that act as a suitable scale to differentiate the values. Feature 'F3' however contained non-ordinal values; and instead contained values regarding countries. As we could not provide a set of numbers as before, one-hot encoding was instead used on this feature to create a new feature for each of the possible values, where a 1 or 0 would be present depending on the location.

The data was then again split into training and validation testing sets, before I decided to reduce the number of features that were to be included. As we had more features as we had previously, I wanted to first investigate the importance of each feature based on it's information score. From this, I found that only around twenty-two of the features offered a score noticeably higher than 0. Using the function 'SelectKBest()', I then transformed the training and validation testing sets to only include these features. As with before, I also repeated all the above steps for the testing data.

I then repeated the stages stated in the previous task, however instead of using a confusion matrix, I also calculated the root mean squared error of the training and validation sets for each model. The models selected for experimentation were:

- Linear Regression Regressor
- SVM Regressor
- Lasso Regressor
- SGD Regressor

Experimentation

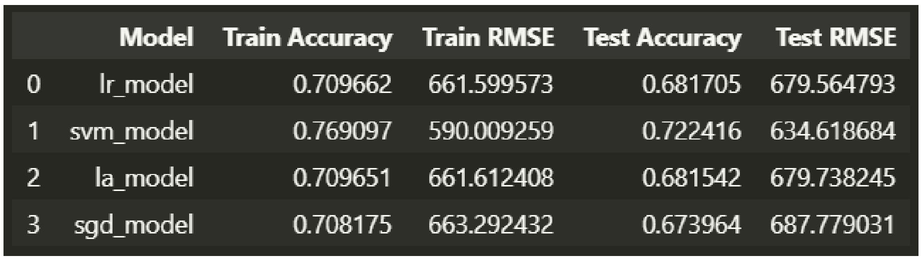
As there are not many parameters that are able to be tuned for linear regressors, I attempted to experiment with what was on the documentation. However, none of these made much of an effect, but were left in anyway.

For my SVM, I used the same parameters as I did for my classifier, and the best accuracy had parameters of 1000 for C, and 0.01 for gamma.

For my lasso regressor, I only used the alpha coefficient, and selected values between 0.1 and 0.9. The best value of alpha with the model's best accuracy was a value of 0.9.

Lastly, I decided to experiment with the SGD regressor, as this allowed me to use a procedure like linear regression whilst allowing for hyperparameter tuning. The parameters I decided to include were the alpha coefficient, the loss function, the penalty function, and the learning rate method. From my experiments, the best parameters were a value of 0.00001 for the alpha, 'invscaling' for the learning rate, 'squared_loss' for the loss function, and 'l2' for the penalty.

From the results of my experimentation, I decided to use my SVM model as my regressor for this prediction task. This model produced the best accuracy in both its training and validation testing sets, as well as also producing the lowest RMSE of all the models (which is the metric to be judged against). The accuracies and RMSEs of all models can be seen below.



	Model	Train Accuracy	Train RMSE	Test Accuracy	Test RMSE
0	lr_model	0.709662	661.599573	0.681705	679.564793
1	svm_model	0.769097	590.009259	0.722416	634.618684
2	la_model	0.709651	661.612408	0.681542	679.738245
3	sgd_model	0.708175	663.292432	0.673964	687.779031

Figure 3: Table of Model Comparisons

I again wanted to visualise these values, so plotted the accuracies and RMSEs on different plots. It is clear to see that there is some very slight overfitting in each of the models, however the difference is consistent between models indicating that is related to the data itself.

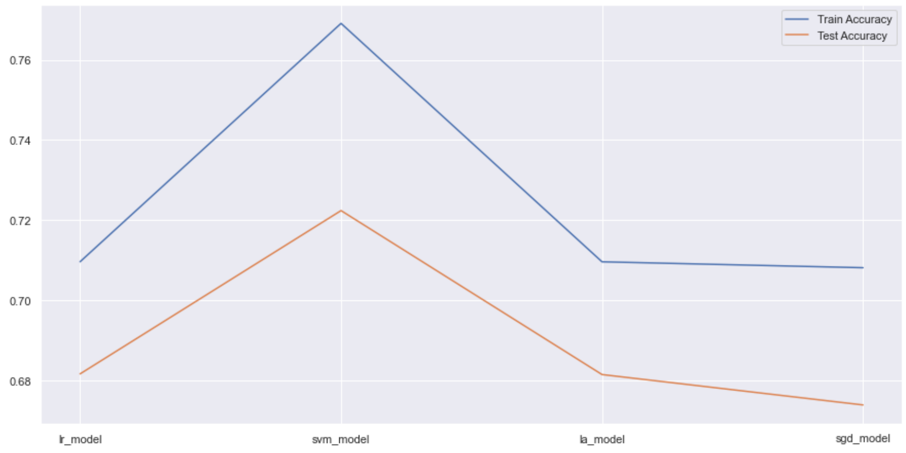


Figure 4: Visualizing Train and Test Accuracies

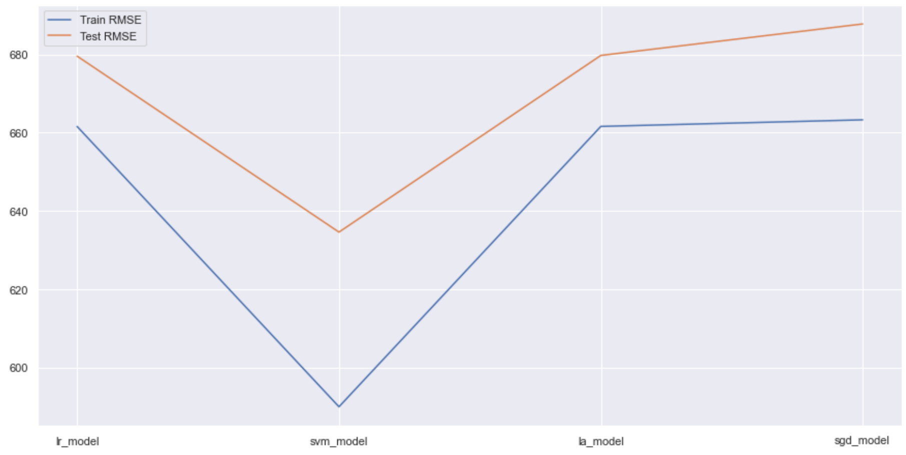


Figure 5: Visualizing Train and Test RMSEs

Conclusions:

From the results from my experimentation, it can be said that these problems are suitable for a machine learning tasks. Reasonable models were produced for each task, but better parameters could have been chosen to test (such as SVM kernels). However due to hardware limitations this was not practical to do. However, with better hardware and experimenting with other procedures, it is clear to see that there is room for improvement.