COMP3009 Machine Learning

Assignment 3: Report

December 9, 2021

Group 18

Name	Student ID	Username
Teo Shi Bin	20183717	hcyst2
How Khai Chuin	20210654	hcykh1
Loh Qian Kai	20194664	hcyql1
Mohamad Arif Bin Mohamed Abu Baker	20116042	hfymm5
Muhammad Fikri Bin Mohd Roslee	20116065	hfymm4

Contents

Introduction

In this coursework, we are using artificial neural network (ANN) to do both classification and regression with data sets we used from previous coursework.

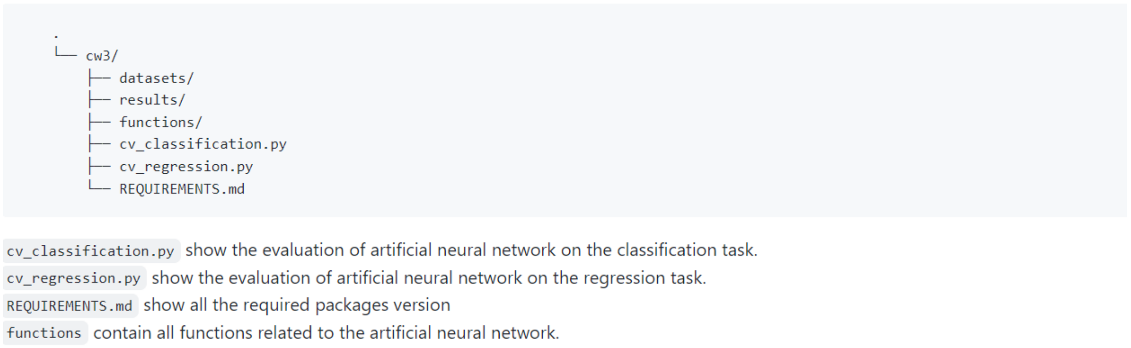
Data Set for Classification & Regression

Classification data set consists of 12 features and 1 column of class labels with a total of 299 instances within the data set. The goal is to classify the death event based on the 12 features.

Regression data set consists of 8 features and a total of 1030 instances were consisted. The goal is to predict the Concrete Compressive Strength based on the 8 features.

Implementations

Folder Structure



```
.
└── cw3/
    ├── datasets/
    ├── results/
    ├── functions/
    ├── cv_classification.py
    ├── cv_regression.py
    └── REQUIREMENTS.md
```

`cv_classification.py` show the evaluation of artificial neural network on the classification task.
`cv_regression.py` show the evaluation of artificial neural network on the regression task.
`REQUIREMENTS.md` show all the required packages version
`functions` contain all functions related to the artificial neural network.

Pseudo-code

Complete implementation of ANN with CV, extra details in it's corresponding section.

Algorithm 1 ANN Cross Validation Experiments (Classification & Regression)

```
1:  Set Experiments Parameters
2:  Load Dataset
3:  for $hyperparameterConfigurations = \{p1_1, p2_1, \ldots\}, \{p1_1, p2_1, \ldots\}, \ldots$ do    ▷ Grid Search
4:      for kfolds $= 1, 2 \ldots, k$ do    ▷ Cross Validation
5:          Reset seed state
6:          Define network with the same initial weights
7:          Define loss function & optimizer
8:          Init tensorflow session graphs and run session
9:          Cross validation dataset train & validation split
10:         for epochs $= 1, 2, \ldots, maxEpochs$ do    ▷ Training
11:             Train model by feeding in entire dataset and optimize loss with optimizer
12:             if better performance then    ▷ Early Stopping
13:                 Print and store performance results or make model checkpoint
14:             end if
15:         end for
16:         Reset tensorflow session graphs
17:     end for
18: end for
19: Plot stored Results    ▷ Evaluation
```

ANN for Classification

Data pre-processing

The first thing that we do before feeding the data into neural network is to do Min-Max 5 and 95 Percentile Normalization into range of [0,1] bounding possible outliers, the exact same method was applied to previous work of SVM to make the feature space sparse and easier for the model to fit numerically. Hyperparameter tuning is done along with 10 folds cross validation to minimize the generalization error of the model obtained in a single hyperparameter setting.

Hyperparameter Tuning

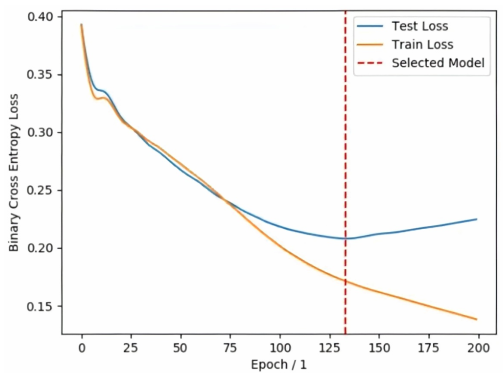
In classification, the only thing that we tune is learning rate. Learning rate is a indispensable hyperparameter to tune in ANN, given the gradient descent provided the direction of parameter update, learning rates define a the magnitude of change in a single parameter update. This is an important hyperparameter as a big learning rate may overshoot and do not converge to a good value, while a small learning rate may take too long to converge.

Model Architecture and Selection

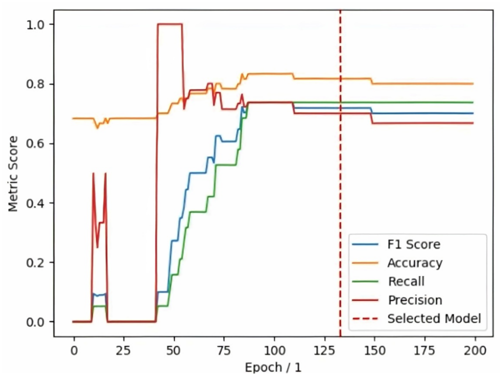
Model architecture can be treat as one of the hyperparameters to tune in order to find out a relatively good model to do the task. Due to the limit of time and computation resources, we only tried several possibles on small ANN and arbitrary picked our model as it performs relatively more consistent in our few trials. We also decided not to do deeper neural network as the number of instances in our classification dataset is only 299.

Our chosen model have 5 layers with numbers of node corresponded to [12, 24, 32, 24, 2] where the first and last element being the number of input attributes and number of output labels. All hidden layers and the input layer are using sigmoid activation function. Finally, the output layer is pass through a softmax layer for prediction.

The selection of the model after training is done by using early stopping which can be depicted as Figure 1a. The model before the divergence of test loss and train loss is chosen by the early stopping mechanism as the divergence is a sign of overfitting. In Figure 1a, we picked the model with lowest test loss, but it is possible to pick the model with highest performance metric, as we did in our experiment. Both ways realises the idea of select the final model before it overfitting.



(a) Cross Entropy Loss



(b) Corresponding Performance Metrics

Figure 1: Early Stopping

Loss Function and Optimizer

The loss that we feed into this optimizer is a weighted version of binary cross entropy, the cross entropy loss of each class is weighted using this equation, where the weight of class $\boldsymbol{j}$ is equal to the number of total instances $\boldsymbol{n}$ divided by total number of classes $\boldsymbol{k}$ multiply by positive labels in this class $\boldsymbol{n_j}$. This improves the model as it is trying to penalise the model more when false negative prediction occurs which is useful to solve the imbalance classes.

$$Loss = -\sum_{i=1}^{2} w_i \cdot y_i \cdot \log(h(x))$$

$$w_i = \frac{n}{kn_i}$$

The main loss optimization method that we used is Adam optimizer which comes with its own new hyperparameters, however we decided to use the default for the simplicity of the implementation. We selected this optimizer is because the speed of convergence is faster than standard gradient descent optimizer. It is also known for good consistency compared to other optimizers.

Evaluation

F1 score is our performance metric. Box plots below (Figure 2) depict the range of the F1 results acquired from performing 10-fold cross validations while using different learning rates. This is simply a grid search method of identifying better parameters. The best models selected in each fold is trained within max epoch of 200.

The learning rate of 0.02 returns the best model in our experiment with the lowest range between minimum and maximum of ± 0.0864 and a decent high accuracy of 0.7967 and median of 0.7810. Despite of the model with learning rate of 0.05 have a higher average F1 score, we did not select it because it not consistent among the folds as the model we choose.

Folds	1	2	3	4	5	6	7	8	9	10
F1 Score	0.8000	0.7368	0.7368	0.9000	0.7273	0.7619	0.7500	0.8889	0.8235	0.8421

Table 1: Folds Performance of $lr = 0.02$ Model

	F1 Score
Average	0.7967
Range	0.8136 ± 0.0864
Median	0.7810

Table 2: Performance of $lr = 0.02$ Model

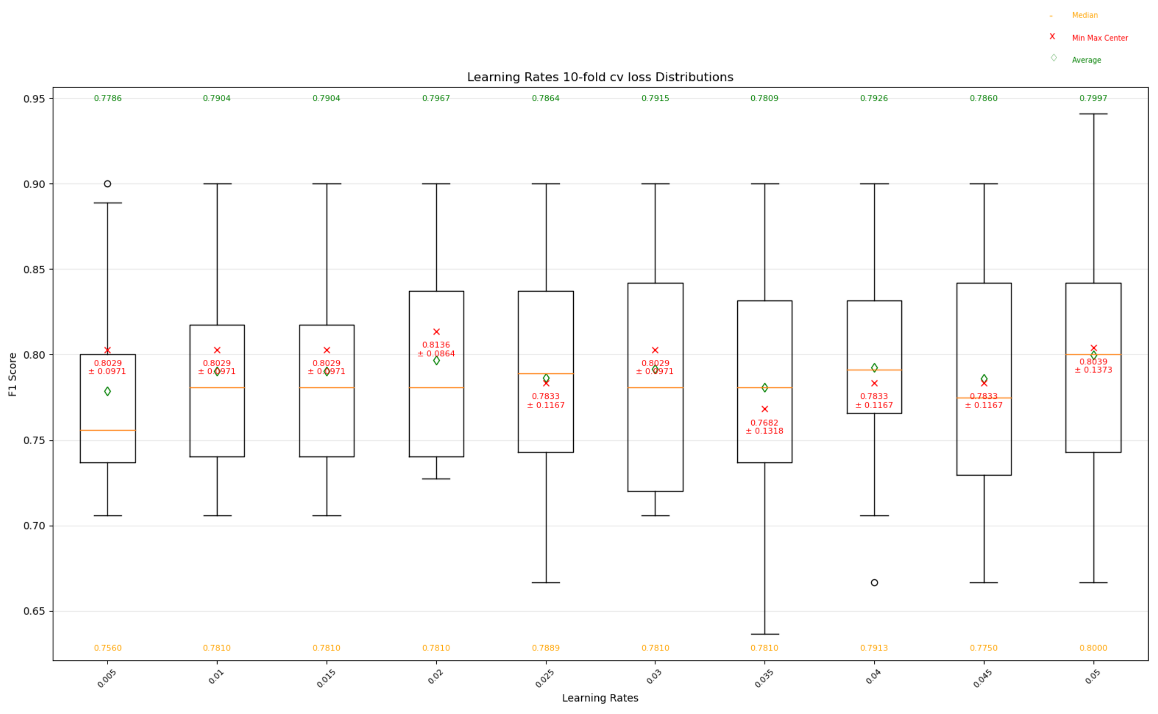


Figure 2: Best F1 Scores Models with different lr

ANN for Regression

Data pre-processing

All the 12 features were normalized into range of [0, 1] on their own feature space before they feed into ANN. Similarly, the data is split into 10 fold to test the robustness and consistency of the learning ability of the ANN designed.

Hyperparameter Tuning

Similar to the classification, we included learning rate as our hyperparameter. Addtionally, we also include weight decay for regularization in this experiment.

Model Architecture and Selection

Our chosen model have 7 layers with numbers of node corresponded to [8, 48, 32, 16, 32, 8, 1], there is only one output as our label is a single number. ReLU activator are used between the hidden layers. We used early stopping to select the final model.

Loss Function and Optimizer

We use the very common cost function for regression which is mean square error.

$$Loss = \frac{1}{N}\sum_{i=1}^{N}(h(x) - Y)^2$$

Furthermore, an extra regularization is added to prevent overfitting by penalising high weight values.

$$Loss = \frac{1}{N}\sum_{i=1}^{N}(h(x) - Y)^2 + \lambda\sum_{j=1}^{M}{w_j}^2$$

The loss optimizer that we used is Adam optimizer, additionally we did some tuning to the Adam optimizer parameter instead of using the default given. We choose 0.799 for the $\boldsymbol{beta1}$ parameter because it produce the lower Root Mean Square Error (RMSE) and loss score than the default. The choice for value $\boldsymbol{beta1}$ is arbitrary, we did not use hyperparameter tuning because it will take too much time and computation to finish.

Evaluation

We used RMSE as our performance metric. Box plots below (Figure 3) show the best RMSE result acquired after performing 10-fold cross validation with different learning rate. The table 6 shown is the best RMSE scores in each fold with 5000 epoches of training.

The learning rate of 0.045 and weight decay of 0.025 return the best model in our range of search. It contain the lowest average RMSE scores, 4.0904 and its average loss is also one of the lowest with a value of 4.9374.

Folds	1	2	3	4	5	6	7	8	9	10
RMSE	3.8373	4.2918	3.7205	3.4146	3.6695	5.0615	3.8219	3.7209	4.3911	4.9746

Table 3: Folds Performance of $lr = 0.045$, $wd = 0.025$ Model

	RMSE
Average	4.0904
Range	4.2380 ± 0.8235
Median	3.8296

Table 4: Performance of $lr = 0.045$, $wd = 0.025$ Model

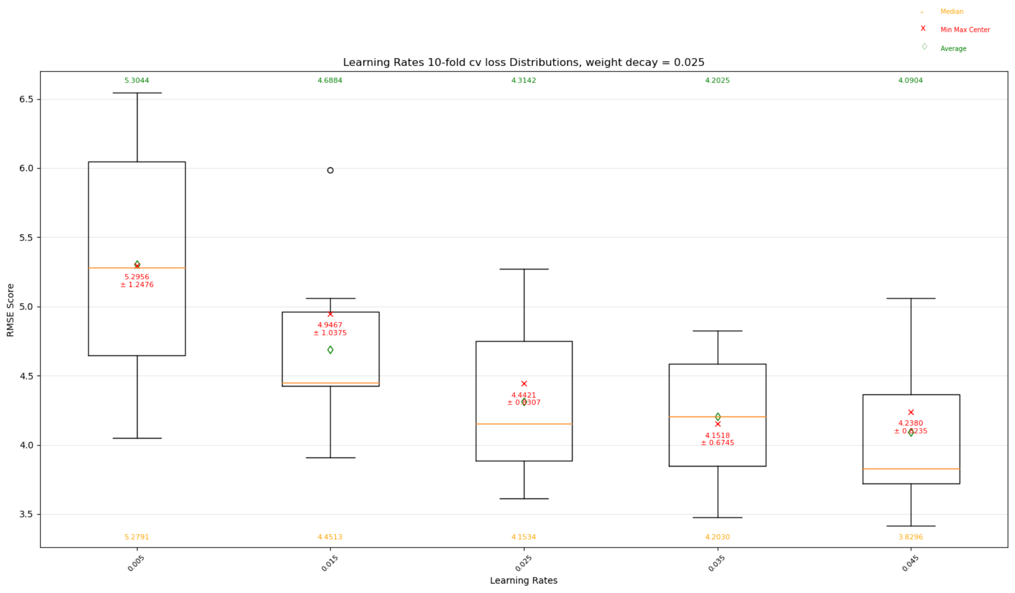


Figure 3: Best RMSE Score Models with different lr

Discussion

Overcoming Over-fitting

We have added an extra regularization term into the loss of our regression training for the main goal of preventing over-fitting. We also randomise our data set so each fold of the dataset is a good generalization sample of the truth task. This can prevent overfitting because the train set is not biased to certain group of data. Finally, we uses early stopping to choose best model before it overfitting, overfitting in long training is unavoidable in our experiments because the amount of data in our experiment is low.

SVM, Decision Tree and ANN

Table below show the different performance of F1 score from different models where linear SVM has the highest F1 score among other SVM. We believe that ANN performs the best as it has the highest F1 score among all the models.

Model	**F1 score**
SVM (linear)	0.654426
Decision Tree	0.690919
ANN	0.7967

Table 5: Model Performance for Classification

Model	**RMSE**
SVM (Polynomial)	3.769308
Decision Tree	5.682935
ANN	4.0904

Table 6: Model Performance for Regression

Based on results shown in the table above for classification, SVMs performance was not optimal. This might be due to the low number of instances which in the data set that we had chosen only has 299 instances, which against the fact that SVM relies on the data to be support vector for better generalization.

Decision tree performs well for classification and outperforms SVM, they are easier to implement and interpret. It also require less effort for hyperparameter tuning. However, decision tree is not suitable for regression problems from the observed poor performance in the table above. It is due to the nature of regression tree need as many lead node as the number of predictions.

ANN performs very well for both for classification and regression because it can work relatively well with a low number of instances. ANN has the potential to have a lower RMSE value than SVM if we have the time to fully optimise it. One of the con of ANN is that it is very sensitive to hyperparameter and it could take a very long time to find a settings that optimise an ANN.

From our results, we identified that the results from SVM and decision tree can be interpreted and explained clearly when compared to ANN which are typically black box systems. While they can learn abstract representations of a dataset, these representations are hard to interpret by human analysts. This means that while neural networks can, in principle, perform accurate predictions, it's unlikely that we'll obtain insights on the structure of a dataset through them.

Repository

Complete implementation of MATLAB code for SVM, Decision Tree and Python code for ANN and images of training results are all available on this repository. https://github.com/teoshibin/COMP3009_ML_CW