COMP9417 Project

Multitask Machine Learning (MML)

Prepared By Group:
We're going to be Genshin Impact masters

Huayang Xie (z5393197)
Chaorong Lei (z5405778)
Yinan Yan (z5454519)
Wenzheng Bian (z5469536)
Zelong Hu (z5507000)

2024.04

1 Introduction

In the medical system, there are lots of data have been generated every day, especially in some medical trial processes. All these data need to be analysed before it can be used to help patients. To make better use of these data results, we need to utilize a tool like machine learning to analyse the data first and then predict a clear and reliable result to help medical researchers make next decision.

In this project, the original dataset has 111 features, including different types such as binary, categorical and continuous, which gives us more challenges when we pre-process data. In addition, due to the privacy issue, we don't know what the real meaning of each feature is, and it also make it difficult for us difficulties to have a better understanding of features. Usually, it's much easier to do a single target prediction, but for this task, we need to generate multiple target variables to predict, which means multitask machine learning required to predict multiple results simultaneously.

Our approach is to utilize multitask machine learning to analyse medical data and help to predict several medical conditions. Rich Caruana mentioned that "MTL improves generalization by leveraging the domain-specific information contained in the training signals of related tasks."[i] We will use the knowledge we learned to explore different models for classification based on the data we received. After evaluation and comparison, we'll have a model that can generalize well for unseen data and achieve relevant goals based on the current training dataset.

2 Data Analysis

After receiving data, the first step is reviewing data and have an initial understanding of data. There is total 111 features and 1000 observations in the training data set (X_train). We also noticed that some features and samples have missing values which needs to be addressed before model training process. Therefore, we started to process these missing data first.

2.1 Remove missing features and samples

From figure 1, we can easily see the feature 6 and 87 almost missing 90-100% of data, and for feature 33 and 34, there are more than 60% data are missing. This information gives us confidence to delete these features in the original data set without having a big impact on the final prediction.

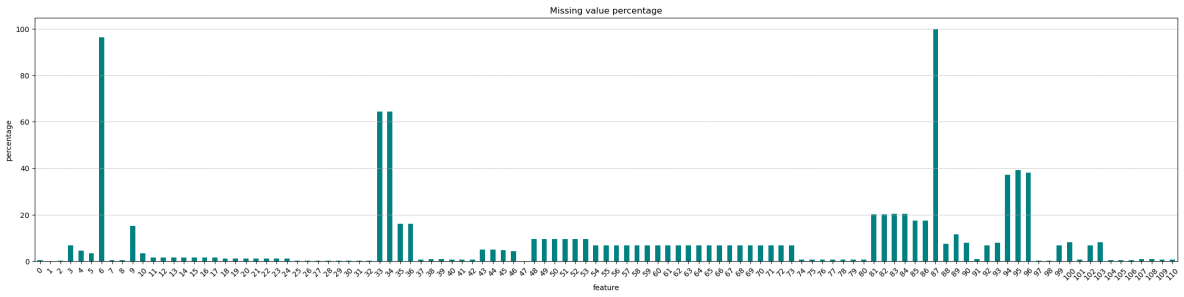

Figure 1: Percentage of missing values of all features

We have set the threshold for high missing rate features to 60% and high missing rate samples to 20%. Remove features and samples with high missing rates from X_train, and simultaneously remove the corresponding high missing rate samples from Y_train and high missing rate features from X_test.

In addition, the features that only have unary value will be deleted as it's not contributed to the model training. After that, we have cleaned dataset with shape of X_train shape (913, 95) and Y_train shape (913, 11).

2.2 Feature Classification and Naïve Imputation

From the images below, we found three different types of features: binary, categorical and continuous. Therefore, we classified the features based on their value. For example, if the value is only 0 or 1, the feature will be classified as binary (Figure 2, Left). The missing value of this binary type of feature will be filled with the most frequent value of the existing value. In addition, if the feature is categorical (Figure 2, middle), the value will be filled with the median value of existing value. Last, if the feature is continuous (Figure 2, right), the value will be filled with the mean value of the existing value.

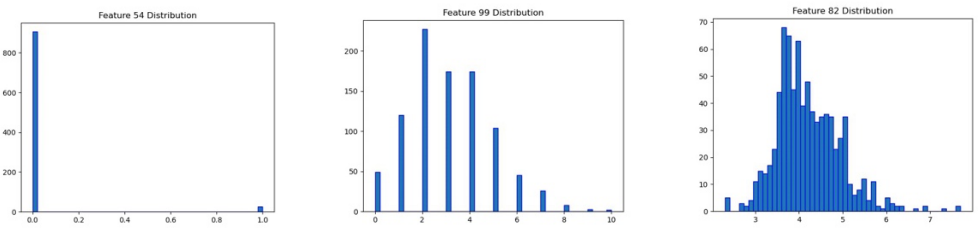

Figure 2: Example of Different types of features

2.3 Other Imputation Method

Except for the naive imputation, we also tested following other methods to impute missing values.

- KNN imputation. From a test, Olga Troyanskaya demonstrated that "KNN impute appears to provide a more robust and sensitive method for missing value estimation than SVDimpute, and both SVDimpute and KNNimpute surpass the commonly used row average method (as well as filling missing values with zeros)." [ii]

- Linear Regression Imputation. Pedro mentioned that “Linear regression can be used if the variable dependency follows a linear relationship.”[iii]
- Random Forest Imputation
- Support Vector Machine Imputation. Bhattacharyya mentioned that “In recent years, some works have extended the standard formulation of support vector machines (SVMs) for handling uncertainty in the input data.”[iv]

2.4 Process Outliers of Data

As Min-Wei stated, “if the observed data contain some noisy information or outliers, the estimations of the missing values may not be reliable or may even be quite different from the real values.”[v] Last step, we have analyzed the outliers of each feature when we detected any noise values, we used similar strategies of imputation to adjust the noise values to the new values, including Naïve imputation, KNN imputation, Linear Regression imputation, Random Forest imputation and support vector machine imputation. Figure 3 shows one example of an outlier processing result that would help to reduce the noise data.

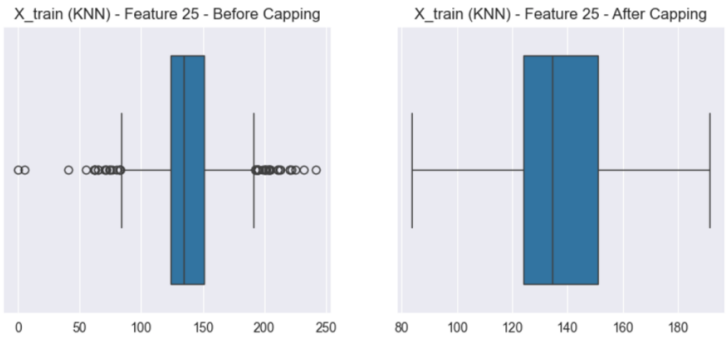

Figure 3: Example of the outlier processing result

2.5 Feature Selection

In addition, for the single training model, we also generated a dataset that considered the feature relevance of each label. Started analysing features using mutual information classification and then remove irrelevant features that are not related to the labels. For example, for the label 2, there are 48 relevant features and 47 irrelevant features. After deleting 47 irrelevant features for label 2, we’ll test whether this feature selection will affect final result.

3 Methodology

To make a better prediction, we have chosen three models to use in this project: Logistic Regression, Random Forest, and Neural Network.

3.1 Logistic Regression

Logistic regression is a fundamental statistical model in machine learning, and it is usually used for binary classification problems, which is suitable for this project. In addition, the binary cross entropy loss function is used in the logistic regression for binary classification which is required for this project. However logistic regression is not a good model to be used for multitask project without further amendments. Therefore, we choose the logistic regression model to construct 11 separate models to predict values. In the data pre-processing stage, we also selected different features for each label instead of using all the features. During this separate training process, this data set will be tested to compare with the dataset without selecting features. In the logistic regression model, there are several hyper-parameters can be used for tunning process.

Class Weight

Class weight can provide different values for different classes in the training process. In the default setting, the class weight will be 1 for each class, which is a balanced setting. However, we can adjust the class weight to improve the performance of the minor classes when the predicted classes are imbalanced. Different class weights {0: 1, 1: 11}, {0: 1, 1: 12}, {0: 1, 1: 13}, {0: 1, 1: 14}, {0: 1, 1: 15}) have been tested in the training process to find a better value for training.

Max Iter

Max iter is the maximum iteration that the solver will take to converge for the result. If max iter is too low, the model cannot reach the converge which will raise a waning during the training process. Therefore, we have chosen several max iter values (5000, 6000,7000,8000,9000,10000) to test, then get the best max iter value for each model.

C

C is a positive floating value to control the degree of regularisation. Lower value of C will apply strong regularization which will have impact of feature weight. We use Grid search method (a search method to find the optimal hyper-parameter) to find the best parameter of C for each model.

Penalty

Penalty is the parameter to define the norm used in the regularisation process. There are four options for penalty: l1, l2, elasticnet (both l1 and l2 added), and none (no penalty is added). We also used grid search to find the optimal penalty for each model.

Solver

Solver is the algorithm that will be used in the optimisation problem including lbfgs, liblinear, newton-cg, newton-cholesky, sag, and saga. For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and ‘saga’ are faster for large ones. For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss.[vi] In the end, grid search was used to find the optimal solver for models.

3.2 Random Forest

Random Forest is another good model for solving this multitask binary class problem. It can improve the overall performance and stability of the model by constructing multiple decision trees and combining their predictions. It also has better overfitting resistance than a single decision tree. Random forest does not need to make any assumptions about the distribution of data, which makes it very flexible and effective in practical applications. Using Random Forests does reduce the need for additional code and complex model tuning to a large extent, as it comes with some built-in features such as automatic feature selection. By tuning these parameters, we can easily control the complexity and training time of the model.

We started with Random Forest's built-in automatic feature selection. It randomly selects a subset of features as the Random Forest constructs the decision tree, increasing the diversity of the model and helping the model to identify and focus on those features that will be most influential in the model's predictions as the process is automated. In addition, we also use a resampling technique, which is a method to deal with the problem of class imbalance in a dataset. We tested both oversampling and undersampling techniques to find which one was better for this project.

In the end, we adjusted the hyperparameters of the random forest model to check whether it could improve the model performance. The parameters are shown below.

N estimators:

The number of trees. Increasing the number of trees usually improves the stability and performance of the model, but at the same time increases the computational cost and time.

Max depth:

The maximum depth of the tree. Deeper trees can model more complex data relationships but may also lead to overfitting. Shallower trees may not capture enough important patterns in the data, leading to underfitting. Choosing the appropriate depth is key to ensuring good generalisation capabilities.

Min samples split:

The minimum number of samples required to split the internal nodes. This parameter determines when the tree stops splitting nodes. Larger values prevent the model from learning "noise" from the data but may also ignore useful information.

Min samples leaf:

The minimum number of samples a leaf node must have. This parameter helps generalise the model by ensuring that leaf nodes are not too specific. Larger values can reduce overfitting, but too large may lead to underfitting.

Class weight:

The weight of the category. For datasets with uneven categorisation, more attention can be given to a small number of categories by adjusting the category weights, which helps to improve the model's prediction performance on a small number of classes.

3.3 Neural Network (NN)

Neural Network is another model that can solve classification problems. And it was designed naturally to deal with multitask learning. Ruder et al. (2019) said that “While MTL is being more frequently used, the 20-year old hard parameter sharing paradigm is still pervasive for neural-network based MTL”[vii]. The Neural Network model can capture complex nonlinear relationships between features. When dealing with medical features and prediction targets, the representation required for the prediction task can be learned directly from the data through end-to-end learning. The model can not only predict a single prediction target based on feature values during training but can also learn possible associations between different prediction targets.

A clear description of hyper-parameter tuning is below if applicable. In our neural network model, the experimental hyperparameters are.

Batch size:

Batch size refers to the number of samples used to calculate model errors and update weights in each training iteration. Adjusting the batch size can affect the convergence speed and generalization ability of the model.

Learning rate:

The learning rate determines the step size for each weight update and determines the magnitude of the weight adjustment during the gradient descent process.

Dropout rate:

Dropout is a regularization technique used to prevent overfitting in neural networks. During training, the dropout rate determines what proportion of neurons will be randomly and temporarily dropped from the network.

Activation function:

Activation functions are used in neural networks to introduce nonlinear factors, allowing the network to learn and perform more complex tasks. Common activation functions include ReLU (linear rectifier unit), Sigmoid, Tanh, etc.

Optimizer:

The optimizer defines how the model updates its weights, i.e., what algorithm is used to perform gradient descent or other optimization algorithms. Common optimizers include SGD (stochastic gradient descent), Adam, RMSprop, etc.

Loss function:

The loss function measures the difference between the model's predicted value and the true value and is the goal of optimization during the training process. Common loss functions include mean square error (MSE), cross-entropy loss (Cross-Entropy), Hinge Loss, etc.

For these hyperparameters, we will combine task requirements and network search strategies to determine the optimal values of these hyperparameters.

3.4 Evaluation metrics

We chose the F1 score as the evaluation metric. The F1 score is a very commonly used and important evaluation metric, especially effective in classification problems when dealing with datasets with unbalanced categories, and it just fits the dataset of our task. The formula for calculating the F1 score is:

$$F1 = 2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}$$

Where Precision is the ability of the model to correctly predict positive categories, and Recall is the ability of the model to find all positive category samples.

4 Results

4.1 Logistic Regression

First, before applying any parameter adjustment, we get the initial result, which will be used as the base case. The average loss is 0.58775, and the average f1_score is 0.05478.

The initial result shows the F1 score is very low, and some of score close to 0, which means the model hasn't been trained very well, the performance of the minor classes needs to be improved by adjusting class weight. After testing, we chose {0: 1, 1: 15} as the class weight for the next step. After applying class weight, the average loss is 1.35818, and the average f1_score is 0.39709. The f_1 score has been improved by updating the class wight. However the loss value also has been increased.

Next, we started to use grid search method to find the optimal parameters for each model. It shows that max_iter is 5000 for every single model, C value is varying from 0-3, penalty is l2 for every model, and the solver is lbfgs for each model.

After finding all the optimal parameters for each model, we start to train final models. And the final result of logistic regression shows in figure 4 : f_1 score is 0.39805 (binary cross entropy loss 1.36523). This will be our base result used to compare with other models later.

```
Task 0: Binary Cross-Entropy = 1.3933544642250812, F1 Score = 0.42608695652173906
Task 1: Binary Cross-Entropy = 1.291521349468821, F1 Score = 0.30985915492957744
Task 2: Binary Cross-Entropy = 1.4274244564637093, F1 Score = 0.3055555555555556
Task 3: Binary Cross-Entropy = 1.2488401859148235, F1 Score = 0.42105263157894735
Task 4: Binary Cross-Entropy = 1.3022566911196134, F1 Score = 0.408695652173913
Task 5: Binary Cross-Entropy = 1.2339340841188622, F1 Score = 0.4910714285714286
Task 6: Binary Cross-Entropy = 1.3118271771650742, F1 Score = 0.3474178403755869
Task 7: Binary Cross-Entropy = 1.3902498037489597, F1 Score = 0.3891402714932127
Task 8: Binary Cross-Entropy = 1.576931978026156, F1 Score = 0.5182186234817814
Task 9: Binary Cross-Entropy = 1.359513865592892, F1 Score = 0.41860465116279066
Task 10: Binary Cross-Entropy = 1.481706678284432, F1 Score = 0.3428571428571429
Average loss is 1.365232794011675
Average f1_score is 0.39805090079106137
```

Figure 4: Final result of the Logistic Regression Model

4.2 Random Forest

The first time, the model was trained without any adjustments. Same with logistic regression, the f1_score is very low. Then the undersampling technique was found to be more suitable for the dataset and model of this task. Therefore, start first training and get the result: average loss is 0.70012 and the average f1_score is 0.34236.

Due to the concern that the classes in the dataset are uneven respectively, resulting in poor model training, we introduce the parameter class_weight. In order to allow the model to automatically and comprehensively traverse all hyperparameter combinations, GridSearchCV is introduced into the model for tuning; GridSearchCV is a method for systematically traversing multiple hyperparameters and determining the optimal parameter through cross-validation. After setting the hyperparameter values and the number of cross-validation times several times, we obtained the following experimental results: the average loss 0.70116 and the average f_1 score is 0.34013.

It was found that the final generated evaluation data became worse after the model training was over. We suspected that the introduction of a hyperparameter caused the model performance to become poor, so we felt that the hyperparameters were called in one by one and found that the introduction of the parameter class_weight led to a reduce performance of the model. So, we removed the parameter and trained again. We found that the performance improved, which shows the final result of Random Forest in figure 5 (f_1 score is 0.34986 (binary cross entropy loss 0.69842)) based on current dataset and time constraint.

```
Task 0: Accuracy = 0.5291970802919708, Binary Cross-Entropy = 0.6914450691799889, F1 Score = 0.4317180616740088
Task 1: Accuracy = 0.5, Binary Cross-Entropy = 0.7014777814529327, F1 Score = 0.3045685279187817
Task 2: Accuracy = 0.48905109489051096, Binary Cross-Entropy = 0.7049089285831891, F1 Score = 0.3269230769230769
Task 3: Accuracy = 0.5328467153284672, Binary Cross-Entropy = 0.6909766493680733, F1 Score = 0.37254901960784315
Task 4: Accuracy = 0.5, Binary Cross-Entropy = 0.7035737693113594, F1 Score = 0.3251231527093596
Task 5: Accuracy = 0.5364963503649635, Binary Cross-Entropy = 0.6905382475968059, F1 Score = 0.38048780487804873
Task 6: Accuracy = 0.4963503649635036, Binary Cross-Entropy = 0.7030358749779863, F1 Score = 0.34285714285714286
Task 7: Accuracy = 0.5291970802919708, Binary Cross-Entropy = 0.6940102022381406, F1 Score = 0.32460732984293195
Task 8: Accuracy = 0.48905109489051096, Binary Cross-Entropy = 0.7014019246100937, F1 Score = 0.36936936936936937
Task 9: Accuracy = 0.46715328467153283, Binary Cross-Entropy = 0.7099530860553845, F1 Score = 0.3113207547169811
Task 10: Accuracy = 0.5437956204379562, Binary Cross-Entropy = 0.6913025366780085, F1 Score = 0.358974358974359
Average Accuracy across all tasks: 0.5103
Average F1 Score = 0.3498635090429003
Average Binary Cross-Entropy (Binary tasks) = 0.6984203700047238
```

Figure 5: Final result of Random Forest Model

4.3 Neural Network

In the nn_model function, we build a neural network model by using keras library. The model includes an input layer (hidden layer), dropout layer, hidden layer, dropout layer, and output layer. The two hidden layers have 500 and 100 neurons, respectively, and use the ReLU activation function. Including a dropout layer can help us capture the complexity of the data while avoiding overfitting. We also use the Sigmoid activation function, adam optimizer, and binary cross-entropy loss function in the model based on the binary classification task.

To explore the hyperparameters of batch size, learning rate, and dropout rate, we used a grid search strategy and compared data sets processed in different ways. However, because of the small number of data samples, we added five times cross-validation (n_splits) by dividing the training set and validation set multiple times to avoid learning errors caused by data imbalance.

Explorations in other hyperparameters include:

	Part 1	**Part 2**	**Part 3**	**Part 4**
Batch_sizes	[8, 16, 32, 64]	[16, 32]	[16]	[16]
Learning_rates	[0.01, 0.001, 0.0001, 0.00001]	[0.00001]	[0.00001, 0.000005, 0.000001 0.0000005, 0.0000001]	[0.0000005, 0.0000004, 0.0000003, 0.0000002, 0.0000001, 0.00000005, 0.00000001]
Dropout_rates	[0, 0.05, 0.1, 0.2, 0.3]	[0.3]	[0.3, 0.4, 0.5]	[0.4, 0.5]
Epochs	100	100	300	300

(Dataset used: X_outliers_knn / X_outliers_naive / X_outliers_regression_naive / X_outliers_regression_reverse / X_outliers_regression_rf / X_outliers_regression_svr)

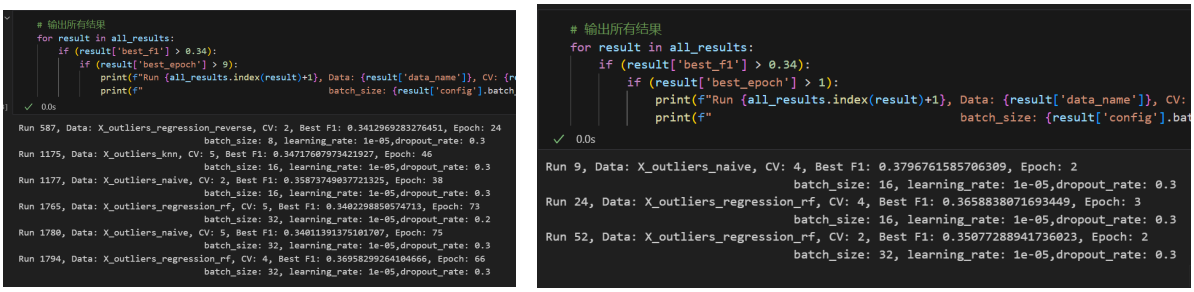

Figure 6: Part 1(Left) & Part 2(Right) Results

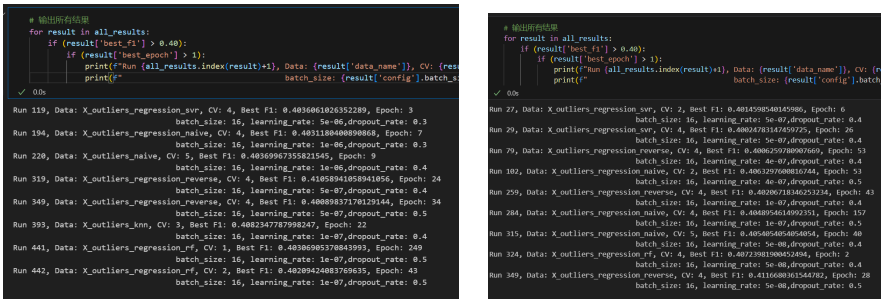

Figure 7: Part 3(Left) & Part 4(Right) Results

After the above 3330 tests and comparisons of different hyperparameter combinations, no more combination tests were conducted due to time constraints. Finally, we selected the following parameters as model prediction parameters: batch_size 16, learning_rate 1e-06, drop rate 0.3, and epoch 9. The F1_score is 0.41699.

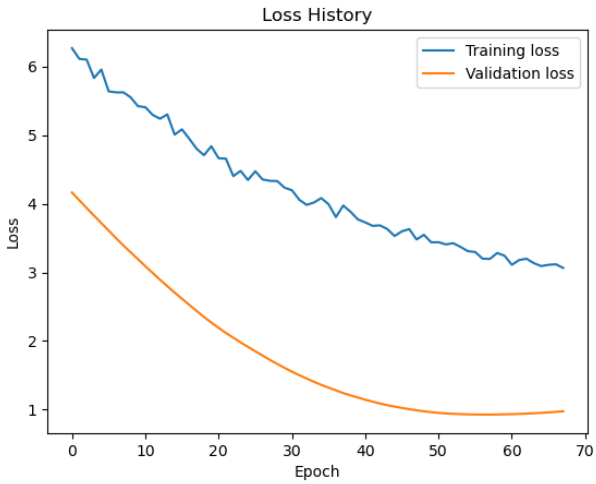

Figure 8: Final Loss value of Neural Network model

5 Discussion

Based on the results (f_1 and binary entropy cross loss) regenerated from these three models, logistic regression has an f_1 score of 0.39805 (binary cross entropy loss 1.36523), random forest has an f_1 score of 0.34986 (binary cross entropy loss 0.69842), and Neural Network model generated the result of f_1 score 0.41699 based on the current dataset. Although the logistic regression model has better f_1 score, its loss value is higher than the random forest. Therefore, we can get the result that the neural network model is better than the random forest model, and the random forest model is better than the logistic regression model in this project. The neural network was used for our final final model to predict Y_test from given X_test dataset. This result demonstrates that the neural network has strengths, including the ability to capture complex nonlinear relationships in data and learn the representations required for prediction tasks directly from the data through end-to-end learning. In addition, we also demonstrated different hyperparameters such as batch size, learning rate, dropout rate, activation function, and optimizer do have impacts on model performance.

6 Conclusion

The final model selection was based on the stable performance shown in 5-fold cross-validation, especially on the dataset after handling outliers. The performance of neural network models is evaluated in terms of

loss value, F1 score, precision, and recall and demonstrated that neural network is suitable for medical diagnostic problems due to their emphasis on false positives and negatives.

In addition, by comparing different models and their configurations, we found that dropout rate and learning rate have a significant impact on neural network model performance. A higher dropout rate helps neural network models avoid overfitting, especially when the amount of data is small. Although a lower learning rate increases the time for model convergence, it can also improve the final prediction performance. We also observed that neural network models perform better on datasets with good feature selection and outlier handling, which illustrates that data preprocessing steps can play an important role in building effective multi-label classification models.

Future improvements

- Explore grid search or random search for more comprehensive hyperparameter tuning over a wider range of values.
- Implement feature importance analysis to remove or give more weight to certain features based on their impact on model performance to improve predictions.
- Using other data preprocessing methods or feature engineering may improve model performance.
- Research other multi-task learning frameworks to improve learning efficiency and performance.

7 References

[i] Rich Caruana (1997). Multitask Learning. Machine Learning, 28, 41-57

[ii] Olga Troyanskaya, et al. (2001). Missing value estimation methods for DNA microarrays. Bioinformatics, Vol17 no.6 2001,p.520-525.

[iii] Pedro J. Garc´ıa-Laencina, Jose´-Luis Sancho-Go´mez, An´ıbal R. Figueiras-Vidal. Pattern classification with missing data: a review. Neural Comput & Applic (2010) 19:263–282

[iv] Bhattacharyya C, Shivaswamy PK, Smola AJ (2004) A second order cone programming formulation for classifying missing data. In: Saul LK et al (eds) Adv Neural Inf Process Syst 17. MIT Press, Cambridge, pp 153–160

[v] Min-Wei Huang, Wei-Chao Lin, Chih-Fong Tsai, Outlier Removal in Model-Based Missing Value Imputation for Medical Datasets. Hindawi, Journal of Healthcare Engineering, Volume 2018, Article ID 1817479, 9 pages

[vi] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[vii] Sebastian Ruder. (2019). An Overview of Multi-Task Learning in Deep Neural Networks, p.11.