**Report of Modelling Earthquake Damage**

**Group10: Chen Zhenglin, Gu Yuqing, Liu Jiayang, Liu Wuyue, Zhou Tong**

**(Project Contribution Equally)**

**1 Description**

**1.1 Problem Description**

In our project, our task is to predict the level of damage to buildings caused by the 2015 Gorkha earthquake in Nepal, based on the database that contains several aspects of building, including the location and construction.

**1.2 Dataset Description**

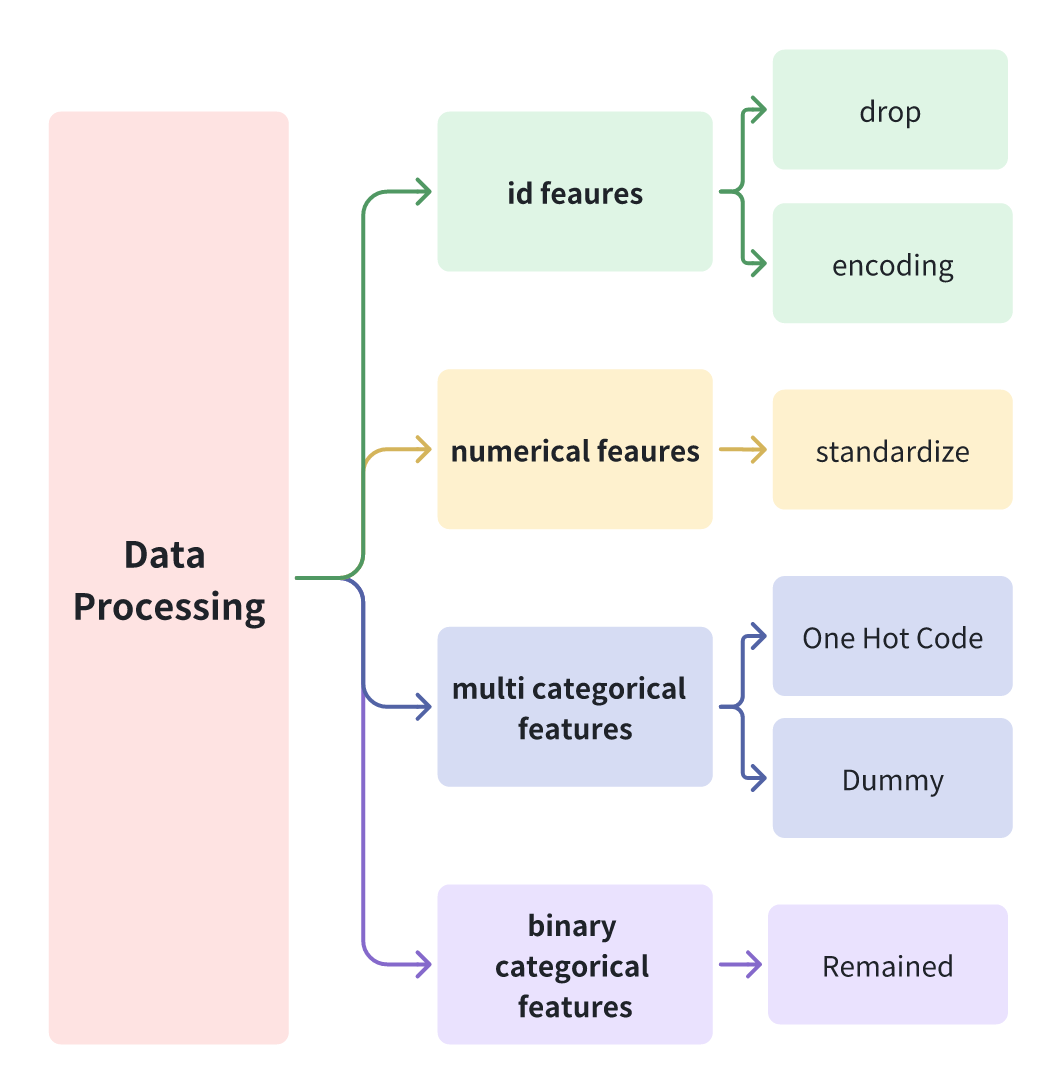
Our original Dataset contains 260,601 and 86,868 samples in the train and test datasets respectively. There are 39 features in the dataset before feature engineering and data processing.

**2 Data Processing**

After observation of the dataset and the meaning of each feature, we separate the 39 original data into 4 types: id features, numerical features, multi categorical features and binary categorical features. follows are the details:

| **Feature Types** | **Feature List** |
| --- | --- |
| id features | 'building\_id', 'geo\_level\_1\_id', 'geo\_level\_2\_id', 'geo\_level\_3\_id' |
| numerical features | 'count\_floors\_pre\_eq', 'age', 'area\_percentage', 'height\_percentage' |
| multi categorical features | 'land\_surface\_condition', 'foundation\_type', 'roof\_type', 'ground\_floor\_type', 'other\_floor\_type', 'position', 'legal\_ownership\_status', 'plan\_configuration' |
| binary categorical features | 'has\_superstructure\_adobe\_mud', 'has\_superstructure\_mud\_mortar\_stone', 'has\_superstructure\_stone\_flag',  'has\_superstructure\_cement\_mortar\_stone', 'has\_superstructure\_mud\_mortar\_brick',  'has\_superstructure\_cement\_mortar\_brick', 'has\_superstructure\_timber', 'has\_superstructure\_bamboo', 'has\_superstructure\_rc\_non\_engineered',  'has\_superstructure\_rc\_engineered', 'has\_superstructure\_other', 'count\_families', 'has\_secondary\_use',  'has\_secondary\_use\_agriculture', 'has\_secondary\_use\_hotel',  'has\_secondary\_use\_rental', 'has\_secondary\_use\_institution',  'has\_secondary\_use\_school', 'has\_secondary\_use\_industry',  'has\_secondary\_use\_health\_post', 'has\_secondary\_use\_gov\_office',  'has\_secondary\_use\_use\_police', 'has\_secondary\_use\_other' |

Then, we process each type of features with different methods, the conclusion of the methods can be seen in the following picture:



Picture 1: Processing Methods

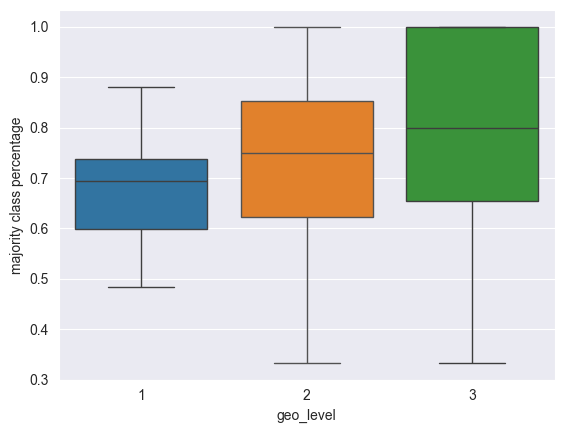
As can be seen in the picture, for the id features, we drop the building\_ids, while using a method called encoding to process the rest 3 id features. As for the numerical features, we use the z-score fumula to standardize them. We used two methods to treat the multi categorical features, for the classification algorithm, we generated the One Hot Code from them, while we generated the dummy variables from these features for the regression algorithms, as the One Hot Code may result in the completely collinear problem in the regression algorithms, which may lead biased estimators. Finally, for the binary categorical features, we do not do any process.

**2.2 TargetEncoder**

**2.2.1 Data Analysis**

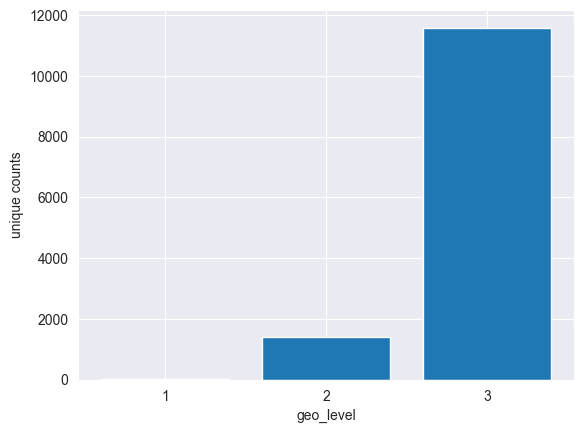
Based on the actual situation, it is speculated that the geographical location of the building is highly correlated with the damage level. After conducting exploratory data analysis, the speculation was confirmed.

As shown in the figure, we classified by geo\_level\_id and counted the proportion of majority classes in each geo\_level\_id group. It can be observed that the geographic location of the building is valid information and should be retained.



Picture 2: Proportion of Majority Classes in Each geo\_level\_id Group

There are 31 categories of geo\_level\_id\_1, and tens of thousands of categories of geo\_level\_id\_3. If geo\_level\_id is treated as a category variable, No matter whether we use the tree method or other methods with one-hot encoding, there will be a problem of complexity explosion.



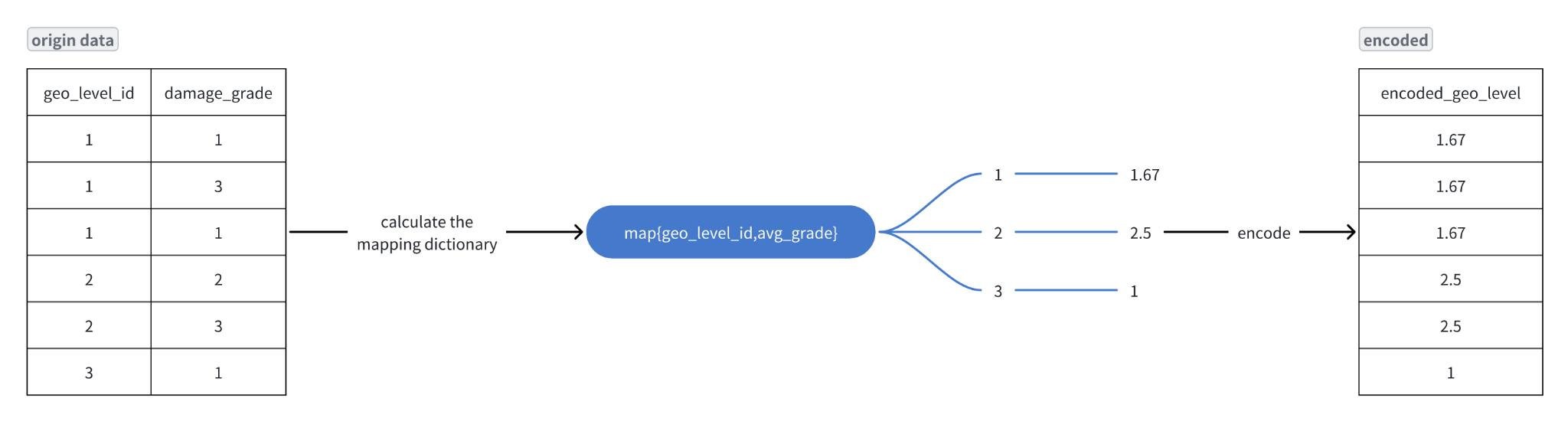
Picture 3: Category Unique Counts of geo\_level\_id

**2.2.2 Encoding Process**

So we introduced TargetEncoder, an encoding method for converting categorical variables to numeric variables, to handle geo\_level\_id as much as possible without losing information.

The basic idea of Target Encoding is to use statistical information of target variables to encode each category. The encoding process is as follows:

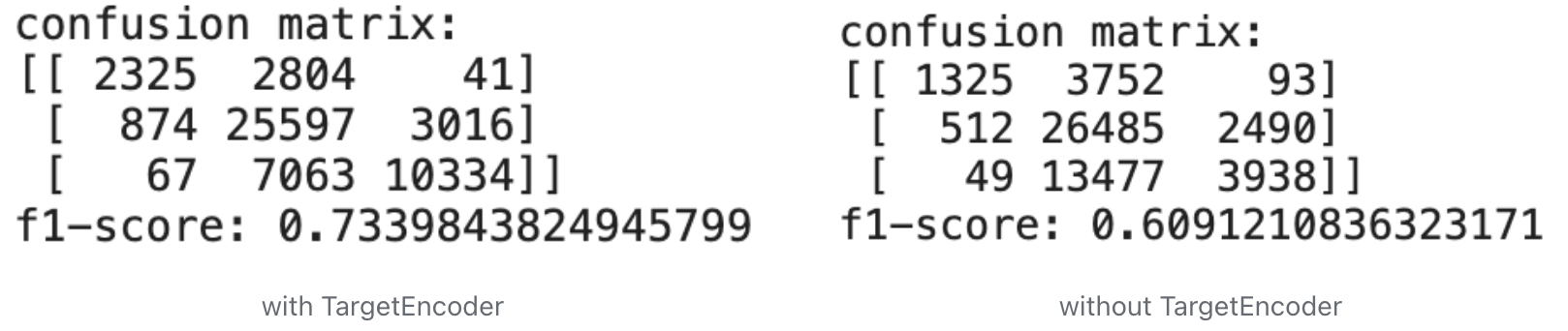
1. For each geo\_level\_id category, calculate the average of the damage.
2. Create a dictionary that maps each category to the corresponding average.
3. Replace the original classification feature with its corresponding encoding value.
4. Use these encoded values as features during Model Training.



Picture 4: Process of Target Encoding

**2.2.3 Encoding Effect**

We compared the effect of training FFNN without geo\_level\_id and instead using TargetEncoder. We can see that the improvement effect of TargetEncoder is obvious, which shows that TargetEncoder can effectively handle many categorical variables and help the model understand the relationship between features and targets.



Picture 5: Comparison of Model Performance with/without TargetEncoder

**2.3 feature selection**

**2.3.1 PCA**

We used PCA to process the data, reduced the dimension of 68 columns to 23 columns, and compared the training time and effect before and after dimensionality reduction.

|  | Original Data with One-Hot  encoding | PCA Data |
| --- | --- | --- |
| num of features | 68 | 23 |
| f1-score on FFNN | 0.733 | 0.733 |
| training time on FFNN | 45.44 seconds | 39.97 seconds |

Table 1: Comparison of Original Data and Data Processed with PCA

From the comparison, we conclude that PCA can reduce the dimension effectively without losing information and accelerate training speed.

**2.3.2 Decision Tree Selection**

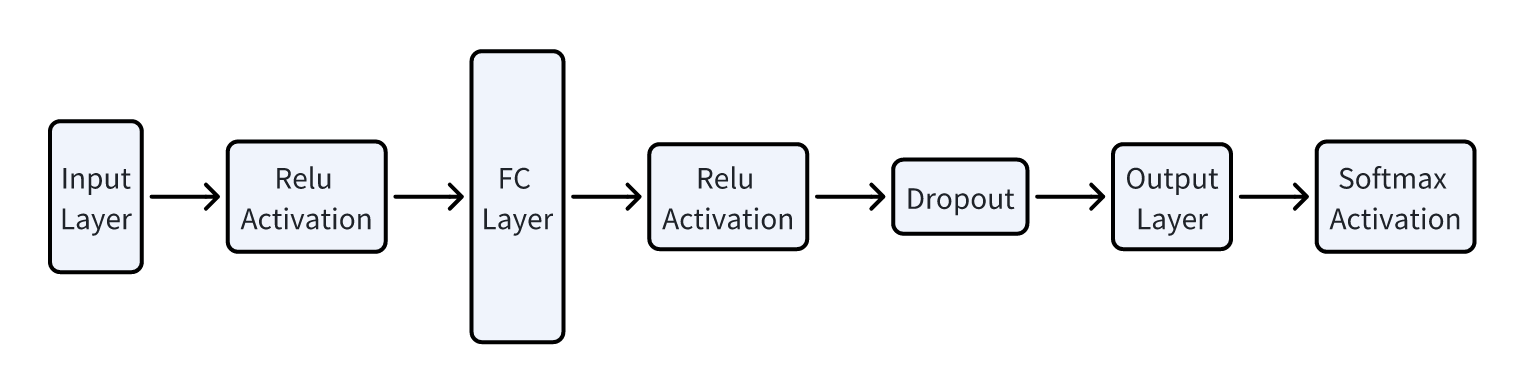
We also use decision tree to recognize the feature importance, the result shows that geo\_level information is important for model training. The below table shows the five top features with importance scores.

| geo\_level\_3\_id | geo\_level\_2\_id | foundation\_type\_i | age | geo\_level\_1\_id |
| --- | --- | --- | --- | --- |
| 0.776478 | 0.048230 | 0.042189 | 0.031537 | 0.021215 |

Table 2: Feature Importance in Decision Tree

**3 Experimental Study and Analysis**

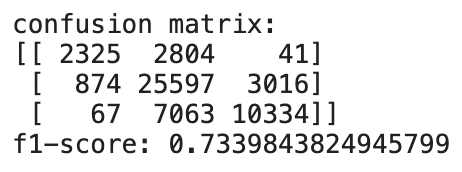
**3.1 FFNN**

****

Picture 6: Structure of FFNN

FFNN is well-suited for complex classification tasks with non-linear decision boundaries due to their layered architecture and activation functions.

Our FFNN model consists of an input layer, an intermediate layer, dropout layers, an output layer, and activation functions. Since we are solving a classification problem, we use softmax activation as the output and cross entropy as objective function.

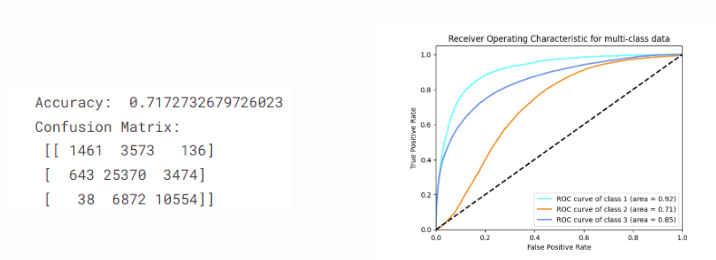
****

Picture 7: Performance of FFNN

**3.2 Lasso logic regression**

Lasso logic regression is basically the logic regression with Lasso(L1) regulation to deal with the classification problem. We utilize the LogisticRegression function in scikit-learn, with parameter L1 to enable the L1 regulation, which is Lasso, and for our multi-class classification problem, we do the Lasso logic regression for each category, so call”one-vs-rest”, and calculate the accuracy of each of them. We also use grid-search during the process of finding the best lambda.

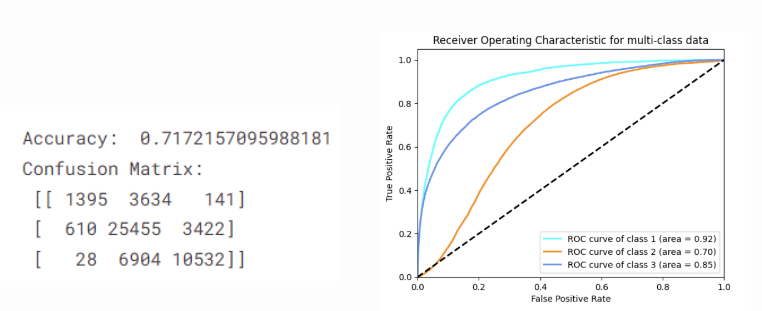
It has benefits like High accuracy, resistance to overfitting, Improved Generalization, and Model Optimization(achieved by cross validation and hyperparameter tuning).



Picture 8: Performance of Lasso Regression

**3.3 Ridge logic regression**

Compared to Lasso:In contrast to Lasso (L1) regularization which can reduce some coefficients exactly to zero, Ridge (L2) regularization reduces the size of coefficients but typically does not set any of them to zero. This means that while Lasso can perform feature selection by eliminating non-contributory predictors, Ridge will include all predictors in the final model but with shrunk coefficient values.



Picture 9: Performance of Ridge Regression

**3.4 K-Nearest Neighbors (KNN)**

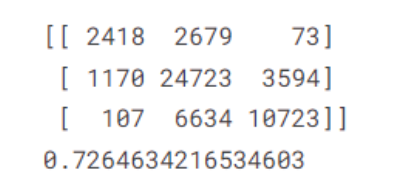
K-Nearest Neighbors (KNN) in Python: We used the KNN regression model from scikit-learn to fit the data, and before this, we used the grid-search cross validation to find the best K value between 3 and 15.



Picture 10: Performance and Hyperparameter Selection of KNN

**3.5 Decision tree**

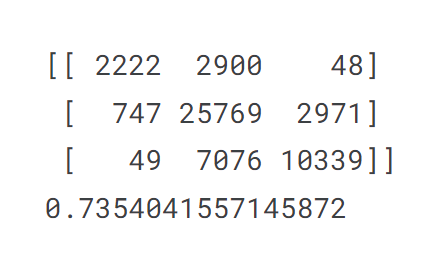
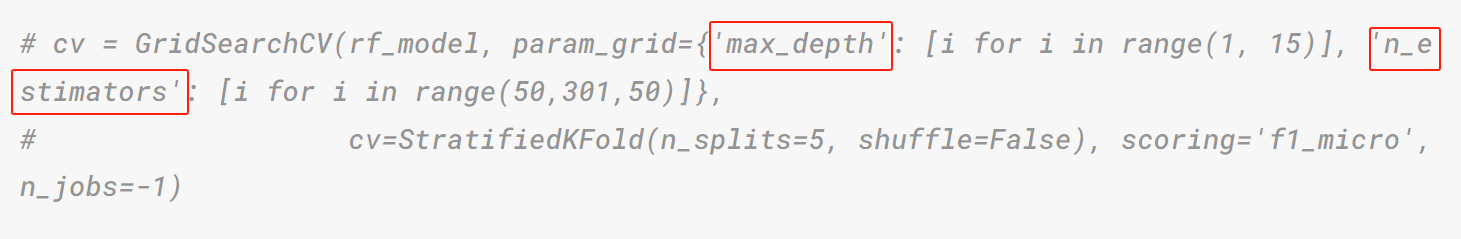
Decision Tree in python: scikit-learn library is used to implement a decision tree classifier, which is a supervised learning algorithm frequently employed for classification tasks. It has benefits like Robustness to Outliers, Applicability to Large Datasets, Accuracy, and Ease of Interpretation and Visualization.



Picture 11: Performance of Decision Tree

**3.6 Random Forest**

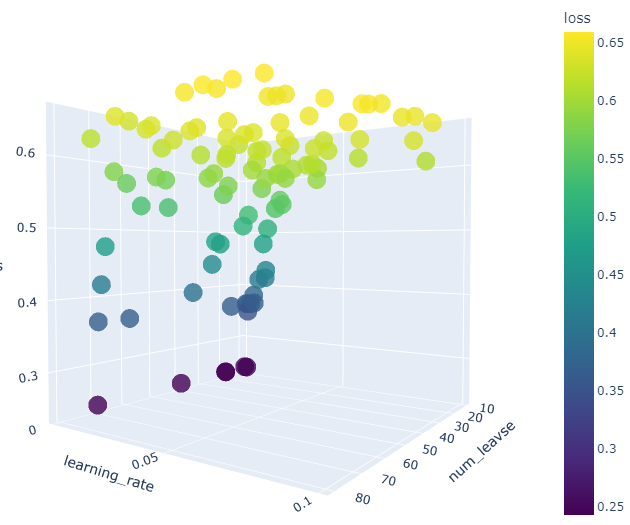
Random Forest is an ensemble method for machine learning, which combines multiple decision trees for accurate and robust predictions. RandomForestClassifier() in Python utilizes scikit-learn to create, train, and evaluate Random Forest models for classification tasks. In our study, we apply this method and regard max\_depth and n\_estimators as hyperparameters, tuning them to find the best model. The pictures below show how we define the grid of candidate values for selected hyperparameters, and the second one is the confusion matrix and f1-score of the Random Forest model with best parameters.



Picture 12: Performance and Hyperparameter Selection of Random Forest

**3.7 LightGBM**

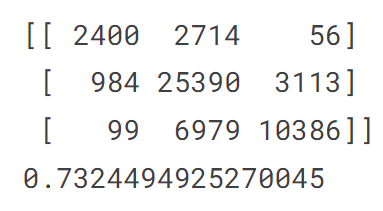
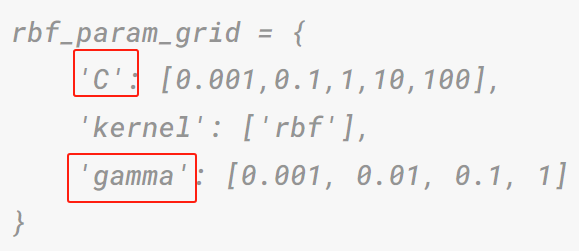
LightGBM is a gradient boosting framework that offers high-performance, scalability, and efficient training for machine learning tasks. It is designed for efficiency and speed, which uses a histogram-based learning method that can lead to faster training times compared to traditional gradient boosting methods. By importing ‘lightgbm’ in Python, we can easily activate GPU acceleration, which can significantly speed up the training process, especially for large datasets. The picture below shows the performance of LightGBM model, with two hyperparameters as x- and y-axis, and the loss of models as z-axis. A model's performance is considered better when it achieves a lower loss.

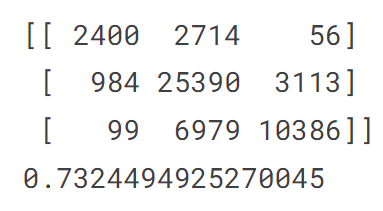


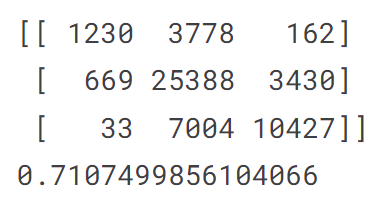
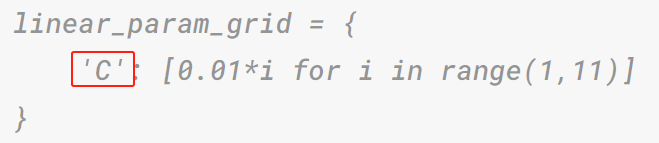
Picture 11: Performance of LightGBM

**3.8 Support Vector Machine**

Support Vector Machine (SVM) is a powerful and versatile machine learning algorithm for classification and regression tasks. SVM aims to find the hyperplane that maximizes the margin between classes, and is effective for both linear and non-linear data. When using a radial basis function (RBF) kernel with SVM, it allows the model to handle non-linear decision boundaries. Otherwise, we can use a linear kernel for linear boundaries. The pictures below show both the hyperparameters selection and model performances of RBF and Linear SVM.



Picture 11: Performance and Hyperparameter Selection of RBF SVM



Picture 11: Performance and Hyperparameter Selection of Linear SVM

**3.9 Hyperparameter tuning**

For different models, we tried different tuning methods. GridSearch is suitable for some relatively simple models but it is time-consuming. For some complex models, we tried an AutomatedSearch Method from an open-source library Hyperopt. It applied the principle of Bayesian Optimization, making it easier to find the best combination of hyperparameters.

**4 Result Comparison**

We use the F1 score as the main standard when judging the performance of models.

From the plot, we can find that Random Forest has the highest F1 score(0.7354) hence the best performance among all models. The performance of each model is relatively close. Meanwhile, LGBM is the worst-performing model(0.6834).

|  | Logistic - Ridge | Logistic - Lasso | KNN | FFNN | Decision Tree | Random Forest | LGBM | RBF SVM | Linear SVM |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F1-Score | 0.7172 | 0.7170 | 0.7228 | 0.7340 | 0.7265 | 0.7354 | 0.6834 | 0.7324 | 0.7107 |

Table 3: Model Training Result with F1-Score

**5 Future Directions**

We still have some outlook on how we can better analyze this in the future.

1. In this data analysis training, we only performed classification training and ignored the hierarchical relationship information in the three labels, which can be taken into account in the future. For example, can treat y as numeric variable, and use regression method to solve the problem.
2. There may be more room for improvement in tuning hyperparameters. For example, more levels of detail tuning after selecting the range parameter range.
3. Some other models we can also try: e.g. Bayesian classifiers, LDA for downscaling and classification of mult-iclassification problems.

**6 Conclusion**

In summary, our study has effectively achieved its research objectives. We generated nine models with robustness, most of which performed well on multi-classification problems. Our evaluation methodology, utilizing the F1-score, has enabled a comprehensive performance assessment of various models.

Although we encountered a number of challenges during our research, we systematically addressed them through extensive and rigorous investigations, including using the target encoder method to encode category variables such as geographic information, using different CV methods to improve CV efficiency, and using GPUs to improve overall model training speed.

In conclusion, our research has been conducted with thoroughness, producing a strong foundation for future improvements and findings in this domain.