Midterm Report

Jialu Li

I am a data scientist working at the Department of Transportation (DOT). With the increasing number of road accidents for different reasons, the department wants to find out what factors influence the survival rate of road accidents. I am responsible for solving this problem using machine learning approaches.

**Dataset:**

For this task, I use the Road Accident Survival Dataset available on Kaggle: https://www.kaggle.com/datasets/himelsarder/road-accident-survival-dataset/data. The dataset has 200 observations and includes variables such as *Age*, *Gender*, *Speed\_of\_Impact*, *Helmet\_Used*, *Seatbelt\_Used*, and *Survived*. I use *Survived* as the target variable and other variables as features. I chose these features because they are all that are available in this dataset, and all of them are critical to the survival rate.

**Feature engineering:**

To complete the task, I first work on feature engineering. I first find missing values in each variable. All variables are complete except for *Gender,* with one missing, and *Speed\_of\_Impact*, with three missing. I fill the value with the missing *Gender* using the most frequent gender in the dataset, which is female. I fill the values with the missing *Speed\_of\_Impact* using the median in the dataset, which is 71. Then, I encode the categorical variable into values 1 and 0, which is mandatory for machine learning classification/regression models to process them. Survival is already encoded into numeric; therefore, I deal with the other three categorical variables: *Gender, Helmet\_Used*, and *Seatbelt\_Used.* For *Gender*, I code Male as 1 and Female as 0. For *Helmet\_Used* and *Seatbelt\_Used*, I code Yes as 1 and No as 0. After handling all the features, I split the dataset into X for features and y for the target, then split X and y into train and test sets with a test size of 0.2. Train and test X sets are then scaled using a standard scaler for further processes.

**Model selection:**

For the prediction task, I choose three models: Random Forest, Logistic regression, and KNN. In this project, I deal with binary classification, with the target variable as ‘Survived’ or ‘Not Survived’, therefore, the first model I select is Logistic regression. Logistic regression is easy to interpret for binary classification as it provides the probability of an event occurring. It works better with a linear relationship and performs fast even with larger datasets. However, it does not work well with non-linear relationships and is sensitive to outliers, so when it assumes a non-linear relationship as linear, the prediction will not be as good.

To overcome the issue of the possible non-linear relationship, I select the Random Forest Classifier model. I choose a Classifier instead of a Regressor because of the binary nature of the target variable in the dataset. Random forest is an ensemble learning method that fits multiple decision tree classifiers and averages them to improve the predictive accuracy and control over-fitting. It works well with both categorical and numeric data and can model complex, non-linear relationships. One feature of the model is that it can provide the importance ranking of features. However, Random Forest is less interpretable compared to logistic regression, and its performance highly depends on hyperparameter selection, making it less flexible and computationally intensive compared to logistic regression.

The other model I select is K-Nearest Neighbors (KNN). It is a non-parametric model, meaning that it does not make assumptions about the data distribution and is suitable for data with undetermined relationships. It can also handle non-linear relationships and work well with small datasets, suitable for our small dataset. The difference of this model is that it is a distance-based classification algorithm that makes predictions based on the majority class of its nearest neighbors. One limitation of this method is that it is sensitive to data imbalance and outliers, and it requires data to be normalized. KNN is also computationally expensive because it has to store all data to calculate distance, but this is not a very big problem in our case as our dataset is small in size.

**Hyperparameter tuning:**

I then used the grid search algorithm for model hyperparameter tuning to identify the best group of parameters for each model. For each model, I chose four parameters, and each has three values for cross-validation except for one parameter for KNN.

For the logistic regression, I choose four hyperparameters: C, solver, max\_iter, and tol. C indicates the inverse of regularization strength; smaller values indicate stronger regularization strength. The values are chosen as 0.1, 1 (default), and 10; different values exert different regularization strengths for the model, which can influence the predictive performance by dealing with overfitting or underfitting. Solver is the algorithm used in the optimization problem. Solvers are chosen as lbfgs (default), liblinear, and saga. lbfgs is the default optimization choice. It works well with medium and large datasets, but it only allows for L2 or None regularization. liblinear works well with small datasets but does not deal with multiclass problems, which is also not the case for the accident dataset. It can incorporate L1 and L2 regularization. saga, on the other hand, works well with bigger datasets, adept at faster computation, it works well with all regularizations for logistic regression. I choose these three algorithms to suit different situations, such as multiclass, regularization, and dataset size. Due to the different levels of suitability for regularization terms, I must leave the penalty parameter as the default to avoid errors. max\_iter and tol represent the thresholds for logistic regression to stop, the maximum number of iterations, and tolerance for stopping criteria. I choose these two criteria is that in the previous choice of parameters, an error of non-convergence occurs, therefore, I intend to twist the stopping criteria a little to see how it may alter the prediction performance. The values of max\_iter are 100 (default), 500, and 1000, and for tol is 0.01, 0.0001 (default), and 0.00001.

For Random Forest, I choose four hyperparameters: n\_estimators, max\_depth, min-samples\_split, and criterion. n\_estimators represent the number of trees in the forest; higher values can lead to better prediction performance to a certain extent. The values chosen for n\_estimators are 50, 100 (default), and 200, I want to see whether different numbers of trees can lead to better performance. max\_depth is the maximum depth of the tree, indicating when the tree leaves stop expanding. The values chosen include None (default), 10, and 20. None max\_depth means that the trees will continue to expand until each leaf is pure for classification. 10 and 20 means that trees will stop expanding after reaching 10 or 20 levels. I choose smaller values for this parameter is that since the dataset is small, smaller stopping criteria may better suit the dataset. min-samples\_split is the minimum number of samples required to split an internal node; a lower number means more splits and more complex trees, and a higher number means fewer splits and simpler trees. Too high a value may lead to overfitting, and too low may cause underfitting. Therefore, in my case, I choose values 2 (default), 5, and 10. because the dataset is rather small and may not require too many splits, 5 and 10 are chosen to fit this situation. Criterion is the function to measure the quality of a split. I chose the gini (default), entropy, and log\_loss functions for model evaluation. gini measures the impurity of each class, aiming at minimizing the impurity, it works well with smaller and balanced datasets. While entropy works well for imbalanced datasets, log\_loss works with probability-based predictions because it provides smoother decision boundaries rather than hard splits. Both are more computationally expensive than gini, and log\_loss is more sensitive to overfitting. Each of these criteria can be applied to the accident dataset while fitting slightly different conditions; I intend to see which one can better fit our case.

The last model is KNN, I also chose four hyperparameters: n\_neighbors, weights, metric, and leaf\_size. n\_neighbors is the number of neighbors to consider when determining the class of a new data point. The different number of neighbors can influence the final class prediction by changing the majority class. Therefore, I choose the values 3, 5 (default), and 10 to determine the best fit for our dataset. weights determine the weight function to be used during the prediction. Uniform (default) and distance weights are chosen. Uniform weights weigh each neighbor equally, while distance weights weigh each neighbor by the inverse of their distance to the data point, meaning that closer neighbors will have a larger influence. Different weights will influence the final prediction of the data class. Therefore, I selected two different weights for consideration. metric is used to determine the distance. I choose three distance metrics: Euclidean (default), cosine, and Manhattan. Euclidean computes the straight-line distance between points, cosine similarity computes the angle between two vectors, and Manhattan distance measures distance by summing absolute differences along each axis. These different distance calculation methods will generate different distance predictions, influencing the model’s performance. The last parameter is leaf\_size, which determines how many points are stored in each leaf node. Smaller leaf sizes will result in deeper trees and slower computation, and larger leaf sizes will produce shallower trees with faster computation. I choose the values of 10, 30 (default), and 100, spanning over small, moderate, and large leaf sizes, hoping to examine the best size for the dataset.

**Evaluation metrics:**

After the hyperparameter tuning, the best hyperparameter groups are used for each model to predict the test set. Then, each model’s performance is evaluated with metrics including accuracy, precision, recall, f1, and Matthews correlation coefficient (MCC). These metrics are commonly used to evaluate binary classification models and report different aspects of the model. Accuracy calculates the ratio of correct predictions, including true positives and true negatives, to the total number of predictions. It is useful when datasets are mostly balanced. Precision calculates the proportion of correct positive predictions (true positives). Recall calculates the proportion of actual positives that are correctly identified (TP/(TP+FN). The F1 score provides a balance between precision and recall. MCC takes into account all four confusion matrix values (TP, TN, FP, FN), it is particularly useful when the datasets are imbalanced. Including all metrics can provide a comprehensive review of the model’s performance, considering all the confusion matrixes. For this accident dataset, I rely more on the accuracy metric for the model comparisons because, in this dataset, the positive and negative classes are approximately balanced, and we do not incline towards minimizing false positive or negative. It is the most straightforward metric in comparison.

**Model performance:**

In this section, we have the model performance. The best hyperparameters for Random Forest are: criterion: gini, max\_depth: None, min\_samples\_split: 2, n\_estimators: 50. The test accuracy of Random Forest is 0.5000, precision is 0.4286, recall is 0.3333, f1 Score is 0.3750, and MCC is -0.0316. For logistic regression, the best hyperparameters are: C: 0.1, max\_iter: 100, solver: liblinear, and tol: 0.01. the test accuracy of Logistic regression is 0.5250, precision is 0.4667, recall is 0.3889, f1 Score is 0.4242, and MCC is 0.0259. The best hyperparameters for KNN are: leaf\_size: 10, metric: cosine, n\_neighbors: 3, weights: uniform. The test accuracy for KNN is 0.5000, precision is 0.4375, recall is 0.3889, f1 Score is 0.4118, and MCC is -0.0205. We can see from the results that Logistic regression produces the highest values across all metrics. Therefore, we can conclude that Logistic regression is the better model to predict the survival rate in road accidents. However, I do not recommend the DOT to use this model to understand factors influencing survival rate because it is clear that the accuracy of the model is only slightly greater than throwing a coin, which is reflected by the MCC score, and both recall and precision is less than 0.5.

**Future plan:**

Traditional machine learning methods did not perform well in predicting the survival rate based on given factors. In the future, I will try to find a dataset with more factors related to road accident survival, and I intend to try neural networks, which is more powerful compared to traditional machine learning models.