**FLOOD PREDICTION IN MALAWI PREDICTION IN MALAWI USING ENSEMBLE METHODS: RANDOM ROREST CLASSIFIER AND GRADIENT BOOSTING CLASSIFIER**

Richard Luke, bscds2720, Data Science, Mzuzu University of Malawi.

**ABSTRACT**

This report proposes the use of ensemble methods, specifically Random Forest Classifier and Gradient Boosting Classifier, to improve flood prediction in districts of Malawi. Results showed good accuracy in predicting flood events per district.

**1. INTRODUCTION**

In this part of the project I worked with MySQL database called weather\_db. The database had 5 tables, when the tables where imported into a data frame called df I generated 11 features and 1 label. The shape of df was 220, 12.

**2. METHODS**

2.1 prepare Data

In this part of the modelling process I wrote a python script called eda.py that was connecting to weather\_db and pulling all the data into a dataframe called df. eda.py uses 6 packages namely: mysql.connector, pandas, seaborn, matplotlib, basemap and warnings.

2.2 Explore Data

The second step involved using eda.py script in 2.1 above, with it I performed exploratory data analysis by first describing the data i.e min, max, mean, percentiles etc. Then I visualized my longitudes and latitudes to determine if the data I obtained from weather\_db was for Malawi using basemap package. After I confirmed that the data was for Malawi from the map, I checked for missing values. When I found missing values in 2 features named distance\_from\_river and rainfall, I replaced the missing values with each features mean. Then I checked the distribution of rainfall in the df and plotted the response as a histogram. Then I checked which districts had the most incidents of flooding and the result I plotted as a bar chart. Then I checked which rivers had most flooding incidents and the results I plotted as a bar chart. Finally, I plotted all the districts that had a flooding incident in the dataset differenciating period by a heatmap using basemap. At this point I felt that I had an overview of the data I was working with therefore I felt confident to move to the third step

2.3 Modelling

The third step was the actual modelling the data to find the best ensemble method that fitted the data using either Random Forest Classifier or Gradient Boosting Classifier in the first place. In the second place I performed cross-validation and hyperparameter tuning to try and improve the models. This was performed using the sklearn package

2.3.1 modelling

2.3.1.1 Gradient Boosting Classifier

I denoted my input data as X and my output data by y. X contained the following features 'Longitude', 'Latitude', 'elevation', 'distance\_from\_river', 'rainfall', 'urban\_population', 'land\_area', 'density' and 'year', while y contained all labels for flooding. Therefore, for each training example i is I represented it by a pair (x i , y i ). The gradient boosting classifier constructed an ensemble of decision trees by minimizing a loss function L(y, F(x)) where F(x) was the ensemble of decision trees. The loss function measured how well the ensemble of decision trees fitted the training data. The loss function we defined as:

L(y, F(x)) = 1/n \* sum(Li(y i , F (x i )))

where Li is a loss function that measured the difference between the predicted value F(xi) and the true value yi for training example i.

Therefore using the gradient boosting classifier I initialized the prediction F0(x) to the average of the training data y. This was the initial prediction before any decision trees were added to the ensemble. For t = 1 to T, where T was the number of decision trees to be added to the ensemble:

a. Computed the negative gradient of the loss function with respect to the current prediction F(x), which is denoted by r(t)i = -[L(y i , F (x i ))/F (x i )]F (x = t − 1) (x i )

b. Trained a decision tree h(t)(x) to predict the negative gradient r(t)i for each training example i.

c. Computed the step size t using line search or heuristics to minimize the loss function: t = argmin L(y, F(x)=F(x=t-1) + \*h(t)(x))

d. Updated the ensemble of decision trees by adding the new tree: F(x) = F(x) + t\*h(t)(x)

e. Returned the final prediction F(x).

2.3.1.2 Random Forest Classifier

I represented each decision tree as a function f(x) that maps the input vector x to an output label y. The function f(x) was constructed by recursively partitioning the feature space into smaller regions using decision rules based on the training data. Each partition was then associated with a leaf node in the tree, which contained a label for the corresponding region. Then the final prediction of the Random Forest was given by averaging the predictions of the individual trees. f(x)= (1/N) \* Σ( f i (x)). I minimized the loss function by measuring the difference between the predicted labels and the true labels of the training data. I expressed the loss function as: L(y, F(x)) = sum(Li(y i , F (x i ))).

2.3.2 cross-validation and hyperparameter tuning

I used k-fold cross-validation to evaluate the performance of the Random Forest Classifier and Gradient Boosting Machines. I used the KFold function from the scikit-learn library to split the data into k folds, and the cross val score function to compute the performance metric for each fold. The function returned an array of scores, one for each fold. I then took the mean of these scores as the estimate of the model’s performance. Hyper-parameter tuning involved finding the optimal values for the hyper-parameters of the two machine learning algorithms to achieve the best performance on the given dataset. In my case, I tuned the hyper-parameters of the Random Forest Classifier and Gradient Boosting Machines using cross-validation to optimize their performance. The hyper-parameters for Random Forest Classifier are the number of trees in the forest (n estimators), the maximum depth of each tree (max depth,) the minimum samples per split(min samples split) and the the minimum number of samples required to split an internal node(min samples leaf). For Gra-

dient Boosting Machines, the hyper-parameters are the number of boosting stages (n estimators), the learning rate (learning rate) and the maximum depth of each tree (max depth).

2.4 Evaluating the model

2.4.1 Accuracy

This was calculated by dividing the number of correct predictions by the total number of predictions made.

2.4.2 Recall

We measured the ability of a classifier to correctly identify positive instances. We calculated it as the ratio of true positive predictions to the total number of actual positive

instances in the dataset.

2.4.3 F1 Score

The F1 score measured the harmonic mean of precision and recall. This was achieved by using accuracy of the binary classification of the models by balancing both precision and recall.

**3. RESULTS**

The evaluation metrics for the project report were based on the performance of two machine learning algorithms, the RandomForestClassifier and the GradientBoostingClassifier. The best parameters for the RandomForestClassifier were determined to be ’max depth’: 5, ’min samples leaf’: 4, ’min samples split’: 4, ’n estimators’: 100, with a resulting best score of 0.8864. For the GradientBoostingClassifier, the best parameters were ’learning rate’: 0.1, ’max depth’: 5, ’n estimators’: 100, and the best score was 0.9261. In addition to these parameters, the performance of the model was evaluated using confusion matrix, F1 score, and recall. The confusion matrix for the models revealed that the number of true positives and true negatives was 25 and 13 respectively, while the number of false positives and false negatives was 2 and 4 respectively. The recall score, which measures the proportion of true positives out of all actual positive cases, was found to be 0.7647. Finally, the F1 score, which is a weighted average of precision and recall, was found to be 0.8125. This metric provides an overall measure of the models’ accuracy, with a higher score indicating better performance.

**CONCLUSION**

Model training and model evaluation was successful.