FoML - VNB Hackathon 2024 Report

Laveena Herman - CS24MTECH11010

November 11, 2024

1 Introduction

This report presents the training and testing process applied to the agricultural field productivity dataset, prepared for the VNB Hackathon 2024. The dataset comprised **112,569 training instances** and **15,921 validation instances**. The primary objective was to develop a predictive model capable of accurately estimating field productivity. To achieve this, preprocessing techniques were employed to transform the data, addressing issues such as **missing values**, **outliers**, and **feature scaling**.

Following data preparation, machine learning models and algorithms like **Random Forest**, **Gradient Boosting**, **Logistic Regression**, **XGBoost**, **Support Vector Machines**, **K-Nearest Neighbors (KNN)**, **LightGBM**, and **CatBoost** were explored. Each model was fine-tuned using **hyperparameter optimization** to maximize performance. The models were evaluated based on their **F1 macro scores** on the validation set, providing a balanced measure of **precision** and **recall** across all classes. After a thorough comparative analysis, the **best-performing model** was identified.

2 Data Preprocessing

2.1 Feature Selection

- A **correlation matrix** was constructed to analyze relationships between features and the target variable.
- Columns with **low absolute correlation** (less than 0.01) with the target were dropped to reduce noise.
- The **percentage of null values** in each column was analyzed.
- Columns with more than **90% missing values** were dropped.
- **Redundant columns**, such as `WaterAccessPointCalc`, that did not contribute unique information were removed.

2.2 Feature Transformation

- Columns that counted occurrences, such as `NumberGreenHouses` and `PartialIrrigationSystemCount`, were converted to binary indicators (`HasGreenHouse`, `HasPartialIrrigationSystem`).
- Features such as `RawLocationID` and `TownID` were transformed using **frequency encoding**.
- The `FieldEstablishedYear` feature was modified by calculating `current year - FieldEstablishedYear`.

2.3 Handling Missing Values

- Null values were filled based on their **domain-specific nature**:
  - **0 or 1** was used where appropriate.
  - Columns with limited missing values were imputed with the **median** for numerical data and the **mode** for categorical data.

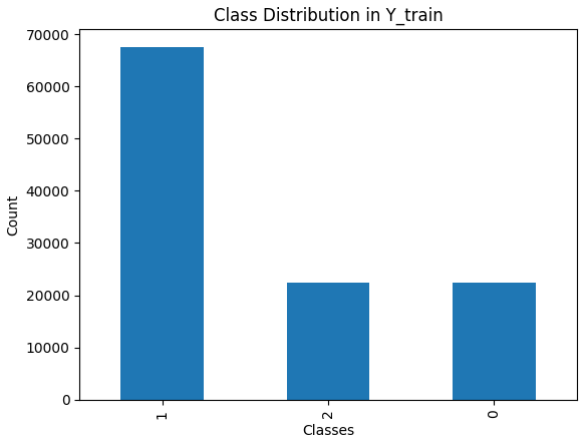

Figure 1: Class Distribution before sampling

2.4 Encoding Categorical Variables

- **Label encoding** was applied to categorical features with only **two unique values**.
- **One-hot encoding** was used for categorical variables with more than two unique values.

2.5 Class Imbalance

- Due to class imbalance, **random oversampling** was applied to augment the minority class.
- **SMOTE (Synthetic Minority Over-sampling Technique)** was used specifically for **gradient boosting** and **logistic regression** models, enhancing their performance on the minority class.

2.6 Dimensionality Reduction and Scaling

- **Principal Component Analysis (PCA)** was applied after oversampling. PCA was used to extract the most important features, reducing the dataset to **40 principal components** while retaining a significant portion of the variance.
- **Standard scaling** was applied prior to PCA to ensure all features were on the same scale. This step was crucial as PCA is sensitive to the relative scaling of features, and standardization ensured that each feature contributed equally to the principal components.

2.7 Experimentation

- **KNN imputer** and **groupby association** methods were also tried but later removed due to poor performance.

3 Models and their Performances

After the data was pre-processed, a number of models were tried on the given dataset. The models tried, the hyper-parameters given for the model and the F1 scores predicted are summarized in the table below:

S.No	Model	Hyper-parameters	F1 Score
1	Random Forest	n_estimators = 200 max_depth = 15 min_samples_split = 50 min_samples_leaf = 10 criterion = gini Used Random Oversampling	0.435
2	Random Forest	n_estimators = 300 max_depth = 20 min_samples_split = 70 min_samples_leaf = 15 criterion = gini Used Random Oversampling	0.435
3	Random Forest	n_estimators = 200 max_depth = 20 min_samples_split = 70 min_samples_leaf = 15 criterion = gini Used Random Oversampling	0.434
4	Random Forest	n_estimators = 200 max_depth = 12 criterion = gini Used Random Oversampling	0.433
5	Gradient Boost	random_state = 0 n_estimators = 300 learning_rate = 0.1 max_depth = 5 subsample = 0.8 max_features = 'sqrt' min_samples_split = 10 Used Random Oversampling	0.431
6	LightGBM	boosting_type = 'gbdt' n_estimators = 2000 learning_rate = 0.05 max_depth = 10 subsample = 0.8 num_leaves = 64 min_data_in_leaf = 50 reg_alpha = 0.1 reg_lambda = 0.1 min_split_gain=0.05	0.431

Table 1: Models and the F1 scores obtained on the validation dataset

S.No	Model	Hyper-parameters	F1 Score
7	LightGBM	n_estimators = 2000 learning_rate = 0.05 max_depth = 12 subsample = 0.75 num_leaves = 128 reg_alpha = 0.2 reg_lambda = 0.2 min_split_gain = 0.05 scale_pos_weight = 1.5 min_child_weight = 10 Used Random Oversampling	0.426
8	Logistic Regression	penalty = 'l2' C = 1.0 solver = 'lbfgs' class_weight = 'balanced' max_iter = 1000	0.400
9	Cat Boost	iterations = 100 learning_rate = 0.1 depth = 6 loss_function='MultiClass' SMOTE used for oversampling	0.399
10	Linear SVC (SVM)	C = 1.0 penalty = 'l2' dual = False tol = 1e-4 max_iter = 1000 random_state = 42 loss = 'squared_hinge' class_weight = 'balanced'	0.386
11	KNN	n_neighbors = 5 weights = 'uniform' SMOTE used for oversampling	0.383
12	XGBoost	random_state = 42 n_estimators = 200 learning_rate = 0.05 max_depth = 3 class_weight = 'balanced' SMOTE used for oversampling	0.365

Table 2: Models and the F1 scores obtained on the validation dataset

3.1 Models Explored

- **Random Forest:** Chose Random Forest for its robustness and ability to handle a high-dimensional dataset like this one, yielded a F1 Macro score of 0.435 making it one of the top-performing models.

- **Gradient Boosting:** Gradient Boosting stood out due to its iterative approach to minimizing prediction error. By combining it with SMOTE to address class imbalance, an F1 Macro score of 0.431 was obtained.
- **Logistic Regression:** Used as a baseline for comparison.
- **Support Vector Machines (SVM):** SVM was explored for its effectiveness in handling high-dimensional data and its flexibility with kernel functions.

4 Observations

From the model performance summarized in Table, several key insights were observed:

- **Random Forest** consistently delivered the highest F1 Macro scores, with the best configuration achieving an F1 score of **0.435**. This model proved to be robust across various hyperparameter settings and handled the dataset's complexity.
- **Gradient Boosting** demonstrated strong performance with an F1 score of **0.431**. However, when **SMOTE** was applied to address class imbalance, it resulted in lower accuracy compared to using **Random Oversampling**. The synthetic data generated by SMOTE did not align well with the model's learning process, leading to a slight drop in performance.
- **Logistic Regression** and **Support Vector Machines (SVM)** served as baseline models. Their performance lagged significantly behind Random Forest and Gradient Boosting.
- **LightGBM** showed competitive performance, with its best configuration achieving an F1 score of **0.431**. Despite its efficiency and ability to handle large datasets, it marginally underperformed compared to Random Forest.

5 Conclusion

Among the various models analyzed, **Random Forest** emerged as the best-performing model with an F1 Macro score of **0.435**. The optimal configuration for Random Forest included the following parameters:

- **n_estimators**: 200
- **max_depth**: 15
- **min_samples_split**: 50
- **min_samples_leaf**: 10
- **criterion**: gini

Gradient Boosting was a close second, achieving an F1 score of **0.431**. Its best configuration used the following parameters:

- **n_estimators**: 300
- **learning_rate**: 0.1
- **max_depth**: 5
- **subsample**: 0.8
- **max_features**: 'sqrt'
- **min_samples_split**: 10