

Malnutrition Risk Prediction ML Model

Machine Learning Project Documentation

Group members:

- Lina Ahmed
- Linda Adil
- Maha Abdalfedil
- Nada Ali

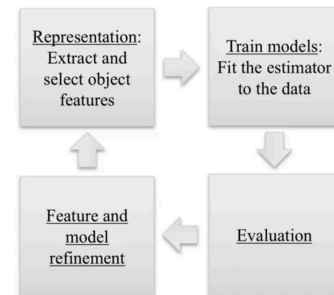
Model Refinement

1. Overview:

In the model refinement phase, our focus shifted from initial model creation to improving the performance and generalizability of the machine learning model.

This phase involved addressing issues such as **overfitting**, **improving model accuracy**, and ensuring that the model can make **reliable predictions** on unseen data. By refining hyperparameters, experimenting with new algorithms, and conducting rigorous validation, we aimed to enhance the predictive power of the model for identifying malnutrition and poverty risk levels in Least Developed Countries (LDCs).

These improvements are crucial in providing actionable insights to NGOs and policymakers targeting malnutrition interventions.



2. Model Evaluation

Initially, the random forest classifier produced promising results, achieving an accuracy of 1.00 with precision, recall, and F-scores close to 0.99 for all three risk classes (high, medium, low), these results also indicated signs of overfitting, as the model performed too well on the training data, suggesting it may not generalize well to unseen data.

Subsequent model iterations, which included the addition of more features, showed a slight decline in performance (accuracy dropping to 0.93, with precision ranging from 0.92 to 0.97). These metrics were visualized to identify areas for improvement and the ROC curve, which initially showed a straight line, confirmed the presence of overfitting, as the model struggled with balanced prediction across different classes. These observations prompted the need for refinement to improve generalization

3. Refinement Techniques

To improve the model and handle overfitting, we used the following techniques:

- **Encoding:** We encoded the country column as numerical values to utilize it in the clustering phase.

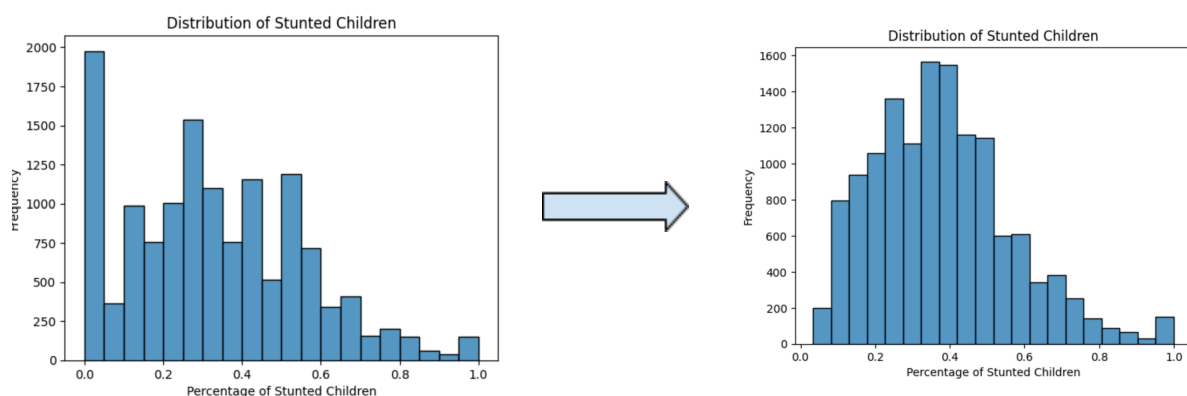
```
[99] from sklearn.preprocessing import OrdinalEncoder

# Create an instance of the OrdinalEncoder class
encoder = OrdinalEncoder()

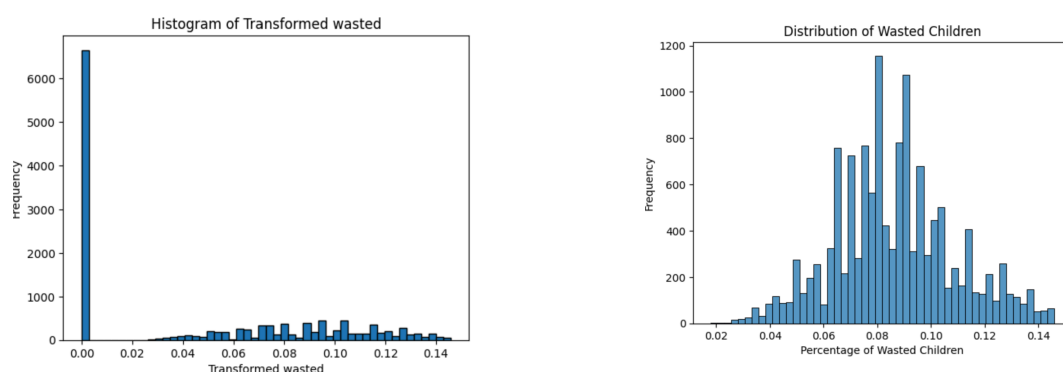
# Use the encoder to transform the 'country' column
df['country'] = encoder.fit_transform(df[['country']])
```

- **Normalizing features:** We applied transformation to features such as alt, pasture, tt00_500k, wasted, healthy, and underweight_bmi to approximate a normal distribution.

and it's end up very well, as we can see in the stunted column for example:



- **Imputation with KNN:** After transformation, we used the KNN imputer to fill in missing values (zeros) in the dataset.




- **Feature expansion:** We included more features in the clustering process, leading to better separation of clusters and more informative features for classification.

```
features = ['alt', 'chrps', 'URBAN_RURA', 'country', 'latnum', 'longnum', 'lst', 'numevents_transformed',
            'pasture_transformed', 'tt00_500k_transformed', 'year', 'stunted',
            'wasted_transformed', 'healthy_transformed', 'poorest',
            'underweight_bmi_transformed']

# Standardize the features to have a mean of 0 and standard deviation of 1
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[features])
```

These refinements yielded a more balanced Random Forest classification performance, with an overall accuracy of **71%** instead of 1.00

		precision	recall	f1-score	support
	0	0.66	0.57	0.61	877
	1	0.72	0.78	0.75	1686
	2	0.73	0.72	0.72	1502
	accuracy			0.71	4065
	macro avg	0.70	0.69	0.70	4065
	weighted avg	0.71	0.71	0.71	4065

4. Hyperparameter Tuning

We experimented with hyperparameters such as ***the number of clusters***. However, the most significant gains were obtained by refining feature engineering and transforming key variables for better distribution. Additional hyperparameter tuning can be further explored to optimize performance metrics like precision and recall.

5. Cross-Validation

During refinement, cross-validation was utilized to ensure model robustness. We employed k-fold cross-validation (with k=5) to evaluate model consistency across different data splits. The use of cross-validation also helped avoid overfitting by providing a more comprehensive evaluation of the model's generalization ability.

6. Feature Selection

While initially all features were included in the model, feature selection was revisited as the **accuracy decreased** with more features.

We carefully selected features such as:

`alt, chrps, URBAN_RURA, country, latnum, longnum, lst, numevents_transformed, pasture_transformed, tt00_500k_transformed, stunted, wasted_transformed, healthy_transformed, poorest, and underweight_bmi_transformed`

The rationale behind this selection was based on their correlation with malnutrition and poverty indicators. These features were crucial for capturing spatial, environmental, and demographic factors influencing the target phenomena. Despite adding more features, we managed to maintain a reasonable explained variance and cluster coherence.

7. Conclusion

During the refinement phase, we made significant strides in improving the model's handling of diverse data inputs through feature transformations and normalization.

While we achieved a Random Forest accuracy of 71% and reasonable precision and recall across clusters, cross-validation results indicated that the model still struggles with generalization on unseen data. We will continue experimenting with different models, feature selection, and data handling techniques to address these issues.

Test Submission

1. Overview

The test submission phase was essential for assessing the model's performance on unseen data, simulating real-world conditions. This phase included preparing the test dataset, applying the refined model, and evaluating its generalization capabilities. We compared these results to the training and validation metrics to ensure consistency and avoid overfitting.

2. Data Preparation for Testing

The test dataset, sourced from Kaggle, required preprocessing similar to the training data. The same steps—such as encoding categorical features (like the country column using an OrdinalEncoder), applying transformations (like the transformation for normally distributed values), and imputing missing values using KNN—were repeated for the test dataset to ensure uniformity with the training data. These steps ensured that the model could handle similar data structures during testing.

3. Model Application

The trained Random Forest model was applied to the test dataset to evaluate its performance

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluate the model
print(classification_report(y_test, y_pred))
```

We have also tried other models but it yield to a lower accuracy than random forest

```
✓ [184] from sklearn.neighbors import KNeighborsClassifier
0s      from sklearn.metrics import accuracy_score

      knn = KNeighborsClassifier(n_neighbors=18)
      knn.fit(X_train, y_train)
      y_pred = knn.predict(X_test)
      accuracy = accuracy_score(y_test, y_pred)
      print(f"Accuracy: {accuracy}")
```

➡ Accuracy: 0.6110701107011071

```
✓ [185] from sklearn import svm
7s      svm = svm.SVC()
      svm.fit(X_train, y_train)
      y_pred = svm.predict(X_test)
      accuracy = accuracy_score(y_test, y_pred)
      print(f"Accuracy: {accuracy}")
```

➡ Accuracy: 0.590159901599016

4. Test Metrics

The key metrics used to evaluate the model's performance on the test dataset included precision, recall, and F1-score, alongside overall accuracy. Compared to the training phase (where we initially achieved ~99.9% accuracy), the performance on the test dataset saw a reduction in accuracy (71%), as shown in the confusion matrix and classification report. The cross-validation results, with a mean F1-score of 0.56, also indicated that the model's generalization was suboptimal, suggesting room for further improvement.

5. Model Deployment

While full deployment was not in the scope of this project phase, the model is ready for deployment in a real-world scenario. Integration with external systems, such as an API or a web-based dashboard, would allow organizations (e.g., NGOs or government entities) to use the

model to make predictions based on newly collected data. This could support decision-making related to malnutrition, poverty levels, and resource allocation in vulnerable regions.

6. Code Implementation

Feature Transformation & Encoding:

```
1- from sklearn.preprocessing import OrdinalEncoder
   encoder = OrdinalEncoder()
   df['country'] = encoder.fit_transform(df[['country']])

   # Display the DataFrame after encoding categorical values
   print("After encoding categorical values:")
   print(df.head())

2- ] # Transform the values of alt column to be a normally distributed
    df['numevents_transformed'], lambda_ = stats.yeojohnson(df['numevents'])

[116] # Transform the values of alt column to be a normally distributed
      df['pasture_transformed'], lambda_ = stats.yeojohnson(df['pasture'])

✓ s ▶ # Transform the values of alt column to be a normally distributed
      df['tt00_500k_transformed'], lambda_ = stats.yeojohnson(df['tt00_500k'])
```

KNN Imputation & Scaling:

```
▶ from sklearn.impute import KNNImputer

# Get columns with zeros excluding 'URBAN_RURA'
zero_columns = (df == 0).mean() * 100
zero_columns = zero_columns[zero_columns > 0].index.tolist()
zero_columns.remove('URBAN_RURA')

# Replace zeros with NaN only in the columns from zero_columns
df[zero_columns] = df[zero_columns].replace(0, np.nan)

# Initialize KNNImputer
imputer = KNNImputer(n_neighbors=5)

# Apply the imputer only to the columns in zero_columns
df[zero_columns] = pd.DataFrame(imputer.fit_transform(df[zero_columns]),
                               columns=zero_columns,
                               index=df.index)

# Now df[zero_columns] will have the missing values (previously zeros) imputed using KNN

[139] (df == 0).mean() * 100
```

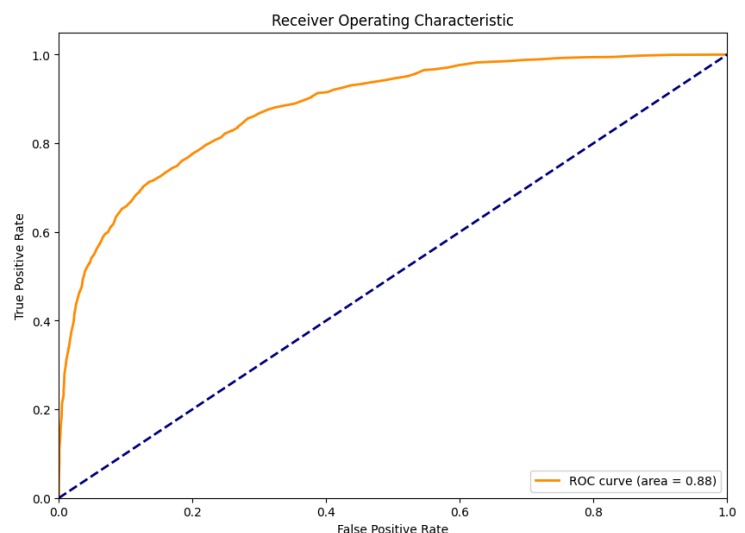

Random Forest Classification & Evaluation:

```
✓ [187] from sklearn.ensemble import RandomForestClassifier  
2s rf = RandomForestClassifier  
rf = RandomForestClassifier()  
rf.fit(X_train, y_train)  
y_pred = rf.predict(X_test)  
accuracy = accuracy_score(y_test, y_pred)  
print(f"Accuracy: {accuracy}")
```

➡ Accuracy: 0.7119311193111931

Comment on ROC Curve:

The curve is rising steeply before leveling off, which indicates that the model has *a decent ability to distinguish between true positives and false positives*. The curve appears to be above the diagonal baseline, which suggests that the model performs better than random guessing.



However, the curve is not as close to the top-left corner as expected for a high-performing model, meaning there is room for improvement. The Area Under the Curve (AUC) could be calculated to provide more quantifiable insight, but visually, this suggests a moderate classifier. Possible refinements could enhance the model's ability to differentiate between the classes more effectively.

Conclusion

Throughout the model refinement and test submission phases, we identified key areas of improvement, particularly in the handling of feature transformations and clustering. Despite initially high training accuracy (~99.9%), cross-validation and test results revealed issues of overfitting. The final accuracy on the test dataset was 71%, and cross-validation produced a mean F1-score of 0.56. These results indicate that further work is required, such as fine-tuning the model's hyperparameters, experimenting with different algorithms, and potentially revisiting feature selection and engineering.