Impacts of climate change on food Security

Group: 17

Group Members Names:

1. Mohamed Fadlelseed
2. Fulchany Arvelin Nanitelamio
3. Bacoro dit Elhadji lansar
4. Madalena Makiesse
5. Zeinab Mohmmed
6. Israa Abdulrahman Mohammed kheir

Model Refinement

7. Overview

The model refinement phase is critical in improving the accuracy and robustness of machine learning models. In this phase, various techniques, such as tuning hyperparameters, selecting relevant features, and testing different algorithms, are employed to enhance model performance. The goal is to reduce prediction errors and increase the generalization capability of the model when applied to unseen data.

8. Model Evaluation

During the initial model evaluation, the performance of seven machine learning models was assessed based on key metrics such as accuracy, Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and R^2 score. The models evaluated included:

- **Decision Tree Regressor**
- **Linear Regression**
- **Random Forest Regressor**
- **Gradient Boosting Regressor**
- **XGBoost Regressor**
- **Bagging Regressor**
- **K-Nearest Neighbors (KNN) Regressor**

Key observations:

- **Bagging Regressor** and **Random Forest Regressor** emerged as the best-performing models with accuracy rates of **98.88%** and **98.97%**, respectively, and consistently low error metrics.

- **KNN Regressor** performed poorly with an accuracy rate of **33.51%**, indicating that it is not suitable for this task.

- **Linear Regression** and **Gradient Boosting Regressor** showed moderate performance but were outperformed by the ensemble methods.

Visualizations during this phase indicated that ensemble models, particularly Bagging and Random Forest, provided the best fit to the data, reducing both variance and bias.

9. Refinement Techniques

Several techniques were employed to refine the models:

- **Hyperparameter tuning** was applied to all models, with a focus on optimizing the depth of trees in decision-tree-based models and adjusting learning rates in gradient boosting models.

- **Ensemble methods** such as Bagging and Random Forest were introduced to improve model stability and accuracy.

- Additional **algorithms**, including XGBoost and Gradient Boost, were tested to compare their performance against simpler models such as Linear Regression and KNN.

10. Hyperparameter Tuning

Hyperparameter tuning was crucial for improving model performance. Techniques included:

- **Decision Tree Regressor**: Tuned for max_depth, leading to a reduction in overfitting.

- **Random Forest Regressor:** The number of trees (n_estimators) and tree depth were optimized to balance between bias and variance.

- **Gradient Boosting Regressor:** Adjusted the learning rate and the number of estimators, which led to a smoother learning curve and reduced overfitting.

11. Cross-Validation

During refinement, the cross-validation strategy was enhanced by increasing the number of folds in K-Fold cross-validation from 5 to 10. This provided a more robust estimate of model performance by ensuring that more data points were included in both the training and validation sets for each fold.

Reasoning:

- A 10-fold cross-validation was chosen to reduce variance and produce a more reliable estimate of how the models would generalize to unseen data.

12. Feature Selection

In the model refinement phase, **feature selection** was critical to improve the model's performance by reducing noise and focusing on the most relevant predictors of crop yield.

. Feature Selection Method: We employed domain knowledge and exploratory data analysis (EDA) to identify the most relevant features. The features used in the model included:

- **Year**: Time-related patterns affecting yield trends.
- **Average Rainfall (mm/year)**: Rainfall is a crucial determinant of crop growth.
- **Pesticides (tonnes)**: The amount of pesticides influences crop health and yield.
- **Average Temperature (°C)**: Temperature affects crop growth cycles.
- **Country**: Categorical feature representing the region where crops are grown.
- **Item (Crop Type)**: Categorical feature representing the specific crop being predicted.

We applied **one-hot encoding** to the categorical features (Country and Item), which expanded these into multiple binary columns. No formal automated feature selection techniques like Recursive Feature Elimination (RFE) or feature importance ranking were used in this phase. Instead, we relied on domain expertise to select features that are known to influence crop yields.

. Impact on Model Performance

By selecting only relevant features, the model:

- **Reduced overfitting**: Avoiding inclusion of irrelevant or redundant features helped the model generalize better on unseen data.
- **Improved interpretability**: Focusing on key predictors allowed for clearer insights into how each factor affects the yield.
- **Increased efficiency**: With fewer features, the model trained faster and performed predictions more efficiently, without significant loss in accuracy.

In summary, careful selection of meaningful features helped balance model complexity and prediction accuracy, leading to improved model performance without the need for extensive automated feature selection techniques.

Test Submission

1. Overview

The test submission phase involved applying the refined models to a test dataset that had not been used during the training process. The goal was to evaluate the model's performance on

unseen data, simulate real-world deployment, and verify its generalization capability.

2. Data Preparation for Testing

The test dataset was carefully prepared by:

- **Cleaning** any missing values and ensuring the dataset followed the same structure as the training data.
- **Scaling** the features using standardization techniques, as some models (like KNN) are sensitive to feature scaling.

3. Model Application

The trained machine learning models were applied to the test dataset to evaluate their predictive performance on unseen data. After training, the models were used to make predictions on the test dataset, which helps in assessing how well the models generalize to new data.

The following code snippet demonstrates how the models were applied to the test dataset :

```python
# Loops through the list of machine learning models above
for name, model in models :
    # Train Model
    model.fit(X_train, y_train)
    # Make Predictions
    y_pred = model.predict(X_test)

    accuracy = model.score(X_test, y_test)
    MSE = mean_squared_error(y_test, y_pred)
    R2_score = r2_score(y_test, y_pred)
    # Add all metrics of model to a list
    results.append((name, accuracy, MSE, R2_score))

    acc = (model.score(X_train , y_train) * 100)
    print(f'Accuracy of {name} Model Train is {acc:.2f}')
    acc = (model.score(X_test , y_test) * 100)
    print(f'Accuracy of the {name} Model Test is {acc:.2f}')

    data = {'y_test' : [y_test],
            'y_pred' : [y_pred]}
    data_df = pd.DataFrame(data)

    fig = px.scatter(data_df, x = y_test, y = y_pred,
                     labels = {'x' : 'Actual Values', 'y' : 'Predicted Values'},
                     trendline = 'ols', trendline_color_override = 'red',
                     template = 'plotly_dark')
    fig.show()
```

4. Test Metrics

The following metrics were used to evaluate model performance on the test dataset:

- **Accuracy**
- **Mean Squared Error (MSE)**
- **Mean Absolute Error (MAE)**
- **Mean Absolute Percentage Error (MAPE)**
- **R^2 Score**

The test results were consistent with the cross-validation scores from the training phase, indicating that the models generalized well to unseen data. For example, the **Bagging Regressor** had an accuracy of **98.88%** on the test set, closely matching its cross-validation

performance.

5. Model Deployment

In a real-world scenario, deploying the model would involve integrating it into a crop management system where it could be used to predict yields based on real-time input data. This could be achieved by:

- Deploying the model as an API endpoint for crop yield prediction.
- Integrating it with a larger system for managing farm operations.

6. Code Implementation

Key code snippets from the refinement and testing phases:

. Hyperparameter tuning for Random Forest Regressor:

```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [50, 100, 200], 'max_features': [0.5, 1.0]}
bagging_grid = GridSearchCV(BaggingRegressor(random_state=1), param_grid, cv=5)
bagging_grid.fit(X_train, y_train)
print(f"Best Parameters: {bagging_grid.best_params_}")
```

Cross-validation and testing:

```
kf = KFold(n_splits = 10, shuffle = True)
scores = cross_val_score(model, X, y, cv = kf)

# Print out the CV Scores for each fold
for fold, score in enumerate(scores) :
    print(f'Fold {fold + 1}: {score}')
    temp_df = pd.DataFrame({'Name' : name, 'Fold' : [fold + 1], 'Score' : [score]})
    dfs = [fold_df, temp_df]
    fold_df = pd.concat(dfs, ignore_index = True)

# Print out the Mean CV scores for each model
mean_score = np.mean(scores)
print(f'Mean Score: {mean_score}')
print('=' * 30)
```

Conclusion

The model refinement and test submission phases resulted in a significant improvement in the model's performance. The Bagging Regressor and Random Forest Regressor were the top-performing models, achieving accuracy rates close to **99%**. Hyperparameter tuning and cross-validation contributed to minimizing overfitting and ensuring that the models generalized well to new data. The KNN model, while simple, did not perform well and was excluded from further consideration.

Challenges included finding the right balance between bias and variance, especially with complex models like Gradient Boosting. However, the final models were robust and performed well during testing.

References

- XGBoost Documentation https://xgboost.ai/
- Scikit-learn Documentation https://scikit-learn.org/
- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron