KoilaBetty / PROJECT-PHASE-3

Code Issues Pull requests Actions Projects Security Insights Settings

0 stars 0 forks 1 watching Branches Activity Tags

Private repository

1 Branch 0 Tags Go to file t Go to file + Add file Code

KoilaBetty Readme 8fca1c9 · 22 minutes ago

Images	Images update	39 minutes ago
Readme.md	Readme	22 minutes ago
SubmissionFormat.csv	project	yesterday
TestSetValues.csv	project	yesterday
TrainingSetLabels.csv	project	yesterday
TrainingSetValues.csv	project	yesterday
index.ipynb	final	28 minutes ago
presentation.pdf	presentation pdf	35 minutes ago

README

Phase 3 Project: Tanzania Wells

Project Overview

Stakeholder: The Tanzanian Ministry of Water, NGOs, and organizations involved in water distribution and maintenance.

Business Problem: This project aims to predict the operational status of water pumps across Tanzania, specifically identifying which pumps are functional, which require repairs, and which are non-functional. The prediction is based on various factors such as pump type, installation date, and management practices. The project uses data from Taarifa and the Tanzanian Ministry of Water, with the challenge provided by DrivenData in 2015. By accurately forecasting pump failures, maintenance efforts can be optimized, ensuring that communities in Tanzania maintain reliable access to clean and potable water.

Business Problem

The Tanzanian Ministry of Water faces the ongoing challenge of maintaining the functionality of water pumps throughout the country. Many pumps break down or require maintenance due to factors such as improper management, outdated installations, or environmental conditions. Accurately predicting whether a pump will remain functional, require repairs, or fail entirely is essential for effective maintenance planning and resource allocation.

Predicting pump functionality can help:

- **Improve maintenance operations** by prioritizing pumps that need immediate attention, ensuring that the most critical issues are addressed first.
- **Ensure continuous access to clean water** by minimizing pump failures and preventing disruptions in water supply.
- **Allocate resources effectively** for repairs and replacements, reducing downtime and minimizing the impact on communities dependent on these water sources.

The goal of this project is to predict which pumps are functional, which need some repairs and which dont work at all.

The challenge from DrivenData. (2015). Pump it Up: Data Mining the Water Table. Retrieved [Month Day Year] from https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table with data from Taarifa and the Tanzanian Ministry of Water. The goal of this project is to predict one of these three classes based on a number of variables about what kind of pump is operating, when it was installed, and how it is managed. A smart understanding of which waterpoints will fail can improve maintenance operations and ensure that clean, potable water is available to communities across Tanzania.

1. Project Setup

- **Import Necessary Libraries**: The first step is to import the required libraries, such as pandas and numpy for data manipulation, sklearn for machine learning, and matplotlib and seaborn for data visualization.

- **Load the Dataset**: The dataset is then loaded into the environment, typically from a CSV or other file formats, to begin the analysis.

- **Set Random Seed for Reproducibility**: A random seed is set to ensure that any random processes (e.g., train-test splits, model initialization) produce the same results each time the code is run, ensuring reproducibility.

2. Data Exploration and Preprocessing

In this step, we focus on preparing the data for building machine learning models:

- **Examine the Dataset**: First, we inspect the dataset by viewing the first few rows (using `head`), getting a summary of its structure (`info`), and understanding basic statistics of the numerical features (`describe`).

- **Check for Missing Values and Handle Them**: We identify missing data within the dataset and address it either by imputing values (e.g., filling with mean or median) or by deleting rows or columns that contain too many missing values.

- **Explore Data Distributions and Correlations**: Next, we analyze the distribution of each feature and assess relationships between features, including correlation analysis to identify potential multicollinearity.

- **Perform Feature Engineering if Necessary**: If needed, we create new features or modify existing ones to improve the model's predictive power. This may involve encoding categorical variables, scaling numerical features, or creating interaction terms.

- **Split the Data into Features (X) and Target Variable(s) (y)**: Finally, we separate the dataset into the features (X) and the target variable(s) (y) to prepare for model training.

Dropped Columns and Rationale:

- **date_recorded**: This is redundant because the construction year is already available, which suffices for predicting well functionality.
- **funder**: Some rows have identical values to the installer column, and many others are variations of the same entry. The installer has a greater impact on well functionality.
- **wpt_name**: Too many unique values to be useful.
- **subvillage**: This can be represented by the region.
- **lga**: Can be represented by the region.
- **ward**: Can be represented by the region.
- **recorded_by**: This column has the same value for all rows, making it irrelevant.
- **scheme_name**: Contains too many unique values and many null entries.
- **extraction_type**: Very similar to the 'extraction_type_class' column.
- **extraction_type_group**: Similar to 'extraction_type_class'.
- **management**: The 'management_group' column already covers the categories of management, so 'management' will be dropped.
- **payment**: It duplicates the 'payment_type' column, so it will be removed.
- **quality_group**: Highly correlated with the 'water_quality' column. Since 'water_quality' offers unique rows, it will be retained while dropping this column.
- **quantity**: The 'quantity_group' column represents categories of quantity, so 'quantity' will be dropped.
- **source**: This can be represented by the 'source_class' column.
- **source_type**: This can also be represented by the 'source_class' column.
- **waterpoint_type**: This is similar to 'waterpoint_type_group', so it will be removed.

Numerical Columns:

- **num_private**: The meaning of this column is unclear.
- **region_code**: It is a duplicate of the 'region' column.
- **district_code**: This can be captured by the 'region' column.

Observations:

- **Funder** and **installer** are similar, but the 'funder' column has fewer null values than the 'installer' column.
- **Waterpoint_type** and **waterpoint** are highly similar; 'waterpoint_type' will be dropped.

3. Logistic Regression

- **Prepare the Data**: The first step involves ensuring that the dataset is ready for modeling. This includes verifying that the target variable is binary, meaning it has only two possible outcomes, which is essential for logistic regression.

- **Create and Train the Model**: Next, a logistic regression model is constructed and trained using the prepared data. The model learns the relationship between the features (independent variables) and the binary target variable.

- **Make Predictions on the Test Set**: After training, the model is used to make predictions on a separate test dataset, which was not used during the training phase. This allows us to assess the model's performance on unseen data.

- **Evaluate the Model**: The model's performance is evaluated using key metrics such as accuracy (the proportion of correct predictions), precision (the proportion of true positive predictions relative to all positive predictions), recall (the proportion of true positives relative to all actual positives), and F1-score (the harmonic mean of precision and recall).

- **Plot the ROC Curve and Calculate AUC**: The Receiver Operating Characteristic (ROC) curve is plotted to visualize the trade-off between true positive rate and false positive rate at various threshold settings. The Area Under the Curve (AUC) is then calculated to quantify the model's ability to distinguish between the classes.

- **Analyze Coefficients and Their Significance**: Finally, the coefficients of the logistic regression model are analyzed to understand the influence of each feature on the target variable. Statistical significance tests are performed to identify which features have a meaningful impact on the outcome.

1. Overall Accuracy:

The logistic regression model achieves an overall accuracy of approximately 70.8%, which reflects a decent performance but also indicates there is significant room for enhancement.

2. Performance by Class:

- **Functional Pumps (Majority Class)**:

  - **Precision**: 0.70 – When the model classifies a pump as "functional," it is correct 70% of the time.
  - **Recall**: 0.87 – The model correctly identifies 87% of the pumps that are actually functional.

  - **F1-Score**: 0.77 – This balanced score between precision and recall indicates strong performance in classifying functional pumps.

- **Functional Pumps Needing Repair (Minority Class)**:

  - **Precision**: 0.42 – The model is only 42% accurate when predicting "functional needs repair," suggesting a high rate of misclassification.
  - **Recall**: 0.01 – The model identifies only 1% of the pumps that actually need repairs, which highlights a critical weakness in detecting this category.
  - **F1-Score**: 0.01 – A very low F1-score further emphasizes the poor performance in recognizing pumps requiring repair.

- **Non-functional Pumps**:

  - **Precision**: 0.74 – When the model predicts a pump as "non-functional," it is correct 74% of the time.
  - **Recall**: 0.61 – The model identifies 61% of the actual "non-functional" pumps.
  - **F1-Score**: 0.67 – This score suggests reasonable performance, though there is room for improvement in recall.

3. Confusion Matrix:

- **Functional Pumps**: The model correctly identifies 4901 out of 5620 functional pumps, misclassifying 716 as "non-functional."
- **Functional Pumps Needing Repair**: The model performs poorly here, correctly identifying just 5 out of 743 pumps that need repair. Most of these pumps are misclassified as "functional" (580) or "non-functional" (158).
- **Non-functional Pumps**: Out of 4023 non-functional pumps, the model correctly classifies 2449 but mistakenly labels 1570 as "functional."

Key Insights:

- The model performs reasonably well in predicting "functional" and "non-functional" pumps but struggles significantly with pumps that require repair.
- The **class imbalance** (with the "functional needs repair" category being underrepresented) seems to contribute heavily to the poor performance in identifying this class.
- The very low recall for the "functional needs repair" class suggests that the model fails to identify most pumps in need of repair, misclassifying them as either "functional" or "non-functional."

4. Decision Trees

- **Preprocess the Data**: Prepare the dataset by handling missing values, encoding categorical variables, and splitting the data into features (X) and the target variable (y).

- **Build and Train the Decision Tree Model**: Construct and train a decision tree model using the training data, allowing it to learn patterns from the features to predict the target variable.

- **Generate Predictions Using the Test Set**: After training, use the decision tree model to make predictions on the unseen test set, which helps evaluate its generalization ability.

- **Assess the Model's Performance**: Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1-score to understand how well it classifies the target variable.
- **Visualize the Structure of the Decision Tree**: Visualize the trained decision tree to gain insights into the decision-making process, showing how features are split at each node.
- **Evaluate the Importance of Each Feature**: Analyze the feature importance scores provided by the decision tree model to identify which features contribute most to the model's predictions.

Interpretation of Findings from Decision Tree:

1. Overall Accuracy:

- **Accuracy**: 0.7531 (75.31%) indicates that 75.31% of the water pumps were correctly categorized as functional, needing repair, or non-functional. While this result is satisfactory, there is still potential for improvement, especially in differentiating between the various classes.

2. Precision, Recall, and F1-Score:

- **Functional Pumps**:
    - **Precision**: 0.79 (79%) means that when the model predicted a pump to be functional, it was correct 79% of the time.
    - **Recall**: 0.80 (80%) indicates the model successfully identified 80% of the pumps that were actually functional.
    - **F1-Score**: 0.80 represents a good balance between precision and recall, showing strong performance for this class.
- **Functional Pumps Needing Repair**:
    - **Precision**: 0.37 (37%) shows that when the model predicted a pump needed repair, it was only correct 37% of the time, suggesting many false positives.
    - **Recall**: 0.36 (36%) means the model only correctly identified 36% of the pumps that actually needed repairs.
    - **F1-Score**: 0.36 highlights poor performance, as the model struggles to accurately predict pumps in need of repair, indicating that this class is particularly difficult for the model.
- **Non-Functional Pumps**:
    - **Precision**: 0.77 (77%) indicates that when the model predicted a pump was non-functional, it was correct 77% of the time.
    - **Recall**: 0.76 (76%) shows the model correctly identified 76% of the actual non-functional pumps.
    - **F1-Score**: 0.77 demonstrates solid performance for this class, comparable to that of functional pumps.

3. Macro and Weighted Averages:

- **Macro Average**: These are the unweighted averages across all classes:
    - **Precision**: 0.64 (64%)

  - **Recall**: 0.64 (64%)
  - **F1-Score**: 0.64 (64%)
  - These values are lower, indicating that the model performs worse on the minority classes, particularly for pumps that need repair.

- **Weighted Average**: These metrics are weighted by the number of instances in each class:

  - **Precision**: 0.75 (75%)
  - **Recall**: 0.75 (75%)
  - **F1-Score**: 0.75 (75%)
  - These values are higher, as the model performs better on the majority class ("functional"), which is more prevalent in the dataset.

4. Confusion Matrix:

- **Functional Pumps**: 4,490 pumps were correctly identified as functional, but 331 were misclassified as "functional needs repair" and 799 as "non-functional."
- **Functional Pumps Needing Repair**: Only 270 were correctly classified, with 352 misclassified as functional and 121 as non-functional. This class has the highest misclassification rate, revealing difficulty in distinguishing "functional needs repair" from the other two categories.
- **Non-Functional Pumps**: 3,062 pumps were correctly identified as non-functional, but 824 were incorrectly labeled as functional, and 137 as needing repair.

Key Insights:

1. **Strength**: The model performs well in predicting the majority classes ("functional" and "non-functional" pumps).

2. **Weakness**: The model faces significant challenges in identifying the "functional needs repair" class. This indicates difficulty in distinguishing pumps requiring minor repairs from those that are either functional or non-functional.

5. Model Comparison and Conclusion

- **Compare Performance Metrics Across All Models**: Evaluate and compare the performance of all models using key metrics such as accuracy, precision, recall, F1-score, and AUC to determine which model performs best on the given data.

- **Discuss Strengths and Weaknesses of Each Approach**: Analyze the strengths and limitations of each model, considering factors such as interpretability, computational efficiency, ability to handle complex data, and overfitting.

- **Recommend the Best Model(s) for the Problem at Hand**: Based on the comparison, recommend the most suitable model(s) for solving the problem, taking into account both performance and practicality for deployment.

- **Summarize Key Findings**: Provide a summary of the main insights discovered during the analysis, such as which features were most influential and how different models performed.

- **Discuss Limitations of the Current Approach**: Highlight any potential limitations in the current modeling approach, such as overfitting, insufficient data, or limited feature engineering.
- **Suggest Potential Improvements or Additional Models to Try**: Recommend any potential improvements, such as tuning hyperparameters, adding more data, or testing other advanced models (e.g., ensemble methods, neural networks) to enhance performance.

Interpretation of the Results:

1. Accuracy:

- **Decision Tree**: 0.757 (75.7%)
- **Logistic Regression**: 0.708 (70.8%)

The Decision Tree model has a higher accuracy (75.7%) compared to Logistic Regression (70.8%), indicating that the Decision Tree performs better overall in correctly classifying the pumps.

2. Functional Precision:

- **Decision Tree**: 0.790 (79%)
- **Logistic Regression**: 0.700 (70%)

The Decision Tree model has higher precision when classifying "functional" pumps, meaning that it makes fewer false positive errors (incorrectly predicting a pump as functional when it is not) compared to Logistic Regression.

3. Functional Recall:

- **Decision Tree**: 0.800 (80%)
- **Logistic Regression**: 0.870 (87%)

Logistic Regression has a higher recall for functional pumps, meaning it identifies a higher proportion (87%) of the actual functional pumps. This suggests that Logistic Regression is better at capturing all the true functional pumps, while the Decision Tree misses more of them (80%).

4. Functional F1-Score:

- **Decision Tree**: 0.800
- **Logistic Regression**: 0.770

The Decision Tree has a slightly better F1-Score for functional pumps (0.800 vs. 0.770), indicating a better balance between precision and recall for this class, although the difference is small.

5. Functional Needs Repair Precision:

- **Decision Tree**: 0.370 (37%)
- **Logistic Regression**: 0.420 (42%)

Logistic Regression has a slightly higher precision for identifying "functional needs repair" pumps, meaning it makes fewer incorrect predictions for this class than the Decision Tree.

6. Functional Needs Repair Recall:

- **Decision Tree**: 0.350 (35%)
- **Logistic Regression**: 0.010 (1%)

The Decision Tree performs significantly better in identifying pumps that "need repair," with a recall of 35%, while Logistic Regression only identifies 1% of the actual repair-needed pumps, suggesting a very poor performance in this category for Logistic Regression.

7. Functional Needs Repair F1-Score:

- **Decision Tree**: 0.360
- **Logistic Regression**: 0.010

Similar to recall, the Decision Tree performs much better with an F1-Score of 0.360, while Logistic Regression's F1-Score is extremely low (0.010), indicating a poor balance of precision and recall for this class.

8. Non-functional Precision:

- **Decision Tree**: 0.770 (77%)
- **Logistic Regression**: 0.740 (74%)

The Decision Tree model has slightly higher precision in identifying "non-functional" pumps, indicating fewer false positives when predicting pumps as non-functional.

9. Non-functional Recall:

- **Decision Tree**: 0.770 (77%)
- **Logistic Regression**: 0.610 (61%)

The Decision Tree also has higher recall for non-functional pumps, meaning it is better at identifying actual non-functional pumps compared to Logistic Regression, which only identifies 61% of them.

10. Non-functional F1-Score:

- **Decision Tree**: 0.770
- **Logistic Regression**: 0.670

The Decision Tree model again outperforms Logistic Regression in terms of the F1-Score for non-functional pumps (0.770 vs. 0.670), indicating a better balance of precision and recall for this class.

Summary:

- **Decision Tree** generally performs better than **Logistic Regression** in classifying functional and non-functional pumps, showing better precision, recall, and F1-Score.
- **Logistic Regression** excels in recall for "functional" pumps, but its performance for "functional needs repair" is extremely poor, while **Decision Tree** performs much better in this regard.
- The **Decision Tree** model provides more balanced results across the categories, while **Logistic Regression** struggles particularly with the "functional needs repair" class.

Strengths and Weaknesses of Each Approach

Decision Tree:

Strengths:

The Decision Tree performs well across most metrics, particularly for functional and non-functional pumps. It achieves the highest F1-Score (80%) for functional pumps and demonstrates good recall (77%) for non-functional pumps. Additionally, it is relatively easy to interpret, offering clear decision-making criteria.

Weaknesses:

The model struggles with identifying pumps that need repair, as evidenced by its low precision and recall (37% and 35%, respectively) for this class. This suggests that the model may be overfitting, failing to generalize well to the minority "functional needs repair" category.

Logistic Regression:

Strengths:

Logistic Regression excels at identifying functional pumps, achieving the highest recall (87%) in this category. It also performs decently in terms of precision for the repair class (42%).

Weaknesses:

The model underperforms when it comes to identifying pumps that need repair, with an extremely low recall (1%) and F1-Score (1%) for this class. This significant limitation makes it inadequate for accurately detecting pumps requiring repair, which is crucial for the task at hand.

Recommended Model(s) for the Problem

Based on the performance comparison, the following models are recommended:

Decision Tree

Why it's recommended:
The Decision Tree model delivers a solid performance in identifying functional and non-functional pumps, achieving an accuracy of 75.7%. Additionally, it is highly interpretable, offering clear decision-making criteria, which is crucial in real-world applications where transparency and understandability are important.

Limitation:
While the model performs well overall, it struggles with accurately classifying the "functional needs repair"

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

- ● **Jupyter Notebook** 100.0%