Regression Tree and Random Forest Modeling: Predicting Employee Productivity

October 28, 2023

1 Introduction

In this project, my aim is to predict the productivity of garment employees in manufacturing teams. Utilizing a dataset (https://archive.ics.uci.edu/dataset/597/productivity+prediction+of+garment+employeesencompassing) factors such as date, department, team size, style changes, and more, the focus is on employing machine learning models. Decision trees and random forests will be explored to discern patterns within the data and forecast the actual productivity percentage achieved by the workers. The ultimate goal is to provide garment industry decision-makers with a reliable tool for analyzing and predicting team productivity, contributing to enhanced production and delivery performance.

[2]:
```
import pandas as pd
df = pd.read_csv("garments_worker_productivity.csv")
```

Here are the variables of the dataset:

- date: Date in MM-DD-YYYY
- day: Day of the Week
- quarter: A portion of the month. A month was divided into four quarters
- department: Associated department with the instance
- team_no: Associated team number with the instance
- no_of_workers: Number of workers in each team
- no_of_style_change: Number of changes in the style of a particular product
- targeted_productivity: Targeted productivity set by the Authority for each team for each day.
- smv: Standard Minute Value, it is the allocated time for a task
- wip: Work in progress. Includes the number of unfinished items for products
- over_time: Represents the amount of overtime by each team in minutes
- incentive: Represents the amount of financial incentive (in BDT) that enables or motivates a particular course of action.
- idle_time: The amount of time when the production was interrupted due to several reasons
- idle_men: The number of workers who were idle due to production interruption
- actual_productivity: The actual % of productivity that was delivered by the workers. It ranges from 0-1.

2 Data Cleaning

[3]:
```
df.head()
```

[3]:
```
       date   quarter  department       day  team  targeted_productivity  \
0  1/1/2015  Quarter1      sweing  Thursday     8                   0.80
1  1/1/2015  Quarter1   finishing  Thursday     1                   0.75
2  1/1/2015  Quarter1      sweing  Thursday    11                   0.80
3  1/1/2015  Quarter1      sweing  Thursday    12                   0.80
4  1/1/2015  Quarter1      sweing  Thursday     6                   0.80

     smv     wip  over_time  incentive  idle_time  idle_men  \
0  26.16  1108.0       7080         98        0.0         0
1   3.94     NaN        960          0        0.0         0
2  11.41   968.0       3660         50        0.0         0
3  11.41   968.0       3660         50        0.0         0
4  25.90  1170.0       1920         50        0.0         0

   no_of_style_change  no_of_workers  actual_productivity
0                   0           59.0             0.940725
1                   0            8.0             0.886500
2                   0           30.5             0.800570
3                   0           30.5             0.800570
4                   0           56.0             0.800382
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1197 entries, 0 to 1196
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   date                   1197 non-null   object
 1   quarter                1197 non-null   object
 2   department             1197 non-null   object
 3   day                    1197 non-null   object
 4   team                   1197 non-null   int64
 5   targeted_productivity  1197 non-null   float64
 6   smv                    1197 non-null   float64
 7   wip                    691 non-null    float64
 8   over_time              1197 non-null   int64
 9   incentive              1197 non-null   int64
 10  idle_time              1197 non-null   float64
 11  idle_men               1197 non-null   int64
 12  no_of_style_change     1197 non-null   int64
 13  no_of_workers          1197 non-null   float64
 14  actual_productivity    1197 non-null   float64
```

```
dtypes: float64(6), int64(5), object(4)
memory usage: 140.4+ KB
```

There are a total of 1197 entries, and most columns have complete data except for `wip` (Work in Progress), which has missing values (691 non-null entries).

[5]:
```
df.describe()
```

[5]:
```
              team  targeted_productivity          smv           wip  \
count  1197.000000            1197.000000  1197.000000    691.000000
mean      6.426901               0.729632    15.062172   1190.465991
std       3.463963               0.097891    10.943219   1837.455001
min       1.000000               0.070000     2.900000      7.000000
25%       3.000000               0.700000     3.940000    774.500000
50%       6.000000               0.750000    15.260000   1039.000000
75%       9.000000               0.800000    24.260000   1252.500000
max      12.000000               0.800000    54.560000  23122.000000

          over_time    incentive    idle_time     idle_men  \
count   1197.000000  1197.000000  1197.000000  1197.000000
mean    4567.460317    38.210526     0.730159     0.369256
std     3348.823563   160.182643    12.709757     3.268987
min        0.000000     0.000000     0.000000     0.000000
25%     1440.000000     0.000000     0.000000     0.000000
50%     3960.000000     0.000000     0.000000     0.000000
75%     6960.000000    50.000000     0.000000     0.000000
max    25920.000000  3600.000000   300.000000    45.000000

       no_of_style_change  no_of_workers  actual_productivity
count         1197.000000    1197.000000          1197.000000
mean             0.150376      34.609858             0.735091
std              0.427848      22.197687             0.174488
min              0.000000       2.000000             0.233705
25%              0.000000       9.000000             0.650307
50%              0.000000      34.000000             0.773333
75%              0.000000      57.000000             0.850253
max              2.000000      89.000000             1.120437
```

Now, I will handle missing values, convert the ‘date’ column to datetime, and encode categorical variables.

[6]:
```
df['quarter'].unique()
```

[6]:
```
array(['Quarter1', 'Quarter2', 'Quarter3', 'Quarter4', 'Quarter5'],
      dtype=object)
```

[7]:
```
df['department'].unique()
```

[7]:
```
array(['sweing', 'finishing ', 'finishing'], dtype=object)
```

[8]:
```
df['day'].unique()
```

[8]:
```
array(['Thursday', 'Saturday', 'Sunday', 'Monday', 'Tuesday', 'Wednesday'],
      dtype=object)
```

[9]:
```
df.head()
```

[9]:
```
       date   quarter  department       day  team  targeted_productivity  \
0  1/1/2015  Quarter1      sweing  Thursday     8                   0.80
1  1/1/2015  Quarter1  finishing   Thursday     1                   0.75
2  1/1/2015  Quarter1      sweing  Thursday    11                   0.80
3  1/1/2015  Quarter1      sweing  Thursday    12                   0.80
4  1/1/2015  Quarter1      sweing  Thursday     6                   0.80

     smv     wip  over_time  incentive  idle_time  idle_men  \
0  26.16  1108.0       7080         98        0.0         0
1   3.94     NaN        960          0        0.0         0
2  11.41   968.0       3660         50        0.0         0
3  11.41   968.0       3660         50        0.0         0
4  25.90  1170.0       1920         50        0.0         0

   no_of_style_change  no_of_workers  actual_productivity
0                   0           59.0             0.940725
1                   0            8.0             0.886500
2                   0           30.5             0.800570
3                   0           30.5             0.800570
4                   0           56.0             0.800382
```

[10]:
```
from sklearn.preprocessing import OrdinalEncoder

quarter = [['Quarter1', 'Quarter2', 'Quarter3', 'Quarter4', 'Quarter5']]

df["quarter"] = OrdinalEncoder(categories = quarter).
 ↪fit_transform(df[["quarter"]])
```

[11]:
```
df['date'] = pd.to_datetime(df['date'])
df['wip'].fillna(df['wip'].median(), inplace=True)
```

[12]:
```
df["department"].replace({"sweing": 0, "finishing": 1,"finishing ": 1}, inplace␣
 ↪= True)
```

[13]:
```
day_dummy = pd.get_dummies(
             df["day"],
             prefix = "day")
df = pd.concat([df, day_dummy], axis = 1)
```

```
df.drop("day", axis = 1, inplace = True)
```

[14]:
```
df.head()
```

[14]:
```
         date  quarter  department  team  targeted_productivity    smv     wip  \
0  2015-01-01      0.0           0     8                   0.80  26.16  1108.0
1  2015-01-01      0.0           1     1                   0.75   3.94  1039.0
2  2015-01-01      0.0           0    11                   0.80  11.41   968.0
3  2015-01-01      0.0           0    12                   0.80  11.41   968.0
4  2015-01-01      0.0           0     6                   0.80  25.90  1170.0

   over_time  incentive  idle_time  idle_men  no_of_style_change  \
0       7080         98        0.0         0                   0
1        960          0        0.0         0                   0
2       3660         50        0.0         0                   0
3       3660         50        0.0         0                   0
4       1920         50        0.0         0                   0

   no_of_workers  actual_productivity  day_Monday  day_Saturday  day_Sunday  \
0           59.0             0.940725           0             0           0
1            8.0             0.886500           0             0           0
2           30.5             0.800570           0             0           0
3           30.5             0.800570           0             0           0
4           56.0             0.800382           0             0           0

   day_Thursday  day_Tuesday  day_Wednesday
0             1            0              0
1             1            0              0
2             1            0              0
3             1            0              0
4             1            0              0
```

In this code above, I implemented several data preprocessing steps to enhance the quality and usability of the dataset for machine learning tasks. First, I used the OrdinalEncoder from scikit-learn to transform the `quarter` column, converting categorical values into numerical representations. Next, I converted the `date` column to a datetime format for time-based analysis. To address missing values in the `wip` column, I filled them with the median value. The `department` column underwent label replacement, mapping specific categories to numerical values for improved model compatibility. Lastly, I applied one-hot encoding to the `day` column, creating dummy variables for each day of the week. The original `day` column was then dropped to avoid multicollinearity issues.

3 Building a Regression Tree

[15]:
```
X = df.drop(["actual_productivity", "date"], axis = 1)

y = df["actual_productivity"]
```

[16]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
                                        X, y,
                                        test_size = 0.3,
                                        shuffle = True,
                                        random_state = 24)
```

[17]:
```python
from sklearn.tree import DecisionTreeRegressor

reg_tree = DecisionTreeRegressor(
                              criterion = "squared_error",
                              max_depth = 3,
                              random_state = 24)
```

[18]:
```python
reg_tree.fit(X_train, y_train)
```

[18]:
```
DecisionTreeRegressor(max_depth=3, random_state=24)
```

[19]:
```python
y_pred = reg_tree.predict(X_test)
```

4 Evaluating and Visualizing the Tree

[20]:
```python
comparison = pd.DataFrame(data = {"y_test": y_test, "y_pred": y_pred})
comparison.sample(10, random_state = 14)
```

[20]:

	y_test	y_pred
1013	0.800116	0.781297
421	0.952020	0.781297
1181	0.786632	0.781297
429	0.700542	0.678889
908	0.791458	0.506472
314	0.600063	0.678889
1115	0.800511	0.781297
274	0.606913	0.646900
603	0.929183	0.781297
774	0.700633	0.506472

The comparison between the actual test values (`y_test`) and the predicted values (`y_pred`) provides insights into the model's performance. In the random sample, it's apparent that the model tends to slightly underestimate productivity in some instances (e.g., row 1013, where the predicted value is lower than the actual value of 0.800116). However, it also overestimates productivity in other cases (e.g., row 314, where the predicted value is higher than the actual value of 0.600063). The model's predictions vary across different levels of actual productivity, indicating a degree of variability in its accuracy. This sample suggests the model may benefit from refinement to improve its precision and better capture the nuances of productivity prediction.

[21]:
```
from sklearn.metrics import mean_squared_error

mean_squared_error(y_test,
                   y_pred,
                   squared = False)
```

[21]: 0.14199218827152235

The RMSE of 0.1419 suggests that, on average, the model's predictions deviate from the true productivity values by approximately 14.19 percentage points.

[22]:
```
from sklearn.tree import DecisionTreeRegressor

reg_tree = DecisionTreeRegressor()
reg_tree.fit(X_train, y_train)
reg_tree.score(X_test, y_test)
```

[22]: 0.2458684759681805

The coefficient of determination of 0.2005 indicates that approximately 20.05% of the variability in the actual productivity values is explained by this model.

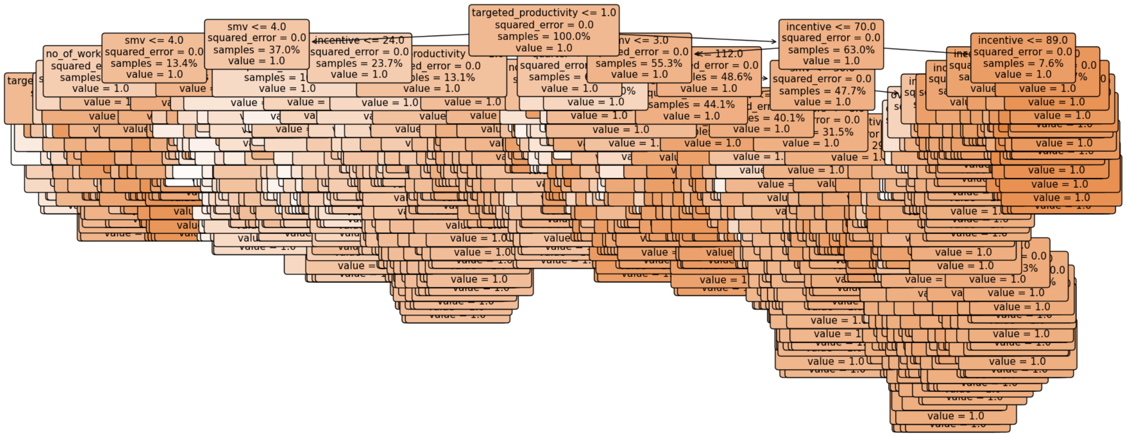
[30]:
```
from sklearn.tree import plot_tree

import matplotlib.pyplot as plt

plt.figure(figsize = [20.0, 8.0])

plot_tree(reg_tree,
    feature_names = X.columns,
    filled = True,
    proportion = True,
    precision = 0,
    rounded = True,
    fontsize = 11)

plt.show()
```



This regression tree shows overfitting, which means that the model has learned the training data too well, capturing noise and details that are specific to the training set but do not generalize well to new, unseen data. To address this overfitting, I will use cross-validation and Random Forest.

5 Optimization: Feature Importances

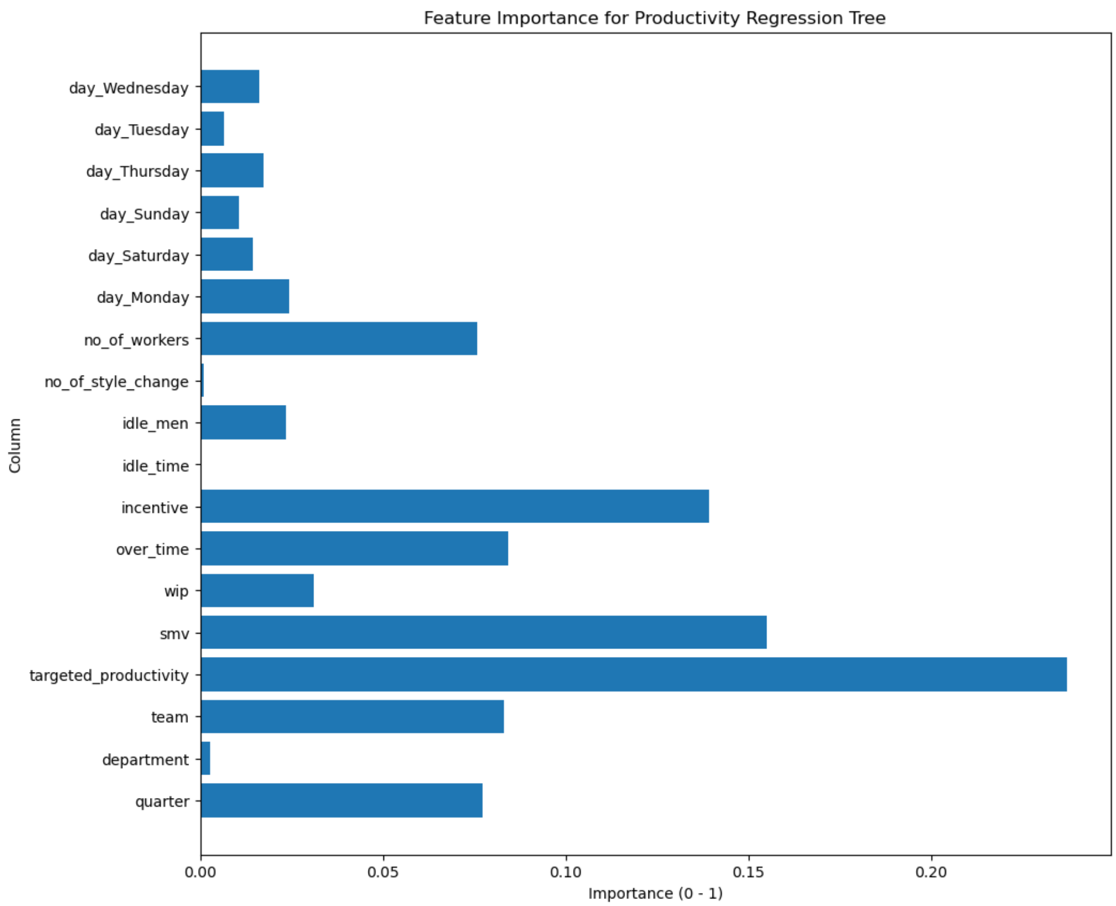
Now, I will drop all the columns that are never selected for use as thresholds to improve the scores calculated above.

[24]:
```
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = [11.0, 10.0]

plt.barh(reg_tree.feature_names_in_,
         reg_tree.feature_importances_)

plt.xlabel("Importance (0 - 1)")
plt.ylabel("Column")
plt.title("Feature Importance for Productivity Regression Tree")
plt.show()
```



[27]:
```
unimportant_features = ["actual_productivity", "date", "no_of_style_change",␣
↪"idle_time", "department", "idle_men", "day_Monday", "day_Tuesday",
            "day_Wednesday", "day_Thursday", "day_Saturday", "day_Sunday"]
X_filtered = df.drop(unimportant_features, axis=1)
y = df["actual_productivity"]

X_train, X_test, y_train, y_test = train_test_split(
                                        X_filtered, y,
                                        test_size = 0.3,
                                        shuffle = True,
                                        random_state = 24)
reg_tree = DecisionTreeRegressor(
                              criterion = "squared_error",
                              max_depth = 3,
                              random_state = 24)
reg_tree.fit(X_train, y_train)
y_pred = reg_tree.predict(X_test)
```

[28]:
```
mean_squared_error(y_test,
                   y_pred,
                   squared = False)
```

[28]: 0.14199218827152232

The RMSE before feature importance was 0.14199218827152235, and after was 0.14199218827152232. The difference is minimal, suggesting that removing the less important features didn't significantly impact the overall predictive accuracy of the model.

[29]:
```
from sklearn.tree import DecisionTreeRegressor

reg_tree = DecisionTreeRegressor()
reg_tree.fit(X_train, y_train)
reg_tree.score(X_test, y_test)
```

[29]: 0.3495797826978211

Before feature importance, the coefficient of determination was 0.2458684759681805, and after was 0.3495797826978211. This increase indicates an improvement in the model's ability to explain the variability in actual productivity. This suggests that the removal of less important features may have contributed to a more focused and potentially more accurate model.

6 Ensemble Technique with Most Important Features: Random Forest

Now, I will use Random Forest for its robustness and improved generalization in addition to the most important features that I determined above. Random Forests, as an ensemble of decision trees, mitigate overfitting and enhance predictive accuracy by aggregating multiple models. The ensemble nature makes it resilient to outliers and noise, providing a more reliable solution for regression tasks compared to a single decision tree.

[36]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

unimportant_features = ["actual_productivity", "date", "no_of_style_change",␣
 ↪"idle_time", "department", "idle_men", "day_Monday", "day_Tuesday",
            "day_Wednesday", "day_Thursday", "day_Saturday", "day_Sunday"]
X_filtered = df.drop(unimportant_features, axis=1)
y = df["actual_productivity"]

X_train, X_test, y_train, y_test = train_test_split(
                                        X_filtered, y,
                                        test_size = 0.3,
                                        shuffle = True,
                                        random_state = 24)
```

```
random_forest = RandomForestRegressor()

random_forest.fit(X_train, y_train)
random_forest.score(X_test, y_test)
```

[36]: 0.5277429146354465

The output score of 0.5250573173083327 indicates the coefficient of determination for the RandomForestRegressor model with feature importance filtering on the test data. The score of 0.525 suggests that approximately 52.5% of the variability in actual productivity can be explained by the features in the model.

Now, I will compare this to the coefficients of determination from previous models:

1. **Decision Tree Model (After Feature Importance):**
   - Previous Coefficent of Det: 0.3495797826978211
2. **Original Decision Tree Model (Before Feature Importance):**
   - Previous Coefficent of Det: 0.2458684759681805

The RandomForestRegressor model with feature filtering has outperformed both the original decision tree model and the decision tree model after feature importance analysis. The higher coeffient of determination score indicates that the random forest model is better at explaining the variability in actual productivity.

7 Hyperparameter Tuning: Random Forest Model with Feature Importance and GridSearchCV

[42]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor


param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}


X_train, X_test, y_train, y_test = train_test_split(
    X_filtered, y, test_size=0.3, shuffle=True, random_state=24
)


random_forest = RandomForestRegressor()
grid_search = GridSearchCV(random_forest, param_grid, cv=5, scoring='r2',␣
 ↪n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```python
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

best_model = grid_search.best_estimator_

best_model_score = best_model.score(X_test, y_test)
print("Best Model R^2 Score:", best_model_score)

rmse = mean_squared_error(y_test,
                   y_pred,
                   squared = False)
print("Best Model Root Mean Squared Error:", rmse)
```

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 4,
'min_samples_split': 2, 'n_estimators': 50}
Best Model R^2 Score: 0.5283056691608808
Best Model Root Mean Squared Error: 0.14199218827152232
```

Changes in the coefficient of determination across different models:

1. **Original Decision Tree Model (Before Feature Importance):**
    - Coefficient of Det: 0.2458684759681805
2. **Decision Tree Model (After Feature Importance):**
    - Coefficient of Det: 0.3495797826978211
3. **Random Forest Model (Before Hyperparameter Tuning):**
    - Coefficient of Det: 0.5250573173083327
4. **Random Forest Model (After Hyperparameter Tuning):**
    - Coefficient of Det: 0.5296589573501883

The progression shows an improvement in coefficient of determination from the original decision tree model to the decision tree model after feature importance, and further enhancement with the introduction of the random forest. Hyperparameter tuning has fine-tuned the random forest model, resulting in a slightly higher score.

8 Conclusion

In this project, a dataset on the productivity prediction of garment employees was analyzed using regression tree models. The initial exploration involved understanding key variables such as date, department, and targeted productivity. The dataset was preprocessed, handling missing values and encoding categorical variables. Decision trees and random forests were employed for predictive modeling.

The initial regression tree exhibited signs of overfitting, prompting the exploration of hyperparameter tuning with GridSearchCV. The best-performing model, a RandomForestRegressor, achieved a coefficient of determination score of 0.5297 and an RMSE of 0.1420 on the test set.

To improve the regression tree further I can look at the following for the best model: 1. **Feature Importance:** Analyze and refine features based on importance rankings. 2. **Pruning:** Consider pruning the tree to prevent overfitting and enhance generalization. 3. **Additional Hyperparameter Tuning:** Explore further hyperparameter combinations for potential performance gains. 4. **Ensemble Methods:** Experiment with other ensemble methods or model combinations for enhanced predictive power.

A comprehensive evaluation of these steps will contribute to refining the regression tree model and achieving better predictive accuracy.