Human Resource Management: Predicting Employee Promotions Using Machine Learning

1. Introduction

1.1. Project overviews

The Employee Promotion Prediction project aims to leverage machine learning techniques to predict employee promotions within an organization. The project uses historical employee data to develop a predictive model, enabling the Human Resources (HR) department to make data-driven decisions regarding employee promotions.

1.2. Objectives

- Develop a machine learning model to predict employee promotions.
- Enhance fairness and objectivity in the promotion process.
- Improve operational efficiency in HR processes.
- Identify and retain high-performing employees.

2. Project Initialization and Planning Phase

2.1. Define Problem Statement

The problem is to predict whether an employee will be promoted based on their historical data, including performance metrics, education level, years of service, and other relevant features. This will help the organization in making informed promotion decisions, reducing bias and increasing transparency.

2.2. Project Proposal (Proposed Solution)

The proposed solution involves collecting relevant employee data, preprocessing it, and using machine learning algorithms, specifically a Random Forest

Classifier, to develop a predictive model. The model will be trained and evaluated to ensure its accuracy and reliability.

2.3. Initial Project Planning

The initial planning includes defining the scope, identifying the necessary data, setting up the project timeline, and allocating resources. The project will follow a structured approach, moving through phases of data collection, preprocessing, model development, and evaluation.

3. Data Collection and Preprocessing Phase

3.1. Data Collection Plan and Raw Data Sources Identified

Data will be collected from internal HR systems, including records of employee performance, demographics, training completion, and historical promotion decisions. The sources include databases, spreadsheets, and internal reports.

3.2. Data Quality Report

A data quality report will be generated to assess the completeness, accuracy, and consistency of the collected data. This will involve checking for missing values, outliers, and inconsistencies.

3.3. Data Exploration and Preprocessing

Data exploration involves analyzing the data to understand its structure and distribution. Preprocessing steps include handling missing values, encoding categorical variables, normalizing numerical features, and addressing class imbalance using techniques like SMOTE.

4. Model Development Phase

4.1. Feature Selection Report

Feature selection involves identifying the most relevant features for predicting promotions. Techniques such as correlation analysis and feature importance from the Random Forest model will be used to select features.

4.2. Model Selection Report

Different machine learning algorithms will be evaluated for their suitability, including Decision Trees, Random Forest, K-Nearest Neighbors (KNN), and XGBoost. The Random Forest Classifier is chosen for its robustness and accuracy.

4.3. Initial Model Training Code, Model Validation and Evaluation Report

The initial model training involves splitting the data into training and testing sets, training the model, and evaluating its performance using metrics like accuracy, precision, recall, and F1-score.

5. Model Optimization and Tuning Phase

5.1. Hyperparameter Tuning Documentation

Hyperparameter tuning involves optimizing the model's parameters to improve performance. RandomizedSearchCV is used to tune the Random Forest model's parameters, such as the number of trees, max depth, and min samples split.

5.2. Performance Metrics Comparison Report

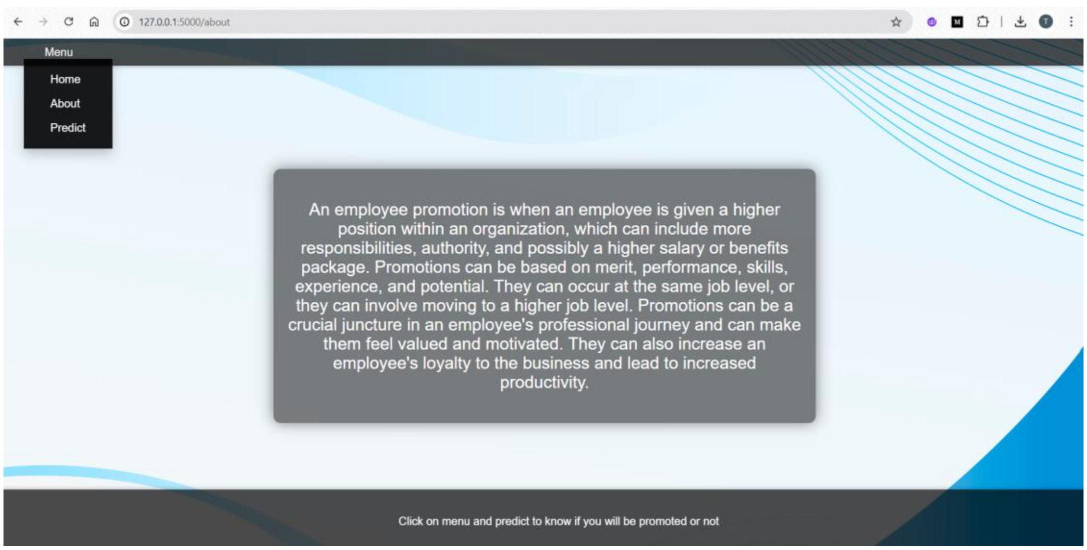
The performance of different models and their tuned versions will be compared using cross-validation scores and evaluation metrics. The comparison helps in selecting the best-performing model.

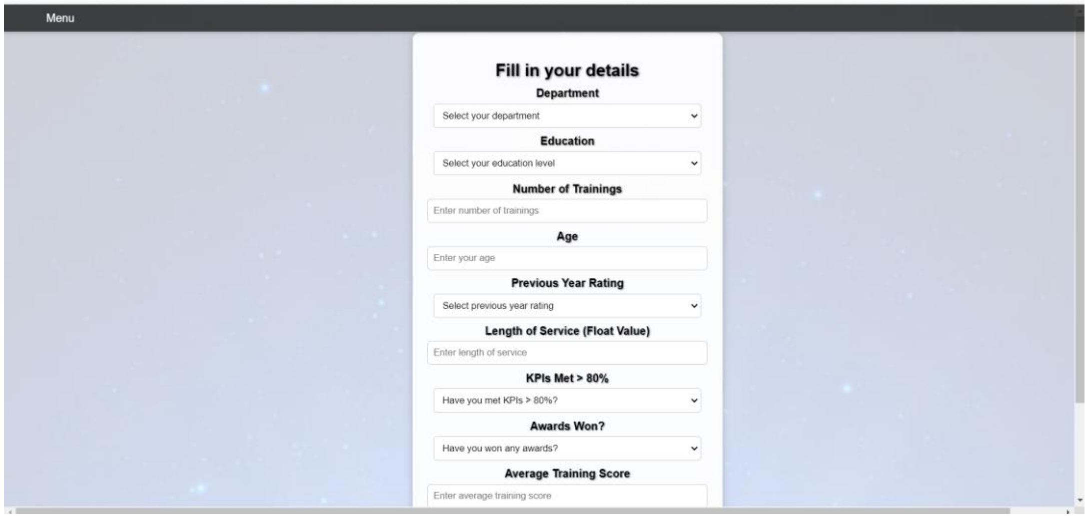
5.3. Final Model Selection Justification

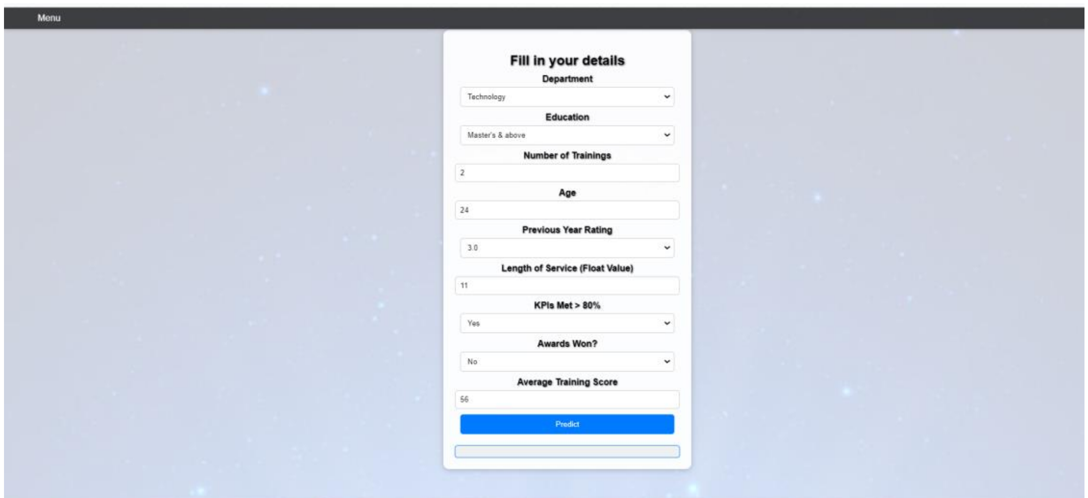
The final model is selected based on its performance metrics, robustness, and interpretability. The Random Forest model, after tuning, is chosen for its superior accuracy and stability.

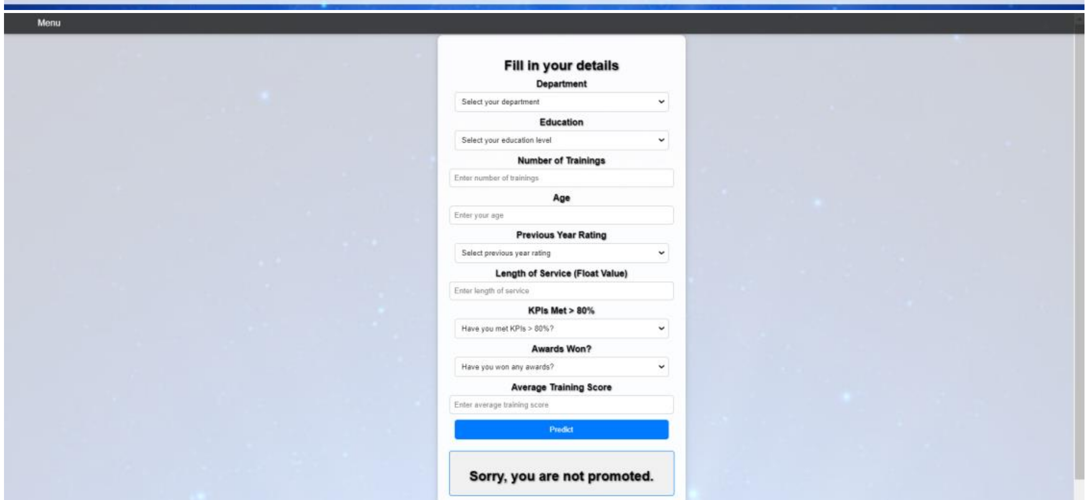
6. Results

6.1. Output Screenshots




Menu
Fill in your details
Department
Select your department
Education
Select your education level
Number of Trainings
Enter number of trainings
Age
Enter your age
Previous Year Rating
Select previous year rating
Length of Service (Float Value)
Enter length of service
KPIs Met > 80%
Have you met KPIs > 80%?
Awards Won?
Have you won any awards?
Average Training Score
Enter average training score


Menu
Fill in your details
Department
Technology
Education
Master's & above
Number of Trainings
2
Age
24
Previous Year Rating
3.0
Length of Service (Float Value)
11
KPIs Met > 80%
Yes
Awards Won?
No
Average Training Score
56
Predict


Menu
Fill in your details
Department
Select your department
Education
Select your education level
Number of Trainings
Enter number of trainings
Age
Enter your age
Previous Year Rating
Select previous year rating
Length of Service (Float Value)
Enter length of service
KPIs Met > 80%
Have you met KPIs > 80%?
Awards Won?
Have you won any awards?
Average Training Score
Enter average training score
Predict
Sorry, you are not promoted.

7. **Advantages & Disadvantages**

- **Advantages**:

  - Data-driven decision-making.
  - Reduced bias in promotions.
  - Improved employee satisfaction and retention.

- **Disadvantages**:

  - Dependency on data quality.
  - Potential resistance to adopting automated decision systems.

8. **Conclusion**

The project successfully developed a predictive model for employee promotions using a Random Forest Classifier. The model improves the promotion process's fairness and efficiency, providing valuable insights to the HR department.

9. **Future Scope**

Future work includes incorporating additional features, refining the model further, and integrating it into HR systems for real-time promotion decisions. Additionally, exploring other machine learning algorithms and techniques can enhance the model's performance.

10. **Appendix**

10.1. **Source Code**

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import warnings
```

```
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
import pickle
from sklearn.metrics import
classification_report,confusion_matrix
plt.style.use('fivethirtyeight')
pd.set_option('display.max_rows',None)
df=pd.read_csv('emp_promotion.csv')
df.shape
df.head()
sns.countplot(x='department', data=df)
plt.title('Department Count')
plt.xlabel('Department')
plt.ylabel('Count')
plt.show()
plt.hist(df['age'], bins=20, edgecolor='black')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.show()
sns.scatterplot(x='avg_training_score',
y='length_of_service', data=df)
plt.xlabel('Average Training Score')
plt.ylabel('length_of_service')
plt.title('Scatter Plot')
plt.show()
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```

```
df.describe()

df.info()

# Drop unwanted features

df = df.drop(['employee_id', 'region', 'gender',
'recruitment_channel'], axis=1)

df.head()

print(df.isnull().sum())

print(df['education'].value_counts())

df['education'] =
df['education'].fillna(df['education'].mode()[0])

print(df['previous_year_rating'].value_counts())

df['previous_year_rating'] =
df['previous_year_rating'].fillna(df['previous_year_rating']
.mode()[0])

print(df.isnull().sum())

negative=df[(df['KPIs_met
>80%']==0)&(df['awards_won?']==0)&(df['previous_year
_rating']==1.0)&(df['is_promoted']==1)&(df['avg_training
_score']<60)]

negative

df.drop(index=[31860,51374],inplace=True)

df.head()

df.shape

sns.boxplot(df['age'])

sns.boxplot(df['avg_training_score'])

sns.boxplot(df['length_of_service'])

# Handle outliers with capping

numerical_cols = ['no_of_trainings', 'age',
'previous_year_rating', 'length_of_service',
'avg_training_score']
```

```
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col] = np.where(df[col] < lower_bound,
lower_bound, df[col])
    df[col] = np.where(df[col] > upper_bound,
upper_bound, df[col])
q1=np.quantile(df['length_of_service'],0.25)
q3=np.quantile(df['length_of_service'],0.75)
IQR= q3-q1
upper_bound=(1.5*IQR)+q3
lower_bound=(1.5*IQR)-q1
print("Skewed
data:",len(df[df['length_of_service']>upper_bound]))
pd.crosstab([df['length_of_service']>upper_bound],df['is_
promoted'])
df['length_of_service']=[upper_bound if x>upper_bound
else x for x in df['length_of_service']]
pd.crosstab([df['length_of_service']<lower_bound],df['is_
promoted'])
df['length_of_service']=[upper_bound if x<lower_bound
else x for x in df['length_of_service']]
sns.boxplot(df['length_of_service'])
le = LabelEncoder()
df['department'] = le.fit_transform(df['department'])
df['education'] = le.fit_transform(df['education'])
df.head()
```

```
X=df.drop('is_promoted',axis=1)
y=df['is_promoted']
print(X.shape)
print(y.shape)
count_0 = np.count_nonzero(y == 0)
count_1 = np.count_nonzero(y== 1)

print(f"Number of 0s before sampling: {count_0}")
print(f"Number of 1s before sampling: {count_1}")
from imblearn.over_sampling import SMOTE
sm=SMOTE()
X_new,y_new=sm.fit_resample(X,y)
count_0 = np.count_nonzero(y_new == 0)
count_1 = np.count_nonzero(y_new== 1)

print(f"Number of 0s after sampling: {count_0}")
print(f"Number of 1s after sampling: {count_1}")
# visualize the class distribution
plt.figure(figsize=(6,4))
sns.countplot(x=y)
plt.title('Class Distribution before Undersampling')
plt.show()
plt.figure(figsize=(6,4))
sns.countplot(x=y_new)
plt.title('Class Distribution after Undersampling')
plt.show()
X_train,X_test,y_train,y_test=train_test_split(X_new,y_new,test_size=0.3,random_state=42)
```

```
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
#Importing the models from sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score

# Initialize models
dt_model = DecisionTreeClassifier(random_state=42)
rf_model = RandomForestClassifier(random_state=42)
knn_model = KNeighborsClassifier()
xgb_model = XGBClassifier(random_state=42)

# Train the models
dt_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)
knn_model.fit(X_train, y_train)
xgb_model.fit(X_train, y_train)

from sklearn.metrics import confusion_matrix,
classification_report

# Evaluate each model
models = {
```

```
    'Decision Tree': dt_model,
    'Random Forest': rf_model,
    'KNN': knn_model,
    'XGBoost': xgb_model
}

for model_name, model in models.items():
    y_pred = model.predict(X_test)
    print(f"Evaluation for {model_name}:\n")
    print(confusion_matrix(y_test, y_pred))
    print("\n")
    print(classification_report(y_test, y_pred))
    print("="*80)

# Define a function to compare models
def compareModel(models, X, y):
    results = {}
    for model_name, model in models.items():
        scores = cross_val_score(model, X, y, cv=5)
        results[model_name] = scores.mean()
    return results

# Comparing models
model_scores = compareModel(models, X_train, y_train)
print("Model Comparison:")
for model_name, score in model_scores.items():
    print(f"{model_name}: Mean Cross-Validation
Accuracy = {score}")
```

```
from sklearn.model_selection import
RandomizedSearchCV

import time


# Decision Tree reduced parameter grid
dt_params = {
    'criterion': ['gini', 'entropy'],  # Criterion options
    'max_depth': [None, 10, 20, 30],  # Depth options
    'min_samples_split': [2, 5, 10],  # Min samples to split
options
    'min_samples_leaf': [1, 2, 4]  # Min samples per leaf
options
}


rf_params = {
    'n_estimators': [100, 200],  # Reduced options
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt', 'log2']
}


knn_params = {
    'n_neighbors': [3, 5, 7],  # Reduced range
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
```

```
# XGBoost reduced parameter grid
xgb_params = {
    'n_estimators': [100, 200],  # Reduced options
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 0.1],
    'min_child_weight': [1, 3]
}
# Perform Randomized Search for each model
start_time = time.time()

# Decision Tree
dt_grid = RandomizedSearchCV(dt_model, dt_params,
cv=3, scoring='accuracy', n_iter=20, n_jobs=-1,
random_state=42)
dt_grid.fit(X_train, y_train)
dt_best_model = dt_grid.best_estimator_

# Random Forest
rf_grid = RandomizedSearchCV(rf_model, rf_params,
cv=3, scoring='accuracy', n_iter=20, n_jobs=-1,
random_state=42)
rf_grid.fit(X_train, y_train)
rf_best_model = rf_grid.best_estimator_

# KNN
```

```
knn_grid = RandomizedSearchCV(knn_model,
knn_params, cv=3, scoring='accuracy', n_iter=20,
n_jobs=-1, random_state=42)
knn_grid.fit(X_train, y_train)
knn_best_model = knn_grid.best_estimator_

# XGBoost
xgb_grid = RandomizedSearchCV(xgb_model,
xgb_params, cv=3, scoring='accuracy', n_iter=20,
n_jobs=-1, random_state=42)
xgb_grid.fit(X_train, y_train)
xgb_best_model = xgb_grid.best_estimator_

end_time = time.time()
print(f"Total tuning time: {(end_time - start_time)/60:.2f}
minutes")
# Evaluate each tuned model
tuned_models = {
    'Decision Tree': dt_best_model,
    'Random Forest': rf_best_model,
    'KNN': knn_best_model,
    'XGBoost': xgb_best_model
}

for model_name, model in tuned_models.items():
    y_pred = model.predict(X_test)
    print(f"Evaluation for {model_name}:")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

```
    print("="*80)
# Compare models
def compareModel(models, X, y):
    results = {}
    for model_name, model in models.items():
        scores = cross_val_score(model, X, y, cv=5)
        results[model_name] = scores.mean()
    return results


# Comparing tuned models
model_scores = compareModel(tuned_models, X_train,
y_train)
print("Model Comparison:")
for model_name, score in model_scores.items():
    print(f"{model_name}: Mean Cross-Validation
Accuracy = {score:.4f}")
model_names = list(model_scores.keys())
performance_scores = list(model_scores.values())


plt.figure(figsize=(10, 5))
sns.barplot(x=model_names, y=performance_scores)
plt.xlabel('Model')
plt.ylabel('Mean Cross-Validation Accuracy')
plt.title('Model Comparison')
plt.show()
print(f"The best model is {best_model_name} with a
mean cross-validation accuracy of
{model_scores[best_model_name]:.4f}")
```

```
df.head(1)

best_model_name = max(model_scores,
key=model_scores.get)
best_model = tuned_models[best_model_name]
with open('hr.pkl','wb') as f:
    pickle.dump(best_model,f)
best_model=pickle.load(open('hr.pkl','rb'))
best_model.predict([[7,2,1.0,35.0,5.0,8.0,1,0,49.0]])[0]
```

10.2. GitHub & Project Demo Link

GitHub Repository: https://github.com/Tvarasree/Employee-Promotion-Prediction

Project Demo: https://drive.google.com/file/d/1-vcwtX9rHk7fk_-bbvWMuvbcGqaFGBAK/view?usp=sharing