Activity_ Course 7 Salifort Motors project lab

November 10, 2023

1 Capstone project: Providing data-driven suggestions for HR

1.1 Description and deliverables

This capstone project is an opportunity for you to analyze a dataset and build predictive models that can provide insights to the Human Resources (HR) department of a large consulting firm.

Upon completion, you will have two artifacts that you would be able to present to future employers. One is a brief one-page summary of this project that you would present to external stakeholders as the data professional in Salifort Motors. The other is a complete code notebook provided here. Please consider your prior course work and select one way to achieve this given project question. Either use a regression model or machine learning model to predict whether or not an employee will leave the company. The exemplar following this actiivty shows both approaches, but you only need to do one.

In your deliverables, you will include the model evaluation (and interpretation if applicable), a data visualization(s) of your choice that is directly related to the question you ask, ethical considerations, and the resources you used to troubleshoot and find answers or solutions.

2 PACE stages

2.1 Pace: Plan

2.1.1 Understand the business scenario and problem

The HR department at Salifort Motors wants to take some initiatives to improve employee satisfaction levels at the company. They collected data from employees, but now they don't know what to do with it. They refer to you as a data analytics professional and ask you to provide data-driven suggestions based on your understanding of the data. They have the following question: what's likely to make the employee leave the company?

Your goals in this project are to analyze the data collected by the HR department and to build a model that predicts whether or not an employee will leave the company.

If you can predict employees likely to quit, it might be possible to identify factors that contribute to their leaving. Because it is time-consuming and expensive to find, interview, and hire new employees, increasing employee retention will be beneficial to the company.

2.1.2 Familiarize yourself with the HR dataset

The dataset that you'll be using in this lab contains 15,000 rows and 10 columns for the variables listed below.

Note: you don't need to download any data to complete this lab. For more information about the data, refer to its source on Kaggle.

Variable	Description
satisfaction_level	Employee-reported job satisfaction level [0–1]
last_evaluation	Score of employee's last performance review [0–1]
number_project	Number of projects employee contributes to
average_monthly_hours	Average number of hours employee worked per month
time_spend_company	How long the employee has been with the company (years)
Work_accident	Whether or not the employee experienced an accident while at work
left	Whether or not the employee left the company
promotion_last_5years	Whether or not the employee was promoted in the last 5 years
Department	The employee's department
salary	The employee's salary (U.S. dollars)

Reflect on these questions as you complete the plan stage.

- Who are your stakeholders for this project?
- What are you trying to solve or accomplish?
- What are your initial observations when you explore the data?
- What resources do you find yourself using as you complete this stage? (Make sure to include the links.)
- Do you have any ethical considerations in this stage?

Who are your stakeholders for this project? HR department at Salifort Motors.

What are you trying to solve or accomplish? They are trying to find out "what's likely to make the employee leave the company?"

What are your initial observations when you explore the data? There are some columns with categorical and numeric variable.

What resources do you find yourself using as you complete this stage? (Make sure to include the links.) Do you have any ethical considerations in this stage?

2.2 Step 1. Imports

- Import packages

- Load dataset

2.2.1 Import packages

[97]:
```
# Import packages
### YOUR CODE HERE ###

# For data manipulation
import numpy as np
import pandas as pd

# For data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# For displaying all of the columns in dataframes
pd.set_option('display.max_columns', None)

# For data modeling
from xgboost import XGBClassifier
from xgboost import XGBRegressor
from xgboost import plot_importance

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# For metrics and helpful functions
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,\
f1_score, confusion_matrix, ConfusionMatrixDisplay, classification_report
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.tree import plot_tree

# For saving models
import pickle
```

2.2.2 Load dataset

`Pandas` is used to read a dataset called **`HR_capstone_dataset.csv`.** As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

```
[71]: # RUN THIS CELL TO IMPORT YOUR DATA.

      # Load dataset into a dataframe
      ### YOUR CODE HERE ###
      df = pd.read_csv("HR_capstone_dataset.csv")


      # Display first few rows of the dataframe
      #df0.head()
```

```
[ ]: # Display first few rows of the dataframe
     df.head(30)
```

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

2.3 Step 2. Data Exploration (Initial EDA and data cleaning)

- Understand your variables
- Clean your dataset (missing data, redundant data, outliers)

2.3.1 Gather basic information about the data

```
[ ]: # Gather basic information about the data
     print(df.size)
     print(df.shape)
     print(df.info())
```

The dataset has 10 ccolumns and 14999 rows. We could see there is no missing values in the dataset.

```
[ ]: df.isnull().sum()
```

2.3.2 Gather descriptive statistics about the data

```
[ ]: # Gather descriptive statistics about the data
     df.describe()
```

2.3.3 Rename columns

As a data cleaning step, rename the columns as needed. Standardize the column names so that they are all in `snake_case`, correct any column names that are misspelled, and make column names more concise as needed.

[ ]:
```
# Display all column names
print(df.columns)
```

[ ]:
```
# Rename columns as needed
df = df.rename(columns={'Department':'department','Work_accident':
 ↪'work_accident',
                        'average_montly_hours':
 ↪'average_monthly_hours','time_spend_company':'tenure'})


# Display all column names after the update
print(df.columns)
```

2.3.4 Check missing values

Check for any missing values in the data.

[ ]:
```
# Check for missing values
df.isna().sum()
```

2.3.5 Check duplicates

Check for any duplicate entries in the data.

[ ]:
```
# Check for duplicates
df.duplicated().sum()
```

3008 rows have duplicate values. It is almost 20% of the data. But there no name or employee id to diffrentiate the duplicates. There is a possibility that two or more employees self-reported the exact same response for every column.

[ ]:
```
# Inspect some rows containing duplicates as needed
df[df.duplicated()].head()
```

[ ]:
```
# Drop duplicates and save resulting dataframe in a new variable as needed
df = df.drop_duplicates()
```

[ ]:
```
# Display first few rows of new dataframe as needed
df.head()
```

[ ]:
```
print(df.shape)
```

2.3.6 Check outliers

Check for outliers in the data.

[ ]:
```python
# Create a boxplot to visualize distribution of `tenure` and detect any outliers
plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers for tenure', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=df['tenure'])
plt.show()
```

[ ]:
```python
# Determine the number of rows containing outliers
percentile25 = df['tenure'].quantile(0.25)

percentile50 = df['tenure'].quantile(0.5)

percentile75 = df['tenure'].quantile(0.75)

#Calculate IQR

IQR = percentile75 - percentile25

print(IQR)

#Calcule lower and upper limit
upper_limit = percentile75 + 1.5 * IQR
lower_limit = percentile25 - 1.5 * IQR
print("Lower limit:", lower_limit)
print("Upper limit:", upper_limit)

# Identify subset of data containing outliers in `tenure`
outliers = df[(df['tenure'] > upper_limit) | (df['tenure'] < lower_limit)]

# Count how many rows in the data contain outliers in `tenure`
print("Number of rows in the data containing outliers in `tenure`:",␣
 ↪len(outliers))
```

Certain types of models are more sensitive to outliers than others. When you get to the stage of building your model, consider whether to remove outliers, based on the type of model you decide to use.

3 pAce: Analyze Stage

- Perform EDA (analyze relationships between variables)

Reflect on these questions as you complete the analyze stage.

- What did you observe about the relationships between variables?
- What do you observe about the distributions in the data?

- What transformations did you make with your data? Why did you chose to make those decisions?
- What are some purposes of EDA before constructing a predictive model?
- What resources do you find yourself using as you complete this stage? (Make sure to include the links.)
- Do you have any ethical considerations in this stage?

[Double-click to enter your responses here.]

3.1 Step 2. Data Exploration (Continue EDA)

Begin by understanding how many employees left and what percentage of all employees this figure represents.

```
[ ]: # Get numbers of people who left vs. stayed
     print(df['left'].value_counts())
```

```
[ ]: # Get percentages of people who left vs. stayed
     print(df['left'].value_counts(normalize = True))
```

83% of employees stayed and 16% of employees left.

3.1.1 Data visualizations

Now, examine variables that you're interested in, and create plots to visualize relationships between variables in the data.

```
[ ]: # Set figure and axes
     fig, ax = plt.subplots(1, 2, figsize = (22,8))

     # Create boxplot showing `average_monthly_hours` distributions for␣
      ↪`number_project`, comparing employees who stayed versus those who left
     sns.boxplot(data=df, x='average_monthly_hours', y='number_project', hue='left',␣
      ↪orient="h", ax=ax[0])
     ax[0].invert_yaxis()
     ax[0].set_title('Monthly hours by number of projects', fontsize='14')

     # Create histogram showing distribution of `number_project`, comparing␣
      ↪employees who stayed versus those who left
     tenure_stay = df[df['left']==0]['number_project']
     tenure_left = df[df['left']==1]['number_project']
     sns.histplot(data=df, x='number_project', hue='left', multiple='dodge',␣
      ↪shrink=2, ax=ax[1])
     ax[1].set_title('Number of projects histogram', fontsize='14')

     # Display the plots
     plt.show()
```

Everybody with seven projects left the company, and they also worked the most hours relative to the other employees.

People who left the company worked more hours than their colleagues on the same amount of projects. However, we can observe that individuals who left worked less than those who stayed. It's probable they were dismissed for that reason.

If you assume a work week of 40 hours and two weeks of vacation per year, then the average number of working hours per month of employees working Monday–Friday = 50 weeks * 40 hours per week / 12 months = 166.67 hours per month. This means that, aside from the employees who worked on two projects, every group—even those who didn't leave the company—worked considerably more hours than this. It seems that employees here are overworked.

```
[ ]: # Get value counts of stayed/left for employees with 7 projects
     df[df['number_project']==7]['left'].value_counts()
```

```
[ ]: # Plot `department` split into groups by `left`
     plt.figure(figsize = (4, 4))
     sns.histplot(data = df,
                  x = "department",
                  hue = "left",
                  discrete = True,
                  multiple = "dodge",
                  shrink = 0.8)
     plt.legend(labels = ["Left", "Stayed"])
     plt.title("Histogram of Employees Per Department")
     plt.xlabel("Department")
     plt.xticks(rotation = 45, horizontalalignment = "right")
     plt.show()
```

```
[ ]: # Create a plot as needed  by department
     plt.figure(figsize=(12, 6))
     sns.boxplot(data=df, x='department', y='satisfaction_level')
     plt.title('Satisfaction by Department')
     plt.xticks(rotation=45)
     plt.show()
```

```
[ ]: # Create a plot as needed by salary
     plt.figure(figsize=(10, 6))
     sns.boxplot(x='salary', y='satisfaction_level', data=df, order=['low',␣
      ↪'medium', 'high'])
     plt.title('Satisfaction by Salary Level')
     plt.show()
```

```
[ ]: # Create a plot
     plt.figure(figsize = (16, 9))
     sns.scatterplot(data = df,
                     x = 'average_monthly_hours',
                y = 'satisfaction_level',
                hue = 'left',
                alpha = 0.4)

plt.axvline(x = 167, color = 'Red', ls = '--')
plt.legend(labels = ['167 hrs/month' ,'left', 'stayed'])
plt.title('Monthly Hours by Satisfaction', fontsize= '18')
```

we can observe that individuals who left worked less than those who stayed.

```
[ ]: # Plot `department` split into groups by `left`
     plt.figure(figsize = (4, 4))
     sns.histplot(data = df,
                  x = "department",
                  hue = "left",
                  discrete = True,
                  multiple = "dodge",
                  shrink = 0.8)
     plt.legend(labels = ["Left", "Stayed"])
     plt.title("Histogram of Employees Per Department")
     plt.xlabel("Department")
     plt.xticks(rotation = 45, horizontalalignment = "right")
     plt.show()
```

We could see the employees from sales department left more.

```
[ ]: plt.figure(figsize=(16,9))
     heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, cmap= sns.
      ↪color_palette("Greens", as_cmap = True))
     heatmap.set_title('Correlation Heatmap', fontdict={'fontsize': 14}, pad=12)
```

3.1.2 Insights

The above heatmap clearly demonstrates a negative correlation between satisfaction and left, which makes sense–an employee who is satisfied at the company is less likely to leave the company. Furthermore, the heatmap confirms correlations among last_eval, num_projects, and avg_hrs_month.

4 paCe: Construct Stage

- Determine which models are most appropriate
- Construct the model
- Confirm model assumptions
- Evaluate model results to determine how well your model fits the data

Recall model assumptions

Logistic Regression model assumptions - Outcome variable is categorical - Observations are independent of each other - No severe multicollinearity among X variables - No extreme outliers - Linear relationship between each X variable and the logit of the outcome variable - Sufficiently large sample size

4.1 Step 3. Model Building, Step 4. Results and Evaluation

- Fit a model that predicts the outcome variable using two or more independent variables
- Check model assumptions
- Evaluate the model

4.1.1 Modeling: Logistic Regression Model

The target variable is catogerical varible, we will be using Logistic regression technique that models a categorical variable based on one or more independent variables.

Logistic regression Note that binomial logistic regression suits the task because it involves binary classification.

Before splitting the data, encode the non-numeric variables. There are two: `department` and `salary`.

`department` is a categorical variable, which means you can dummy it for modeling.

`salary` is categorical too, but it's ordinal. There's a hierarchy to the categories, so it's better not to dummy this column, but rather to convert the levels to numbers, 0–2.

[48]:
```
# Copy the dataframe
df_enc = df.copy()

# Encode the `salary` column as an ordinal numeric category
df_enc['salary'] = (
    df_enc['salary'].astype('category')
    .cat.set_categories(['low', 'medium', 'high'])
    .cat.codes
)

# Dummy encode the `department` column
df_enc = pd.get_dummies(df_enc, drop_first=False)

# Display the new dataframe
df_enc.head()
```

[48]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
1                0.80             0.86               5                    262
2                0.11             0.88               7                    272
```

```
3                0.72             0.87              5                  223
4                0.37             0.52              2                  159

   tenure  work_accident  left  promotion_last_5years  salary  department_IT  \
0       3              0     1                      0       0              0
1       6              0     1                      0       1              0
2       4              0     1                      0       1              0
3       5              0     1                      0       0              0
4       3              0     1                      0       0              0

   department_RandD  department_accounting  department_hr  \
0                 0                      0              0
1                 0                      0              0
2                 0                      0              0
3                 0                      0              0
4                 0                      0              0

   department_management  department_marketing  department_product_mng  \
0                      0                     0                       0
1                      0                     0                       0
2                      0                     0                       0
3                      0                     0                       0
4                      0                     0                       0

   department_sales  department_support  department_technical
0                 1                   0                     0
1                 1                   0                     0
2                 1                   0                     0
3                 1                   0                     0
4                 1                   0                     0
```

[49]:
```
# Select rows without outliers in `tenure` and save resulting dataframe in a␣
↪new variable
df_logreg = df_enc[(df_enc['tenure'] >= lower_limit) & (df_enc['tenure'] <=␣
↪upper_limit)]

# Display first few rows of new dataframe
df_logreg.head()
```

[49]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
2                0.11             0.88               7                    272
3                0.72             0.87               5                    223
4                0.37             0.52               2                    159
5                0.41             0.50               2                    153

   tenure  work_accident  left  promotion_last_5years  salary  department_IT  \
0       3              0     1                      0       0              0
2       4              0     1                      0       1              0
3       5              0     1                      0       0              0
4       3              0     1                      0       0              0
5       3              0     1                      0       0              0

   department_RandD  department_accounting  department_hr  \
0                 0                      0              0
2                 0                      0              0
3                 0                      0              0
4                 0                      0              0
5                 0                      0              0

   department_management  department_marketing  department_product_mng  \
0                      0                     0                       0
2                      0                     0                       0
3                      0                     0                       0
4                      0                     0                       0
5                      0                     0                       0

   department_sales  department_support  department_technical
0                 1                   0                     0
2                 1                   0                     0
3                 1                   0                     0
4                 1                   0                     0
5                 1                   0                     0
```

[50]:
```python
# Isolate the outcome variable
y = df_logreg['left']

# Display first few rows of the outcome variable
y.head()
```

[50]:
```
0    1
2    1
3    1
4    1
5    1
Name: left, dtype: int64
```

[51]:
```python
# Select the features you want to use in your model
X = df_logreg.drop('left', axis=1)

# Display the first few rows of the selected features
X.head()
```

[51]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
2                0.11             0.88               7                    272
3                0.72             0.87               5                    223
4                0.37             0.52               2                    159
5                0.41             0.50               2                    153

   tenure  work_accident  promotion_last_5years  salary  department_IT  \
0       3              0                      0       0              0
2       4              0                      0       1              0
3       5              0                      0       0              0
4       3              0                      0       0              0
5       3              0                      0       0              0

   department_RandD  department_accounting  department_hr  \
0                 0                      0              0
2                 0                      0              0
3                 0                      0              0
4                 0                      0              0
5                 0                      0              0

   department_management  department_marketing  department_product_mng  \
0                      0                     0                       0
2                      0                     0                       0
3                      0                     0                       0
4                      0                     0                       0
5                      0                     0                       0

   department_sales  department_support  department_technical
0                 1                   0                     0
2                 1                   0                     0
3                 1                   0                     0
4                 1                   0                     0
5                 1                   0                     0
```

[ ]:
```

```

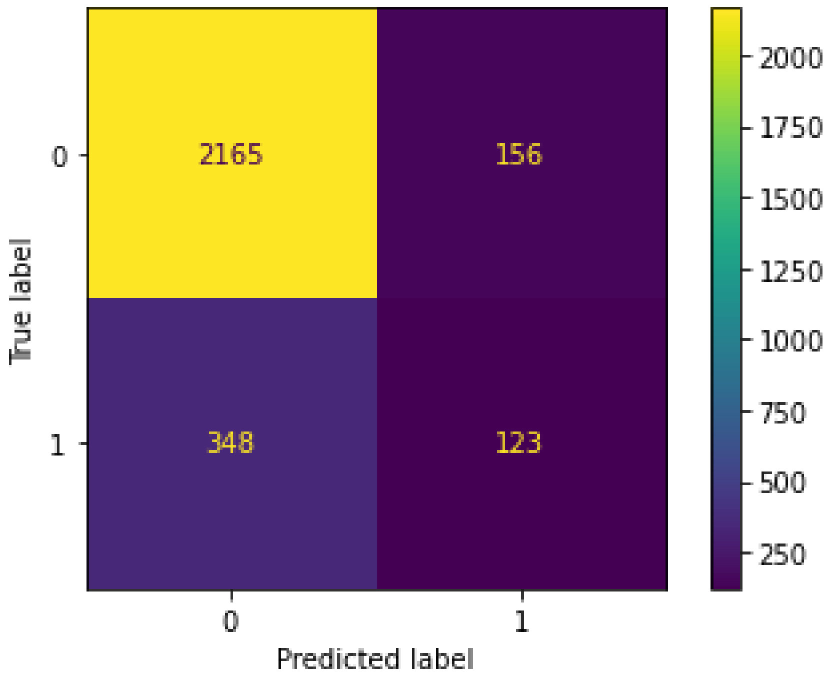
[52]:
```python
# Split the data into training set and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪stratify=y, random_state=42)

# Construct a logistic regression model and fit it to the training dataset
log_clf = LogisticRegression(random_state=42, max_iter=500).fit(X_train,␣
↪y_train)
```

[53]:
```python
# Display the first few rows of the selected features
X_train.head()
```

[53]:
```
      satisfaction_level  last_evaluation  number_project  \
9784                0.43             0.81               3
6535                0.74             0.68               3
5217                0.87             0.50               4
8421                0.61             0.56               2
6134                0.65             0.79               4

      average_monthly_hours  tenure  work_accident  promotion_last_5years  \
9784                    102       3              0                      0
6535                    206       2              1                      0
5217                    267       2              1                      0
8421                    123       2              0                      0
6134                    196       2              0                      0

      salary  department_IT  department_RandD  department_accounting  \
9784       1              0                 0                      0
6535       1              0                 0                      0
5217       1              0                 0                      1
8421       1              0                 0                      0
6134       1              0                 0                      1

      department_hr  department_management  department_marketing  \
9784              0                      0                     0
6535              0                      0                     0
5217              0                      0                     0
8421              0                      0                     0
6134              0                      0                     0

      department_product_mng  department_sales  department_support  \
9784                       0                 1                   0
6535                       0                 0                   1
5217                       0                 0                   0
8421                       0                 1                   0
6134                       0                 0                   0

      department_technical
9784                     0
6535                     0
5217                     0
8421                     0
6134                     0
```

[54]:
```
# Use the logistic regression model to get predictions on the test set

y_pred = log_clf.predict(X_test)
```

[55]:
```
# Compute values for confusion matrix
log_cm = confusion_matrix(y_test, y_pred, labels=log_clf.classes_)

# Create display of confusion matrix
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm,
                                  display_labels=log_clf.classes_)

# Plot confusion matrix
log_disp.plot(values_format='')

# Display plot
plt.show()
```



Quadrants display:

Upper left: number of true negatives.

Bottom right: number of true positives.

Upper right: number of false positives.

Bottom left: number of false negatives.

Meaning if each quadrant:

True negatives: number of employees the model predicted that would stay, and actually stayed. True positives: number of employees the model predicted that would leave, and actually left. False positives: number of emplyees the model predicted that would leave, and actually stayed. False negatives: number of employees the model predicted that would stay, and actually left. A perfect model would yield true positives and true negatives, and no false positives and false negatives.

Now we can evaluate the model performance by accuracy, precision, recall and f1.

[56]:
```
df_logreg['left'].value_counts(normalize=True)
```

[56]:
```
0    0.831468
1    0.168532
Name: left, dtype: float64
```

There is an approximately 83%-17% split. So the data is not perfectly balanced, but it is not too imbalanced. If it was more severely imbalanced, you might want to resample the data to make it more balanced. In this case, you can use this data without modifying the class balance and continue evaluating the model.

[57]:
```
# Create classification report for logistic regression model
target_names = ['Predicted would not leave', 'Predicted would leave']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
                           precision    recall  f1-score   support

Predicted would not leave       0.86      0.93      0.90      2321
    Predicted would leave       0.44      0.26      0.33       471

                 accuracy                           0.82      2792
                macro avg       0.65      0.60      0.61      2792
             weighted avg       0.79      0.82      0.80      2792
```

The classification report above shows that the logistic regression model achieved a precision of 79%, recall of 82%, f1-score of 80% (all weighted averages), and accuracy of 82%. However, if it's most important to predict employees who leave, then the scores are significantly lower.

4.1.2 Modeling Approach B: Tree-based Model

This approach covers implementation of Decision Tree and Random Forest.

[58]:
```
# Isolate the outcome variable
y = df_enc['left']

# Display the first few rows of `y`
y.head()
```

[58]:
```
0    1
1    1
2    1
3    1
4    1
```

```
Name: left, dtype: int64
```

[59]:
```
# Select the features
X = df_enc.drop('left', axis=1)

# Display the first few rows of `X`
X.head()
```

[59]:
```
   satisfaction_level  last_evaluation  number_project  average_monthly_hours  \
0                0.38             0.53               2                    157
1                0.80             0.86               5                    262
2                0.11             0.88               7                    272
3                0.72             0.87               5                    223
4                0.37             0.52               2                    159

   tenure  work_accident  promotion_last_5years  salary  department_IT  \
0       3              0                      0       0              0
1       6              0                      0       1              0
2       4              0                      0       1              0
3       5              0                      0       0              0
4       3              0                      0       0              0

   department_RandD  department_accounting  department_hr  \
0                 0                      0              0
1                 0                      0              0
2                 0                      0              0
3                 0                      0              0
4                 0                      0              0

   department_management  department_marketing  department_product_mng  \
0                      0                     0                       0
1                      0                     0                       0
2                      0                     0                       0
3                      0                     0                       0
4                      0                     0                       0

   department_sales  department_support  department_technical
0                 1                   0                     0
1                 1                   0                     0
2                 1                   0                     0
3                 1                   0                     0
4                 1                   0                     0
```

[60]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪stratify=y, random_state=0)
```

[61]:
```
# Instantiate model
tree = DecisionTreeClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }

# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}

# Instantiate GridSearch
tree1 = GridSearchCV(tree, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

[62]:
```
#Fit the decision tree model to the training data.
tree1.fit(X_train, y_train)
```

[62]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=0, splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'precision', 'recall', 'roc_auc', 'f1', 'accuracy'},
             verbose=0)
```

[63]:
```
#Identify the optimal values for the decision tree parameters.
# Check best parameters
tree1.best_params_
```

[63]:
```
{'max_depth': 4, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

[64]:
```
#Identify the best AUC score achieved by the decision tree model on the␣
↪training set.
# Check best AUC score on CV
```

```
tree1.best_score_
```

[64]:
```
0.969819392792457
```

With a score of 97%, we can say that this model predicts the people that will leave the company quite well. Now, we will use the scores from the gridsearch to see how well our model did.

[65]:
```
def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
        model_name (string): what you want the model to be called in the output␣
 ↪table
        model_object: a fit GridSearchCV object
        metric (string): precision, recall, f1, accuracy, or auc

    Returns a pandas df with the F1, recall, precision, accuracy, and auc scores
    for the model with the best mean 'metric' score across all validation folds.
 ↪
    '''

    # Create dictionary that maps input metric to actual metric name in␣
 ↪GridSearchCV
    metric_dict = {'auc': 'mean_test_roc_auc',
                   'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy'
                  }

    # Get all the results from the CV and put them in a df
    cv_results = pd.DataFrame(model_object.cv_results_)

    # Isolate the row of the df with the max(metric) score
    best_estimator_results = cv_results.iloc[cv_results[metric_dict[metric]].
 ↪idxmax(), :]

    # Extract Accuracy, precision, recall, and f1 score from that row
    auc = best_estimator_results.mean_test_roc_auc
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy

    # Create table of results
    table = pd.DataFrame()
    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy],
                          'auc': [auc]
                        })

    return table
```

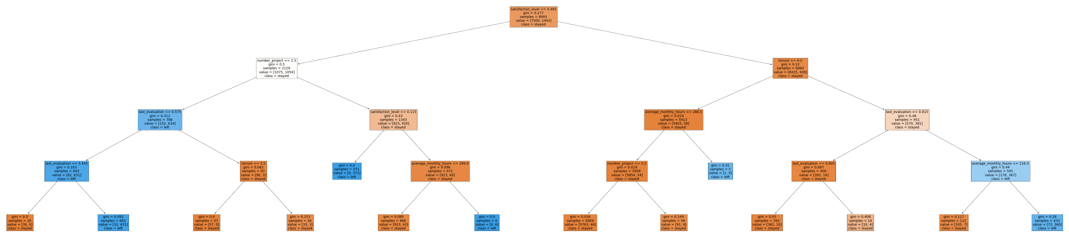
[66]:
```
#Use the function just defined to get all the scores from grid search.
# Get all CV scores
tree1_cv_results = make_results('decision tree cv', tree1, 'auc')
tree1_cv_results
```

[66]:
```
              model  precision    recall        F1  accuracy       auc
0  decision tree cv   0.914552  0.916949  0.915707  0.971978  0.969819
```

All of these scores from the decision tree model are strong indicators of good model performance.

Recall that decision trees can be vulnerable to overfitting, and random forests avoid overfitting by incorporating multiple trees to make predictions. You could construct a random forest model next.

[67]:
```
plt.figure(figsize = (85,20))
plot_tree(tree1.best_estimator_, max_depth =6, fontsize = 14,
         feature_names=X.columns, class_names={0:'stayed', 1:'left'},␣
 ↪filled=True)
plt.show()
```



###Random forest

[72]:
```
# Instantiate model
rf = RandomForestClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }

# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}

# Instantiate GridSearch
rf1 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

[74]:
```
rf1
```

[74]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,…
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                         'max_samples': [0.7, 1.0],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [300, 500]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'precision', 'recall', 'roc_auc', 'f1', 'accuracy'},
             verbose=0)
```

[76]:
```
rf1.fit(X_train, y_train)
```

[76]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         n_estimators=100, n_jobs=None,…
                                                         verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                         'max_samples': [0.7, 1.0],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [300, 500]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'precision', 'recall', 'roc_auc', 'f1', 'accuracy'},
             verbose=0)
```

[77]:
```python
# Check best AUC score on CV
rf1.best_score_
```

[77]:
```
0.9804250949807172
```

[78]:
```python
# Check best params
rf1.best_params_
```

[78]:
```
{'max_depth': 5,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 500}
```

[79]:
```python
# Get all CV scores
rf1_cv_results = make_results('random forest cv', rf1, 'auc')
print(tree1_cv_results)
print(rf1_cv_results)
```

```
               model  precision    recall        F1  accuracy       auc
0  decision tree cv   0.914552  0.916949  0.915707  0.971978  0.969819
               model  precision    recall        F1  accuracy       auc
0  random forest cv   0.950023  0.915614  0.932467  0.977983  0.980425
```

The evaluation scores of the random forest model are better than those of the decision tree model, with the exception of recall (the recall score of the random forest model is approximately 0.001 lower, which is a negligible amount). This indicates that the random forest model mostly outperforms the decision tree model.

Next, you can evaluate the final model on the test set.

[80]:
```python
def get_scores(model_name:str, model, X_test_data, y_test_data):
    '''
    Generate a table of test scores.

    In:
        model_name (string):  How you want your model to be named in the output␣
↪table
        model:                A fit GridSearchCV object
        X_test_data:          numpy array of X_test data
        y_test_data:          numpy array of y_test data

    Out: pandas df of precision, recall, f1, accuracy, and AUC scores for your␣
↪model
    '''

    preds = model.best_estimator_.predict(X_test_data)

    auc = roc_auc_score(y_test_data, preds)
    accuracy = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)

    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'f1': [f1],
                          'accuracy': [accuracy],
                          'AUC': [auc]
                         })

    return table
```

[81]:
```
# Get predictions on test data
rf1_test_scores = get_scores('random forest1 test', rf1, X_test, y_test)
rf1_test_scores
```

[81]:
```
                 model  precision    recall        f1  accuracy       AUC
0  random forest1 test   0.964211  0.919679  0.941418  0.980987  0.956439
```

The test scores are very similar to the validation scores, which is good. This appears to be a strong model. Since this test set was only used for this model, you can be more confident that your model's performance on this data is representative of how it will perform on new, unseeen data.

The first round of decision tree and random forest models included all variables as features. This next round will incorporate feature engineering to build improved models.

You could proceed by dropping `satisfaction_level` and creating a new feature that roughly captures whether an employee is overworked. You could call this new feature `overworked`. It will be a binary variable.

[82]:
```python
# Drop `satisfaction_level` and save resulting dataframe in new variable
df2 = df_enc.drop('satisfaction_level', axis=1)

# Display first few rows of new dataframe
df2.head()
```

[82]:
```
   last_evaluation  number_project  average_monthly_hours  tenure  \
0             0.53               2                    157       3
1             0.86               5                    262       6
2             0.88               7                    272       4
3             0.87               5                    223       5
4             0.52               2                    159       3

   work_accident  left  promotion_last_5years  salary  department_IT  \
0              0     1                      0       0              0
1              0     1                      0       1              0
2              0     1                      0       1              0
3              0     1                      0       0              0
4              0     1                      0       0              0

   department_RandD  department_accounting  department_hr  \
0                 0                      0              0
1                 0                      0              0
2                 0                      0              0
3                 0                      0              0
4                 0                      0              0

   department_management  department_marketing  department_product_mng  \
0                      0                     0                       0
1                      0                     0                       0
2                      0                     0                       0
3                      0                     0                       0
4                      0                     0                       0

   department_sales  department_support  department_technical
0                 1                   0                     0
1                 1                   0                     0
2                 1                   0                     0
3                 1                   0                     0
4                 1                   0                     0
```

[83]:
```python
# Create `overworked` column. For now, it's identical to average monthly hours.
df2['overworked'] = df2['average_monthly_hours']

# Inspect max and min average monthly hours values
print('Max hours:', df2['overworked'].max())
print('Min hours:', df2['overworked'].min())
```

```
Max hours: 310
Min hours: 96
```

166.67 is approximately the average number of monthly hours for someone who works 50 weeks per year, 5 days per week, 8 hours per day.

You could define being overworked as working more than 175 hours per month on average.

To make the `overworked` column binary, you could reassign the column using a boolean mask. - `df3['overworked'] > 175` creates a series of booleans, consisting of `True` for every value > 175 and `False` for every values 175 - `.astype(int)` converts all `True` to `1` and all `False` to `0`

[84]:
```
# Define `overworked` as working > 175 hrs/week
df2['overworked'] = (df2['overworked'] > 175).astype(int)

# Display first few rows of new column
df2['overworked'].head()
```

[84]:
```
0    0
1    1
2    1
3    1
4    0
Name: overworked, dtype: int64
```

[85]:
```
# Drop the `average_monthly_hours` column
df2 = df2.drop('average_monthly_hours', axis=1)

# Display first few rows of resulting dataframe
df2.head()
```

[85]:
```
   last_evaluation  number_project  tenure  work_accident  left  \
0             0.53               2       3              0     1
1             0.86               5       6              0     1
2             0.88               7       4              0     1
3             0.87               5       5              0     1
4             0.52               2       3              0     1

   promotion_last_5years  salary  department_IT  department_RandD  \
0                      0       0              0                 0
1                      0       1              0                 0
2                      0       1              0                 0
3                      0       0              0                 0
4                      0       0              0                 0

   department_accounting  department_hr  department_management  \
0                      0              0                     0
1                      0              0                     0
2                      0              0                     0
```

```
3                       0              0                      0
4                       0              0                      0

   department_marketing  department_product_mng  department_sales  \
0                     0                       0                 1
1                     0                       0                 1
2                     0                       0                 1
3                     0                       0                 1
4                     0                       0                 1

   department_support  department_technical  overworked
0                   0                     0           0
1                   0                     0           1
2                   0                     0           1
3                   0                     0           1
4                   0                     0           0
```

[86]:
```
# Isolate the outcome variable
y = df2['left']

# Select the features
X = df2.drop('left', axis=1)
```

[87]:
```
# Create test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪stratify=y, random_state=0)
```

Decision tree - Round 2

[88]:
```
# Instantiate model
tree = DecisionTreeClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }

# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}

# Instantiate GridSearch
tree2 = GridSearchCV(tree, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

[89]:
```
%%time
tree2.fit(X_train, y_train)
```

```
CPU times: user 2.23 s, sys: 1.82 ms, total: 2.23 s
```

```
Wall time: 2.23 s
```

```
[89]: GridSearchCV(cv=4, error_score=nan,
                   estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                                    criterion='gini', max_depth=None,
                                                    max_features=None,
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    presort='deprecated',
                                                    random_state=0, splitter='best'),
                   iid='deprecated', n_jobs=None,
                   param_grid={'max_depth': [4, 6, 8, None],
                               'min_samples_leaf': [2, 5, 1],
                               'min_samples_split': [2, 4, 6]},
                   pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
                   scoring={'precision', 'recall', 'roc_auc', 'f1', 'accuracy'},
                   verbose=0)
```

```
[90]: # Check best params
      tree2.best_params_
```

```
[90]: {'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split': 6}
```

```
[91]: # Check best AUC score on CV
      tree2.best_score_
```

```
[91]: 0.9586752505340426
```

This model performs very well, even without satisfaction levels and detailed hours worked data.

Next, check the other scores.

```
[92]: # Get all CV scores
      tree2_cv_results = make_results('decision tree2 cv', tree2, 'auc')
      print(tree1_cv_results)
      print(tree2_cv_results)
```

```
              model  precision    recall        F1  accuracy       auc
0  decision tree cv   0.914552  0.916949  0.915707  0.971978  0.969819
               model  precision    recall        F1  accuracy       auc
0  decision tree2 cv   0.856693  0.903553  0.878882  0.958523  0.958675
```

Some of the other scores fell. That's to be expected given fewer features were taken into account in this round of the model. Still, the scores are very good.

Random forest - Round 2

[94]:
```
# Instantiate model
rf = RandomForestClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }

# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}

# Instantiate GridSearch
rf2 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='roc_auc')
```

[95]:
```
%%time
rf2.fit(X_train, y_train) # --> Wall time: 7min 5s
```

```
CPU times: user 6min 51s, sys: 1.13 s, total: 6min 52s
Wall time: 6min 52s
```

[95]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,…
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, None], 'max_features': [1.0],
                         'max_samples': [0.7, 1.0],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [300, 500]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring={'precision', 'recall', 'roc_auc', 'f1', 'accuracy'},
             verbose=0)
```

```
[109]: # Check best params
       rf2.best_params_
```

```
[109]: {'max_depth': 5,
        'max_features': 1.0,
        'max_samples': 0.7,
        'min_samples_leaf': 2,
        'min_samples_split': 2,
        'n_estimators': 300}
```

```
[110]: # Check best AUC score on CV
       rf2.best_score_
```

```
[110]: 0.9648100662833985
```

```
[111]: # Get all CV scores
       rf2_cv_results = make_results('random forest2 cv', rf2, 'auc')
       print(tree2_cv_results)
       print(rf2_cv_results)
```

```
              model  precision    recall        F1  accuracy       auc
0  decision tree2 cv   0.856693  0.903553  0.878882  0.958523  0.958675
              model  precision    recall        F1  accuracy      auc
0  random forest2 cv   0.866758  0.878754  0.872407  0.957411  0.96481
```

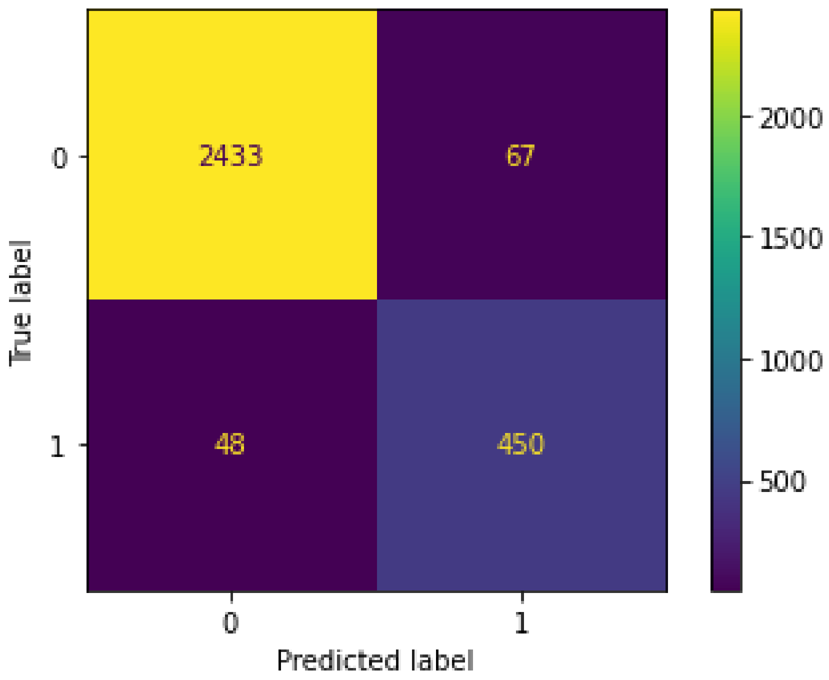
Again, the scores dropped slightly, but the random forest performs better than the decision tree if using AUC as the deciding metric.

Score the champion model on the test set now.

```
[113]: # Get predictions on test data
       rf2_test_scores = get_scores('random forest2 test', rf2, X_test, y_test)
       rf2_test_scores
```

```
[113]:                 model  precision    recall      f1  accuracy       AUC
       0  random forest2 test   0.870406  0.903614  0.8867  0.961641  0.938407
```

```
[114]: # Generate array of values for confusion matrix
       preds = rf2.best_estimator_.predict(X_test)
       cm = confusion_matrix(y_test, preds, labels=rf2.classes_)

       # Plot confusion matrix
       disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                     display_labels=rf2.classes_)
       disp.plot(values_format='');
```

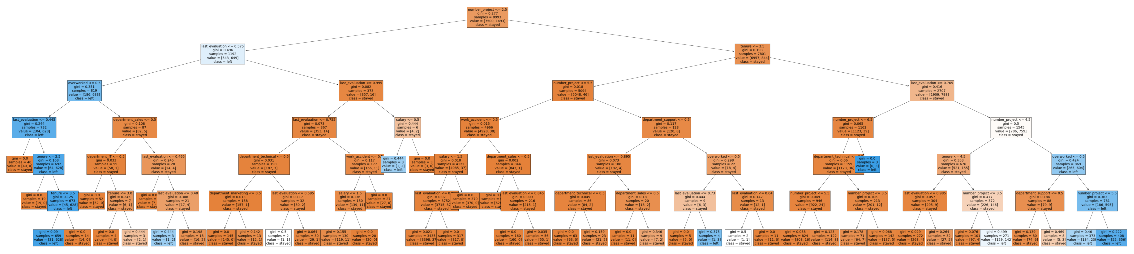


The model predicts more false positives than false negatives, which means that some employees may be identified as at risk of quitting or getting fired, when that's actually not the case. But this is still a strong model.

For exploratory purpose, you might want to inspect the splits of the decision tree model and the most important features in the random forest model.

Decision Tree Feature

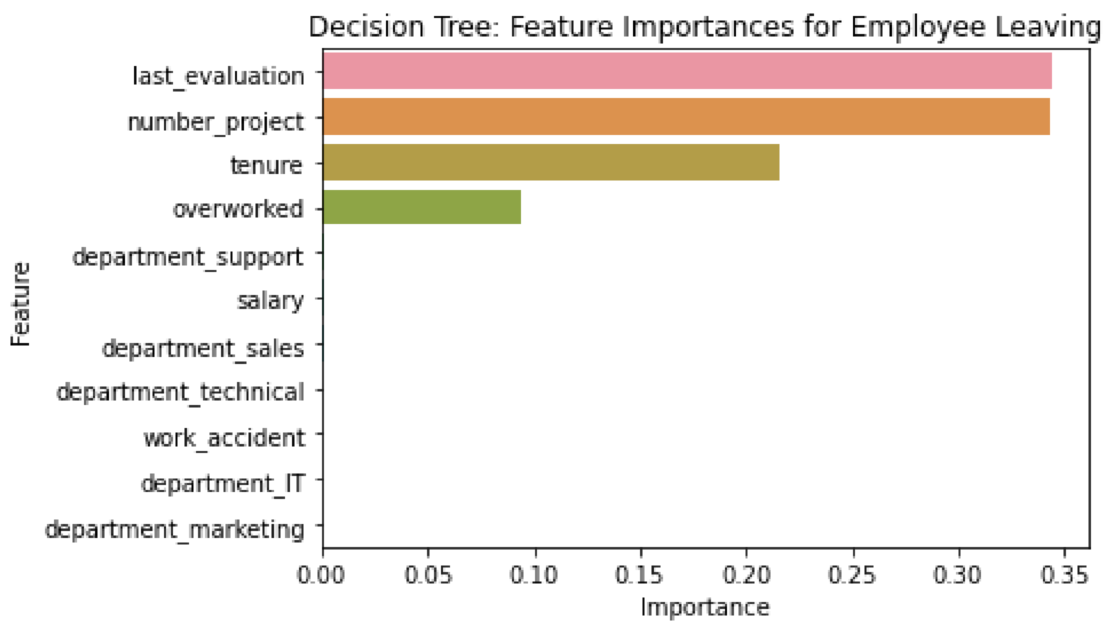
[115]:
```python
# Plot the tree
plt.figure(figsize=(85,20))
plot_tree(tree2.best_estimator_, max_depth=6, fontsize=14, feature_names=X.
 ↪columns,
          class_names={0:'stayed', 1:'left'}, filled=True);
plt.show()
```



[116]:
```
#tree2_importances = pd.DataFrame(tree2.best_estimator_.feature_importances_,␣
↪columns=X.columns)
tree2_importances = pd.DataFrame(tree2.best_estimator_.feature_importances_,
                                 columns=['gini_importance'],
                                 index=X.columns
                                )
tree2_importances = tree2_importances.sort_values(by='gini_importance',␣
↪ascending=False)

# Only extract the features with importances > 0
tree2_importances = tree2_importances[tree2_importances['gini_importance'] != 0]
tree2_importances
```

[116]:
```
                      gini_importance
last_evaluation              0.343958
number_project               0.343385
tenure                       0.215681
overworked                   0.093498
department_support           0.001142
salary                       0.000910
department_sales             0.000607
department_technical         0.000418
work_accident                0.000183
department_IT                0.000139
department_marketing         0.000078
```

[117]:
```
sns.barplot(data=tree2_importances, x="gini_importance", y=tree2_importances.
↪index, orient='h')
plt.title("Decision Tree: Feature Importances for Employee Leaving",␣
↪fontsize=12)
plt.ylabel("Feature")
plt.xlabel("Importance")
plt.show()
```

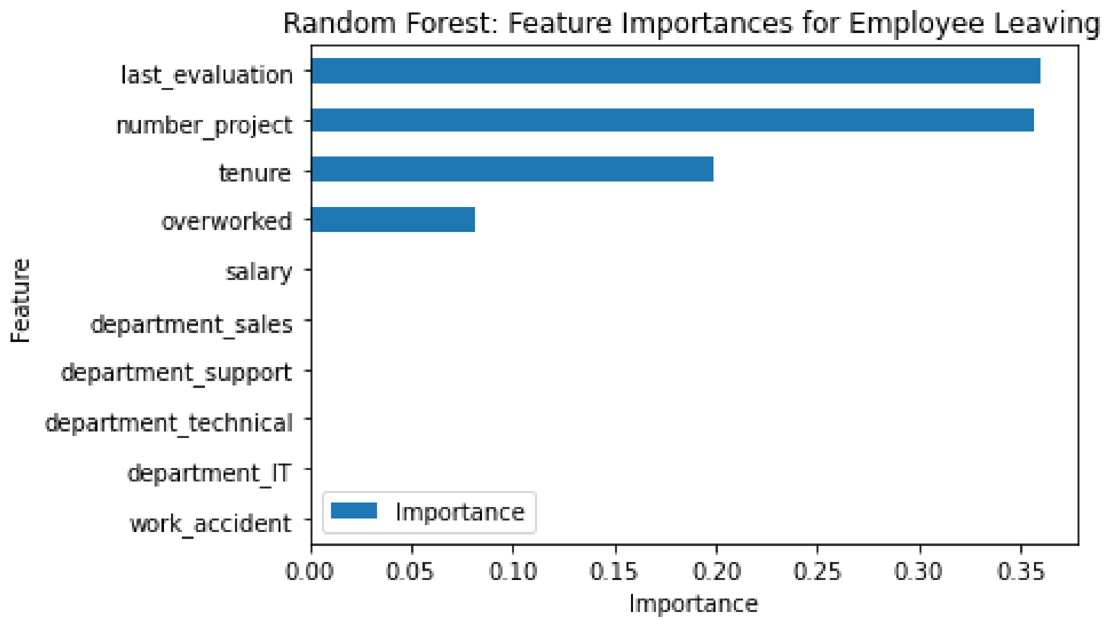


The barplot above shows that in this decision tree model, `last_evaluation`, `number_project`, `tenure`, and `overworked` have the highest importance, in that order. These variables are most helpful in predicting the outcome variable, `left`.

Random forest feature importance Now, plot the feature importances for the random forest model.

[118]:
```
# Get feature importances
feat_impt = rf2.best_estimator_.feature_importances_

# Get indices of top 10 features
ind = np.argpartition(rf2.best_estimator_.feature_importances_, -10)[-10:]

# Get column labels of top 10 features
feat = X.columns[ind]

# Filter `feat_impt` to consist of top 10 feature importances
feat_impt = feat_impt[ind]

y_df = pd.DataFrame({"Feature":feat,"Importance":feat_impt})
y_sort_df = y_df.sort_values("Importance")
fig = plt.figure()
ax1 = fig.add_subplot(111)

y_sort_df.plot(kind='barh',ax=ax1,x="Feature",y="Importance")

ax1.set_title("Random Forest: Feature Importances for Employee Leaving",␣
↪fontsize=12)
```

```
ax1.set_ylabel("Feature")
ax1.set_xlabel("Importance")

plt.show()
```



The plot above shows that in this random forest model, `last_evaluation`, `number_project`, `tenure`, and `overworked` have the highest importance, in that order. These variables are most helpful in predicting the outcome variable, `left`, and they are the same as the ones used by the decision tree model.

5 pacE: Execute Stage

Recall evaluation metrics

- **AUC** is the area under the ROC curve; it's also considered the probability that the model ranks a random positive example more highly than a random negative example.
- **Precision** measures the proportion of data points predicted as True that are actually True, in other words, the proportion of positive predictions that are true positives.
- **Recall** measures the proportion of data points that are predicted as True, out of all the data points that are actually True. In other words, it measures the proportion of positives that are correctly classified.
- **Accuracy** measures the proportion of data points that are correctly classified.
- **F1-score** is an aggregation of precision and recall.

5.0.1 Summary of model results

Decision Tree Model: Without conducting feature engineering the decision Decision Tree Model achieved AUC of 97%, precision of 92.2%, recall of 92%, f1-score of 92.2%, and accuracy of 97.4%. After conducting feature engineering the decision Decision Tree Model achieved AUC of 95.3%, precision of 86.4%, recall of 90.2%, f1-score of 88.3%, and accuracy of 96.02%.

Random Forest Model: Without conducting feature engineering the Random Forest Model achieved AUC of 95.4%, precision of 97.9%, recall of 94.3%, f1-score of 93.3%, and accuracy of 97.8%. . After conducting feature engineering the Random Forest Model achieved AUC of 96.6%, precision of 90.9%, recall of 91.5%, f1-score of 89.4%, and accuracy of 96.5%.

Overall Random Forest Model performs better than the Decision Tree Model.

5.0.2 Conclusion, Recommendations.

Both model feature importance show number of projects, last evaluation, tenure, and overworked are the most crucial variables for predicting the outcome variable, “left”.

Recommendations To determine the best number of projects employees can handle, conduct a comprehensive study considering factors like project complexity, duration, employee skill levels, and workload distribution. Analyze historical data and gather feedback from employees to optimize project allocation.

Conduct employee satisfaction surveys, hold exit interviews, and establish open communication channels to identify the reasons for employee dissatisfaction. Address issues related to work-life balance, compensation, career growth opportunities, company culture, and management support.

Evaluate the performance and potential of employees with over four years of experience and high evaluation scores. Consider promoting those who exhibit leadership qualities, strong skills, and alignment with the company’s values and goals.

Assess the workload distribution and employee capacity to determine if hiring additional workers is necessary. Hiring more employees can help reduce the burden on existing staff, improve productivity, and enhance overall job satisfaction.

Review the evaluation scoring policy to ensure it accurately reflects employee performance and contributions. Consider additional criteria beyond the number of hours worked, such as quality of output, teamwork, innovation, and meeting project deadlines. This will incentivize a healthier work environment and promote employee well-being.

Congratulations! You’ve completed this lab. However, you may not notice a green check mark next to this item on Coursera’s platform. Please continue your progress regardless of the check mark. Just click on the “save” icon at the top of this notebook to ensure your work has been logged.