Week 4 - Laboratory

ECON441B

Mauricio Vargas-Estrada
Master in Quantitative Economics
University of California - Los Angeles

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

1.) Import, split data into `X` / `y`, plot y data as bar charts, turn `X` categorical variables binary and tts.

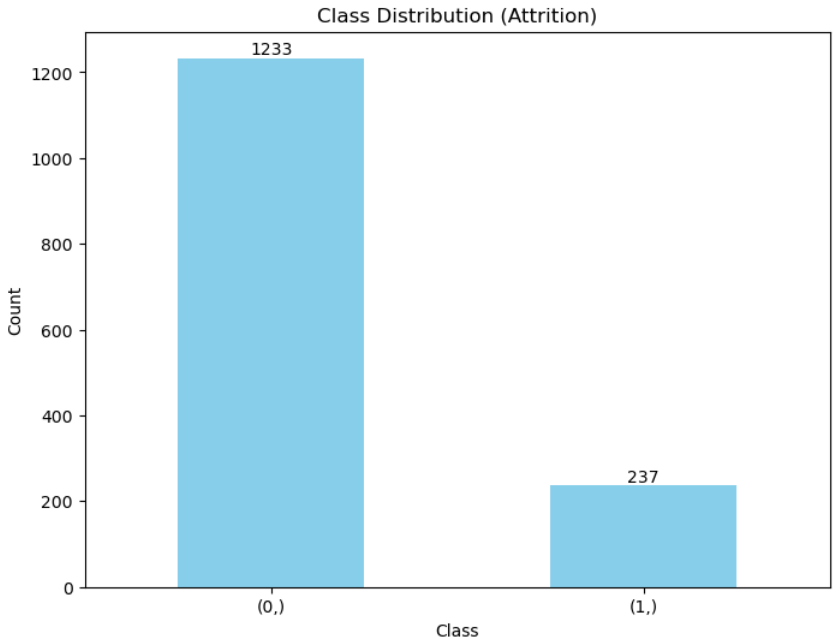
In [ ]:
```
df = pd.read_csv("HR_Analytics.csv")
```

In [ ]:
```
# Separate the target variable and the features
y = df[["Attrition"]].copy()
X = df.drop("Attrition", axis = 1)
```

In [ ]:
```
# Convert the target variable to binary
y["Attrition"] = [1 if i == "Yes" else 0 for i in y["Attrition"]]
```

In [ ]:
```
# Bar plot of the class distribution in y (Attrition)
class_counts = y.value_counts()
```

In [ ]:
```
plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution (Attrition)')
# Remove rotation of x-axis labels
plt.xticks(rotation=0)
# Add labels to the bars
for i, count in enumerate(class_counts):
    plt.text(i, count, str(count), ha='center', va='bottom')
plt.show()
```



In [ ]:
```
# Convert MonthlyIncome to float
X['MonthlyIncome'] = X['MonthlyIncome'].astype(float)
```

In [ ]:
```
# Step 1: Identify string columns
string_columns = X.columns[X.dtypes == 'object']
```

In [ ]:
```
# Step 2: Convert string columns to categorical
for col in string_columns:
    X[col] = pd.Categorical(X[col])
```

In [ ]:
```
# Step 3: Create dummy columns
X = pd.get_dummies(
    X,
    columns=string_columns,
    prefix=string_columns,
    drop_first = True
)
```

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(
    X, y,
    test_size = 0.20,
    random_state = 42
)
```

2.) Using the default Decision Tree. What is the IN/Out of Sample accuracy?

In [ ]:
```
clf = DecisionTreeClassifier()
```

In [ ]:
```
clf.fit(x_train,y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [ ]:
```
y_fitted = clf.predict(x_train)
y_pred = clf.predict(x_test)
```

In [ ]:
```
acc_in = accuracy_score(y_train,y_fitted)
acc_out = accuracy_score(y_test,y_pred)
```

In [ ]:
```
print("IN SAMPLE ACCURACY : " , round(acc_in, 2))
print("OUT OF SAMPLE ACCURACY : " , round(acc_out, 2))
```

```
IN SAMPLE ACCURACY :  1.0
OUT OF SAMPLE ACCURACY :  0.77
```

3.) Run a grid search cross validation using F1 score to find the best metrics. What is the In and Out of Sample now?

In [ ]:
```
# Define the hyperparameter grid to search through
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': np.arange(1, 11),
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

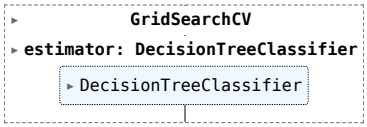
In [ ]:
```
# Defining selection score criteria
scoring = make_scorer(f1_score, average='weighted')
```

In [ ]:
```
dt_classifier = DecisionTreeClassifier(random_state=42)
```

In [ ]:
```
grid_search = GridSearchCV(
    estimator = dt_classifier,
    param_grid = param_grid,
    scoring = scoring,
    cv=5,
    refit = True,
    n_jobs = -1
)
```

In [ ]:
```
grid_search.fit(x_train, y_train)
```

Out[ ]:



In [ ]:
```
# Get the best parameters and the best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best F1-Score:", best_score)
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split': 2}
Best F1-Score: 0.8214764475510983
```

In [ ]:
```
y_fitted = grid_search.predict(x_train)
y_pred = grid_search.predict(x_test)
```

In [ ]:
```
temp = accuracy_score(y_train, y_fitted)
print("IN SAMPLE ACCURACY : " , round(temp,2))
```

```
IN SAMPLE ACCURACY :  0.91
```

In [ ]:
```
temp = accuracy_score(y_test, y_pred)
print("OUT OF SAMPLE ACCURACY : " , round(temp,2))
```

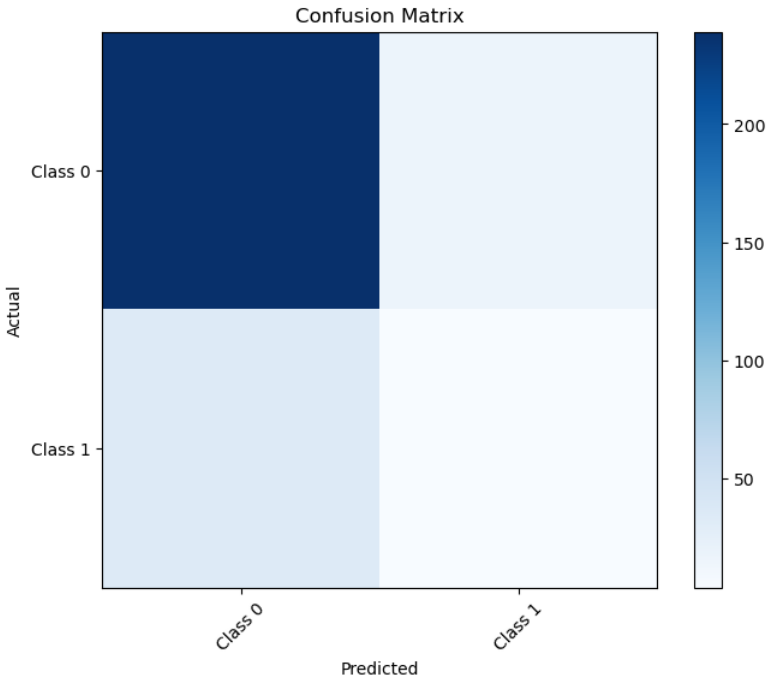
```
OUT OF SAMPLE ACCURACY :  0.83
```

4.) Plot .....

In [ ]:
```
# Make predictions on the test data
y_pred = grid_search.predict(x_test)
# Taking the prob. of the positive class
y_prob = grid_search.predict_proba(x_test)[:, 1]
```

In [ ]:
```
# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

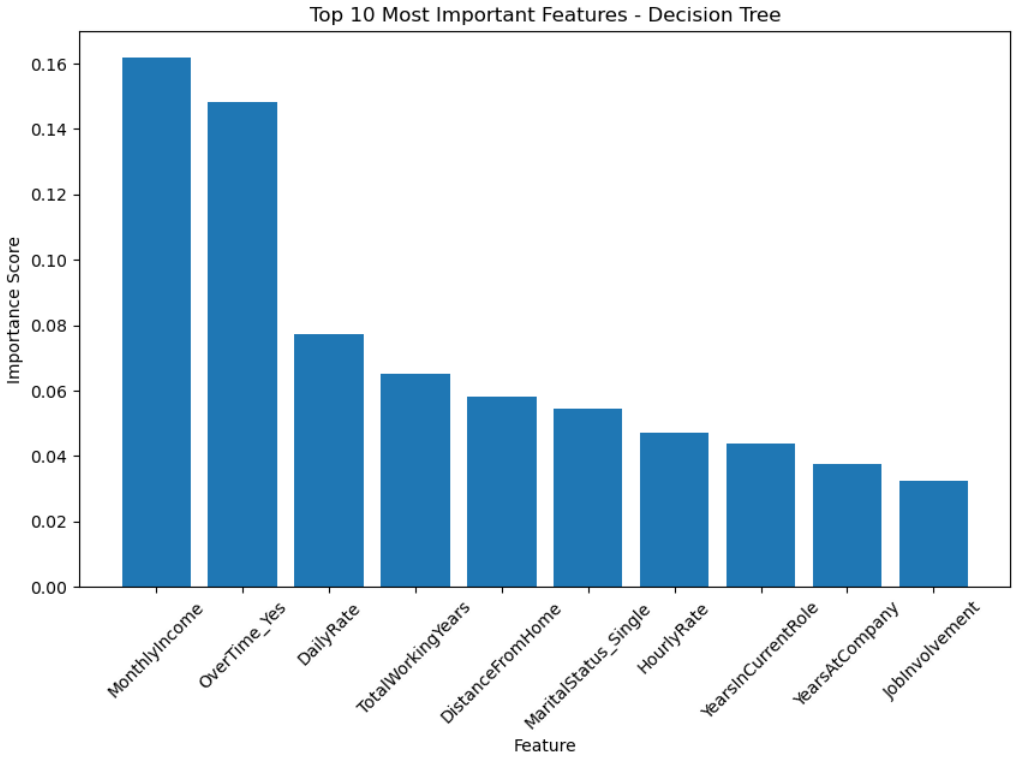
In [ ]:
```
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(conf_matrix))
plt.xticks(tick_marks, ['Class 0', 'Class 1'], rotation=45)
plt.yticks(tick_marks, ['Class 0', 'Class 1'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [ ]:
```
feature_importance = grid_search.best_estimator_.feature_importances_
```

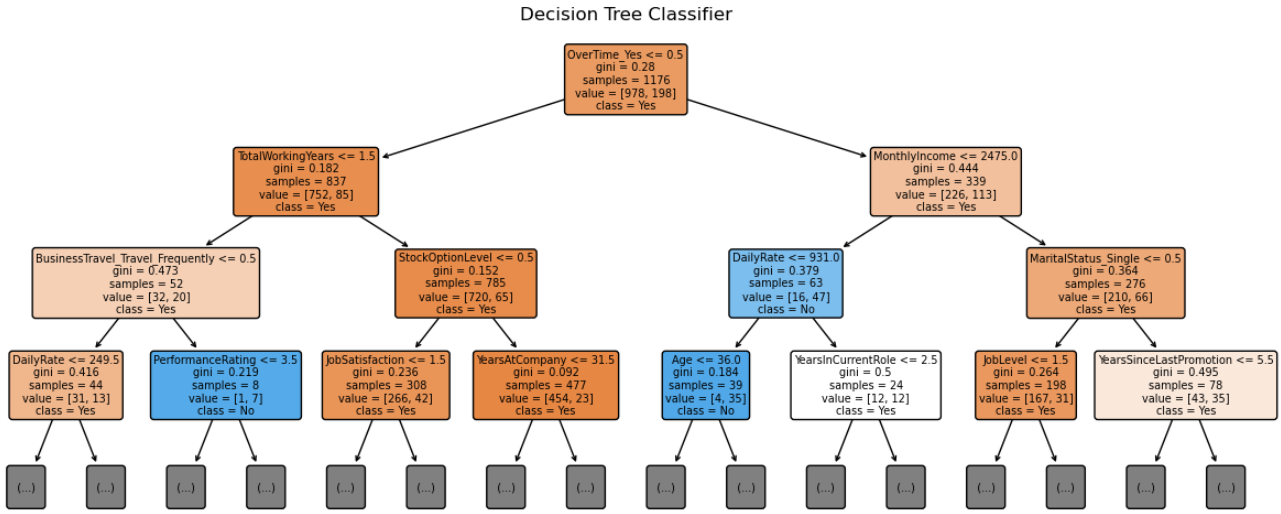
In [ ]:
```
# Sort features by importance and select the top 10
top_n = 10
top_feature_indices = np.argsort(feature_importance)[::-1][:top_n]
top_feature_names = X.columns[top_feature_indices]
top_feature_importance = feature_importance[top_feature_indices]
```

In [ ]:
```
# Plot the top 10 most important features
plt.figure(figsize=(10, 6))
plt.bar(top_feature_names, top_feature_importance)
plt.xlabel('Feature')
plt.ylabel('Importance Score')
plt.title('Top 10 Most Important Features - Decision Tree')
plt.xticks(rotation=45)
plt.show()
```



In [ ]:
```
# Plot the Decision Tree for better visualization of the selected features
plt.figure(figsize=(15, 6))
```

```
plot_tree(
    grid_search.best_estimator_,
    filled = True,
    feature_names = X.columns,
    class_names = ["Yes", "No"],
    rounded = True,
    fontsize = 7,
    max_depth = 3
)
plt.title('Decision Tree Classifier')
plt.show()
```



5.) Looking at the graphs. what would be your suggestions to try to improve customer retention? What additional information would you need for a better plan. Plot anything you think would assist in your assessment.

The decision tree tells us which variables are most important, but we do not know whether these variables have a negative or positive influence on attrition. Correlation can be used to estimate the relationship between the variables and the target variable.

In [ ]:
```
# Calculating the correlation between the features and the target variable
def calculate_correlation(X, feature_name, y):
    feature = X[feature_name]

    coef, _ = np.polyfit(feature, y, 1)
    coef = coef[0]
    return coef * 100
```

In [ ]:
```
# Calculate the correlation between the top 10 features and the target variable
correlations = {}
for feature in top_feature_names:
    coef = calculate_correlation(X, feature, y)
    correlations[feature] = coef
```

In [ ]:
```
correlations = pd.Series(correlations).sort_values(ascending=False)
```

In [ ]:
```
correlations
```

Out[ ]:
```
OverTime_Yes            20.092414
MaritalStatus_Single    13.831915
DistanceFromHome         0.353592
MonthlyIncome           -0.001249
DailyRate               -0.005165
HourlyRate              -0.012387
YearsAtCompany          -0.806949
TotalWorkingYears       -0.808760
YearsInCurrentRole      -1.630040
JobInvolvement          -6.721568
dtype: float64
```

The two features with the greatest influence on attrition are `MonthlyIncome` and `OverTime_Yes`. `MonthlyIncome` has a negative correlation with attrition, meaning that as monthly income increases, the likelihood of attrition decreases. On the other hand, `OverTime_Yes` has a positive correlation with attrition, meaning that if an employee works overtime, the likelihood of attrition increases.

The previous result suggests that perhaps the hourly income is low compared to alternatives in the market. This suggests that the company should consider studying the market to determine if they are paying competitive salaries.

6.) Using the Training Data, if they made everyone work overtime. What would have been the expected difference in client retention?

In [ ]:
```
# Creating a copie of the training data, with everyone
# working overtime.
```

```
x_ex_1 = x_train.copy()

x_ex_1['OverTime_Yes'] = True
```

In [ ]:
```
# Calculating the attrition using the grid search model.
y_ex_1 = grid_search.predict(x_ex_1)
```

In [ ]:
```
# Calculating the expected difference between everyone working
# overtime and the original data. Because is the change in expectations
# we should calculate it using the fitted values in the training sample.
expected_diff = sum(y_ex_1 - y_fitted)
f'Expected difference in attrition: {expected_diff}'
```

Out[ ]:
```
'Expected difference in attrition: 141'
```

Considering the training scenario, if all employees work overtime, an increase of 141 employees leaving the company is expected.

7.) If they company loses an employee, there is a cost to train a new employee for a role ~2.8 * their monthly income.

To make someone not work overtime costs the company 2K per person.

Is it profitable for the company to remove overtime? If so/not by how much?

What do you suggest to maximize company profits?

In [ ]:
```
# Getting the index of the employees that left the company and
# where working overtime in the training data.
condition = np.logical_and(
    y_fitted == 1,
    x_train['OverTime_Yes'] == True
)
att_overtime = x_train[condition].index
# Ordering the arr_overtime by the monthly income of the employees.
att_overtime = x_train.loc[att_overtime].sort_values(
    by = 'MonthlyIncome',
    ascending=False
)
att_overtime = att_overtime.index
```

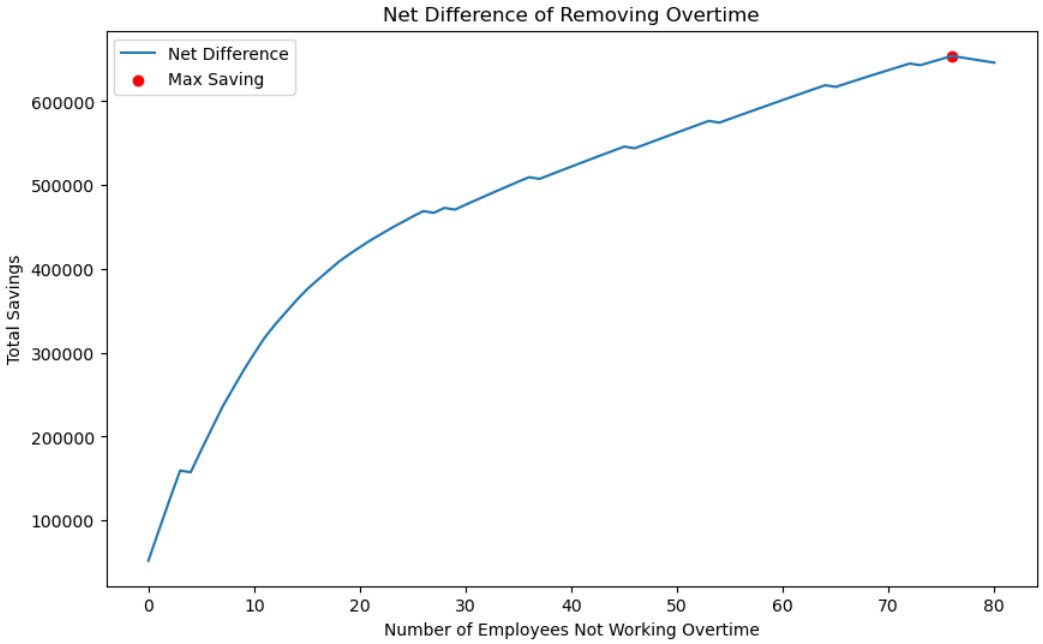
In [ ]:
```
# Defining a function to calculate the cost of removing overtime
# for a set of employees that left the company.
def cost_no_overtime(employee_index, x_train, clf):
    # Create a copy of the training data.
    x_temp_ex = x_train.copy()
    # Eliminating overtime for the employees that are leaving.
    x_temp_ex.loc[employee_index, 'OverTime_Yes'] = False
    # Predicting the attrition using the classifier.
    y_temp_ex = clf.predict(x_temp_ex)
    y_temp_ex = pd.Series(y_temp_ex, index=x_train.index)
    # The saving from those employees not leaving due to removing
    # overtime.
    saving = -1 * (y_temp_ex.loc[employee_index] - 1)
    saving = saving * x_temp_ex.loc[employee_index, 'MonthlyIncome']
    saving = saving * 2.8
    saving = saving.sum()
    # Calculating the cost of removing overtime.
    cost_overtime = 2000 * len(employee_index)
    # Returning the net difference between the cost of removing
    # overtime and the savings from the employees not leaving.
    # If the result is positive, it means that it is profitable
    return saving - cost_overtime
```

In [ ]:
```
# Testing removing overtime for the employees that left the company
# one by one.
saving = pd.DataFrame(
    index = range(len(att_overtime)),
    columns = ['Employee', 'Net Difference']
)

no_over = []
for i, employee in enumerate(att_overtime):
    no_over.append(employee)
    saving.loc[i, 'Employee'] = no_over.copy()
    saving.loc[i, 'Net Difference'] = cost_no_overtime(
        no_over, x_train, grid_search
    )
```

In [ ]:
```
max_saving = saving['Net Difference'].max()
max_saving_index = saving['Net Difference'].idxmax()
```

In [ ]:
```
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(saving['Net Difference'], label='Net Difference')
ax.scatter(max_saving_index, max_saving, color='red', label='Max Saving')
ax.legend()
ax.set_xlabel('Number of Employees Not Working Overtime')
ax.set_ylabel('Total Savings')
ax.set_title('Net Difference of Removing Overtime')
```

Out[ ]: Text(0.5, 1.0, 'Net Difference of Removing Overtime')



In this exercise, those individuals in the training sample who left the company and who worked overtime were identified. The savings that would be obtained if overtime work for each of them was eliminated were calculated. Workers were sorted by monthly income in descending order and the work of overtime was eliminated one by one. It was observed that the net savings increase as more overtime workers are eliminated, reaching a maximum savings of 65,4166.80 dollars. This result is achieved at a point where there are still employees who resign and work overtime, but the cost of training new employees is much higher than the cost of stopping working overtime. This result holds under the assumption that the cost of working overtime is the same for all employees, which is not realistic.

8.) Use your model and get the expected change in retention for raising and lowering peoples income. Plot the outcome of the experiment. Comment on the outcome of the experiment and your suggestions to maximize profit.

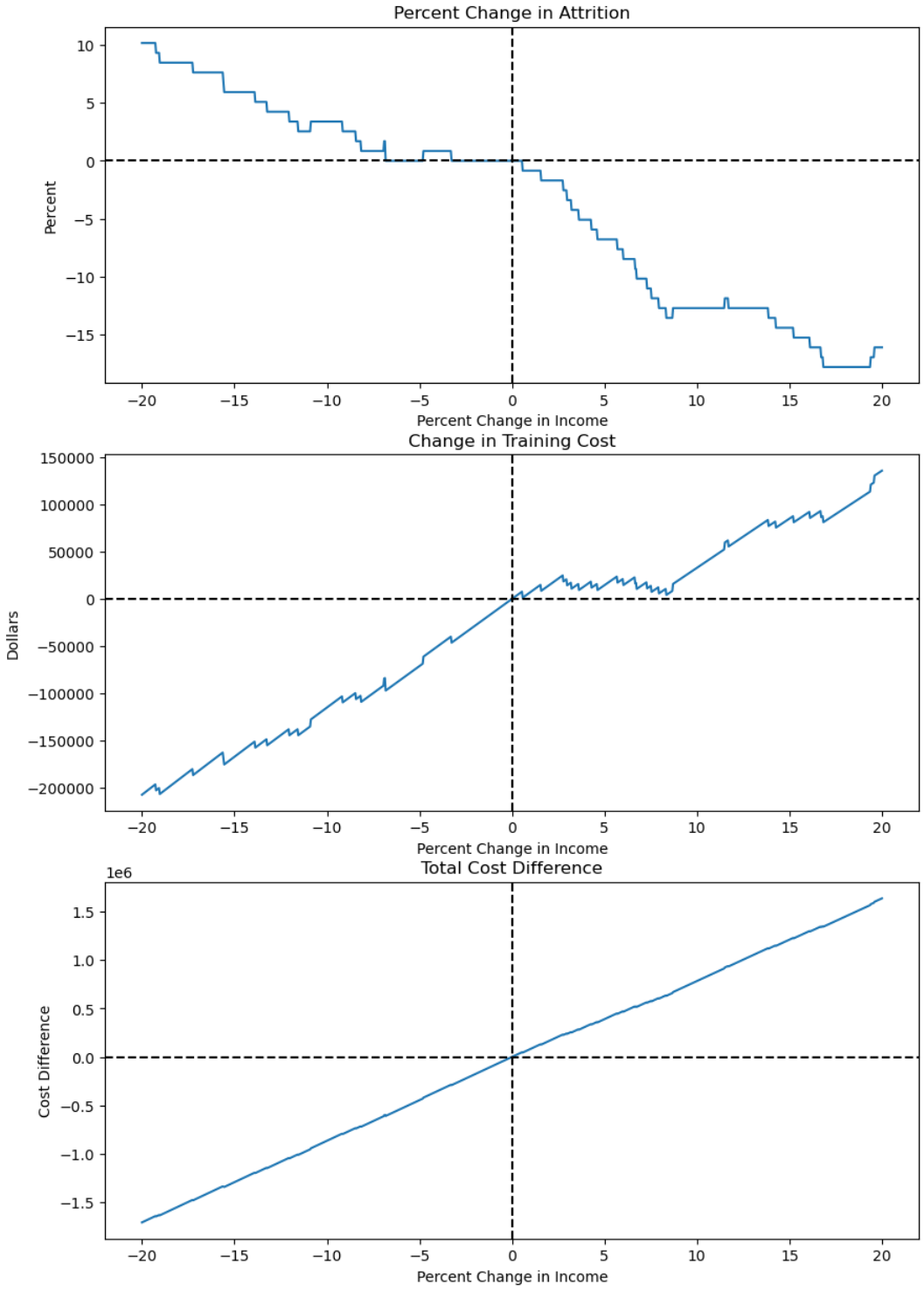
In [ ]:
```
# Defining a function to calculate the cost of raising income.
def cost_raise_income(percent, x_train, y_fitted, clf):
    # Predicting the attrition using the classifier.
    y_fitted = pd.DataFrame(y_fitted, index = x_train.index.values)
    y_fitted.columns = ['Attrition']

    # Getting the employees that left the company.
    condition = y_fitted['Attrition'] == 1
    att = list(y_fitted[condition].index.values)

    # Computing the cost of training new employees in the original
    # scenario.
    training_cost = 2.8 * x_train.loc[att, 'MonthlyIncome']
    training_cost = training_cost.sum()
    # Original attrition.
    attrition = y_fitted.sum().iloc[0].copy()

    # Create a copy of the training data.
    x_temp_ex = x_train.copy()
    # changing the income of all employees indexed by att
    x_temp_ex['MonthlyIncome'] = x_temp_ex['MonthlyIncome'].values * (1 + percent)

    # Computing the cost of the new monthly income. If is negative
    # it means that the company is saving money.
    diff_income = x_temp_ex['MonthlyIncome'] - x_train['MonthlyIncome']
    diff_income = diff_income.sum()

    # Predicting the attrition using the classifier.
    y_temp_ex = clf.predict(x_temp_ex)
    y_temp_ex = pd.DataFrame(y_temp_ex, index = x_train.index.values)
    y_temp_ex.columns = ['Attrition']

    # Attrition in the new scenario.
    attrition_ex = y_temp_ex.sum().iloc[0].copy()
    # Getting the employees that left the company.
    condition = y_temp_ex['Attrition'] == 1
    att_ex = list(y_temp_ex[condition].index.values)
    # Computing the cost of training new employees, asuming that
    # the company will hold the new salaries.
    training_cost_ex = 2.8 * x_temp_ex.loc[att_ex, 'MonthlyIncome']
    training_cost_ex = training_cost_ex.sum()

    # Change in attrition with the new salaries.
    diff_attrition =  attrition_ex/attrition - 1
    diff_attrition = diff_attrition * 100
    # Change in training cost with the new salaries.
    diff_training_cost = training_cost_ex - training_cost
```

```
    # Total difference in cost.
    diff_cost = diff_training_cost + diff_income

    return diff_attrition, diff_training_cost, diff_cost
```

In [ ]:
```
s_raise = np.linspace(-0.2, 0.2, 1000)
attr_diff = []
cost_diff = []
training_cost = []

for i in s_raise:
    temp_attr, temp_training, temp_cost = cost_raise_income(
        i, x_train, y_fitted, grid_search
    )
    attr_diff.append(temp_attr)
    cost_diff.append(temp_cost)
    training_cost.append(temp_training)
```

In [ ]:
```
fig, ax = plt.subplots(3, 1, figsize=(10, 15))
ax[0].plot(s_raise * 100, attr_diff)
ax[0].set_xlabel('Percent Change in Income')
ax[0].set_ylabel('Percent')
ax[0].axhline(0, color='black', linestyle='--')
ax[0].axvline(0, color='black', linestyle='--')
ax[0].set_title('Percent Change in Attrition')

ax[1].plot(s_raise * 100, training_cost)
ax[1].set_xlabel('Percent Change in Income')
ax[1].set_ylabel('Dollars')
ax[1].axhline(0, color='black', linestyle='--')
ax[1].axvline(0, color='black', linestyle='--')
ax[1].set_title('Change in Training Cost')

ax[2].plot(s_raise * 100, cost_diff)
ax[2].set_xlabel('Percent Change in Income')
ax[2].set_ylabel('Cost Difference')
ax[2].axhline(0, color='black', linestyle='--')
ax[2].axvline(0, color='black', linestyle='--')
ax[2].set_title('Total Cost Difference')
```

Out[ ]:
```
Text(0.5, 1.0, 'Total Cost Difference')
```



The exercise was conducted under the assumption that salary increases or decreases were generalized. It is observed that a reduction of up to 5% in salaries does not have a significant impact on attrition, representing a generalized saving.

The analysis of results from a decision tree model necessitates caution, as it is not a structural model. Unlike structural models that elucidate causal relationships between variables through theoretical underpinnings, decision trees are descriptive, focusing on pattern recognition without inferring causality. They segment data into homogeneous groups for predictions, not accommodating the analysis of how changes in one variable might causally affect another due to their data-driven nature. This limitation is crucial when considering the impact of interventions, such as a wage increase, where understanding the causal interplay among variables is essential. Thus, while decision trees can capture complex relationships and interactions, they do not provide insights into the causal mechanisms underlying these relationships, making them less suitable for exercises that require an understanding of the structural or causal impacts of variable changes.