4 - Decsion Trees Python

February 2, 2024

1 HR ATTRIBUTION

[1]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, f1_score
import numpy as np
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, auc
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, roc_auc_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score
```

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

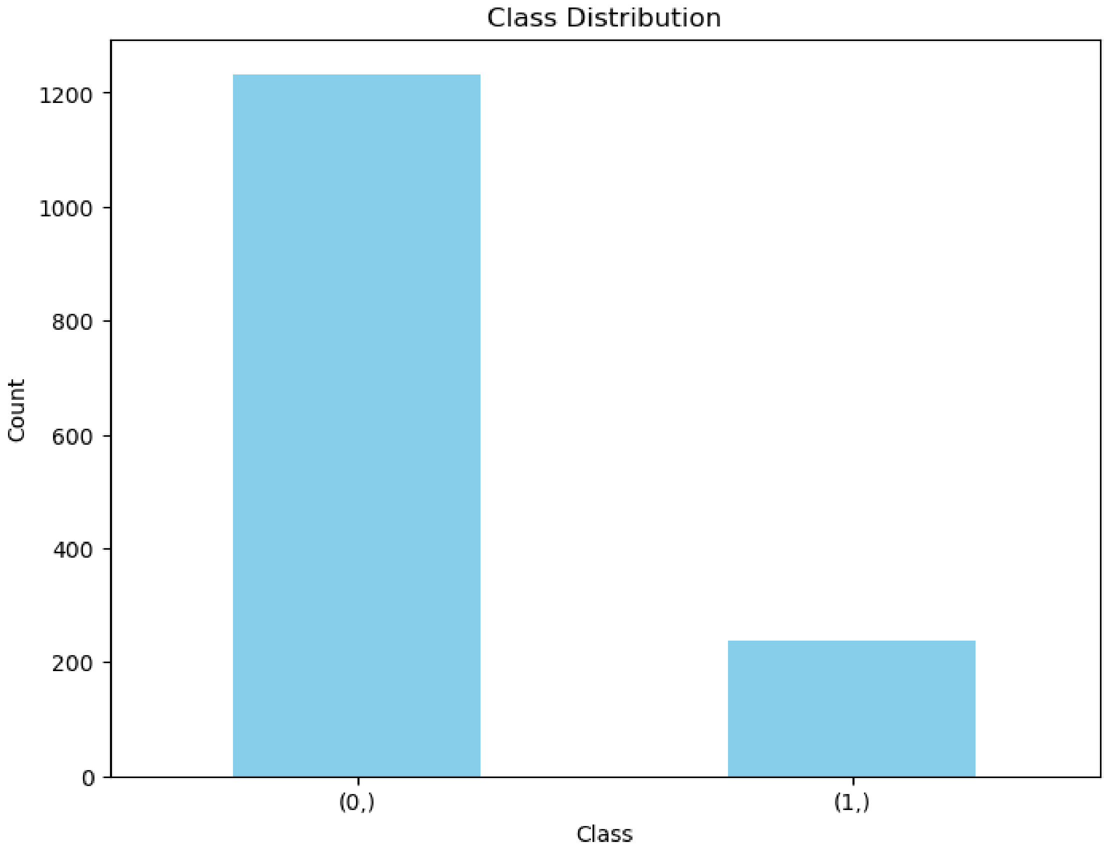
[2]:
```
df = pd.read_csv("HR_Analytics.csv")
```

[3]:
```
y = df[["Attrition"]].copy()
X = df.drop("Attrition", axis = 1)
```

[4]:
```
y["Attrition"] = [1 if i == "Yes" else 0 for i in y["Attrition"]]
```

[5]:
```
class_counts = y.value_counts()


plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution')
```

```
plt.xticks(rotation=0)  # Remove rotation of x-axis labels
plt.show()
```



[6]:
```
# Step 1: Identify string columns
string_columns = X.columns[X.dtypes == 'object']

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

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

[7]:
```
x_train,x_test,y_train,y_test=train_test_split(X,
 y, test_size=0.20, random_state=42)
```

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

[8]:
```
clf = DecisionTreeClassifier()
clf.fit(x_train,y_train)
y_pred=clf.predict(x_train)
acc=accuracy_score(y_train,y_pred)
print("IN SAMPLE ACCURACY : " , round(acc,2))

y_pred=clf.predict(x_test)
acc=accuracy_score(y_test,y_pred)
print("OUT OF SAMPLE ACCURACY : " , round(acc,2))
```

```
IN SAMPLE ACCURACY :  1.0
OUT OF SAMPLE ACCURACY :  0.75
```

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

[9]:
```
# Define the hyperparameter grid to search through
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': np.arange(1, 11),  # Range of max_depth values to try
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}


dt_classifier = DecisionTreeClassifier(random_state=42)

scoring = make_scorer(f1_score, average='weighted')

grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid,
 ↪scoring=scoring, cv=5)

grid_search.fit(x_train, y_train)

# 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
```

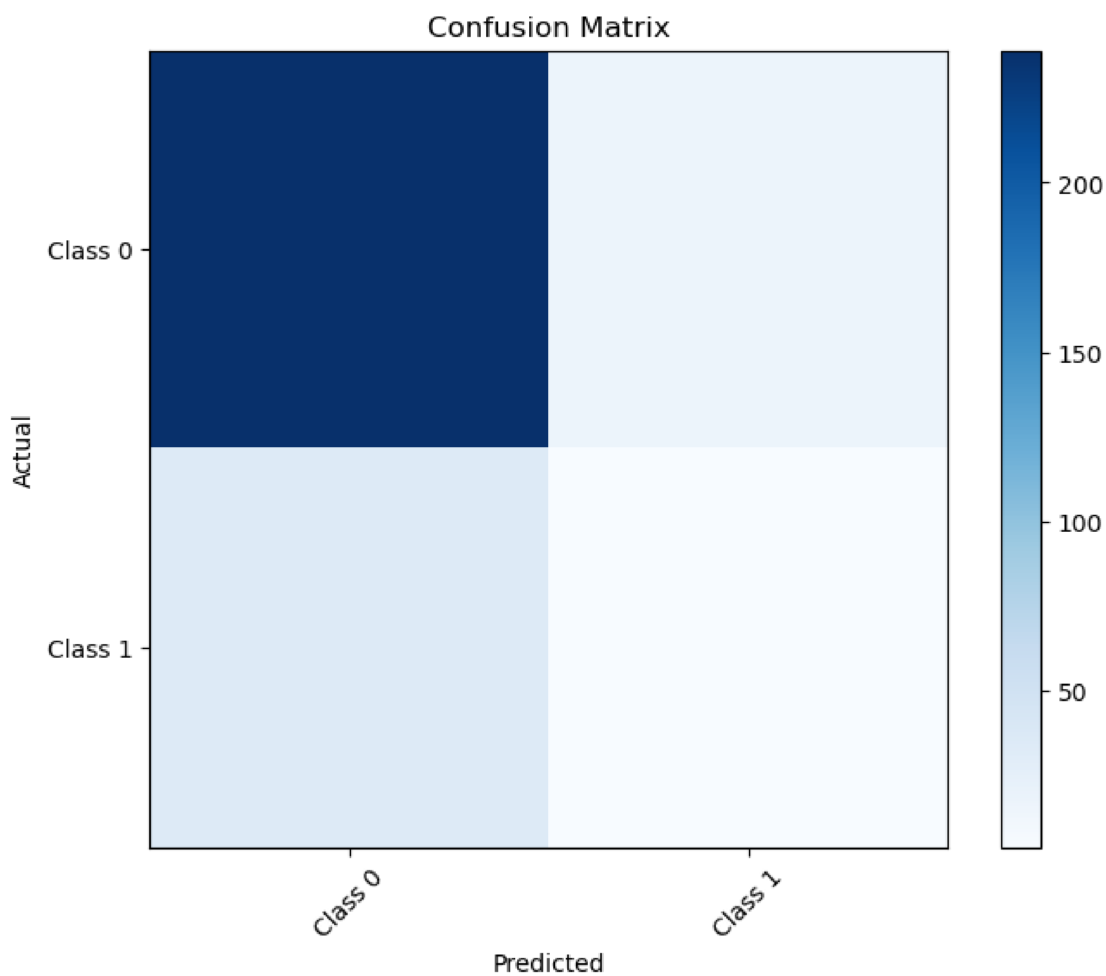
[10]:
```python
clf = tree.DecisionTreeClassifier(**best_params, random_state =42)
clf.fit(x_train,y_train)
y_pred=clf.predict(x_train)
acc=accuracy_score(y_train,y_pred)
print("IN SAMPLE ACCURACY : " , round(acc,2))

y_pred=clf.predict(x_test)
acc=accuracy_score(y_test,y_pred)
print("OUT OF SAMPLE ACCURACY : " , round(acc,2))
```

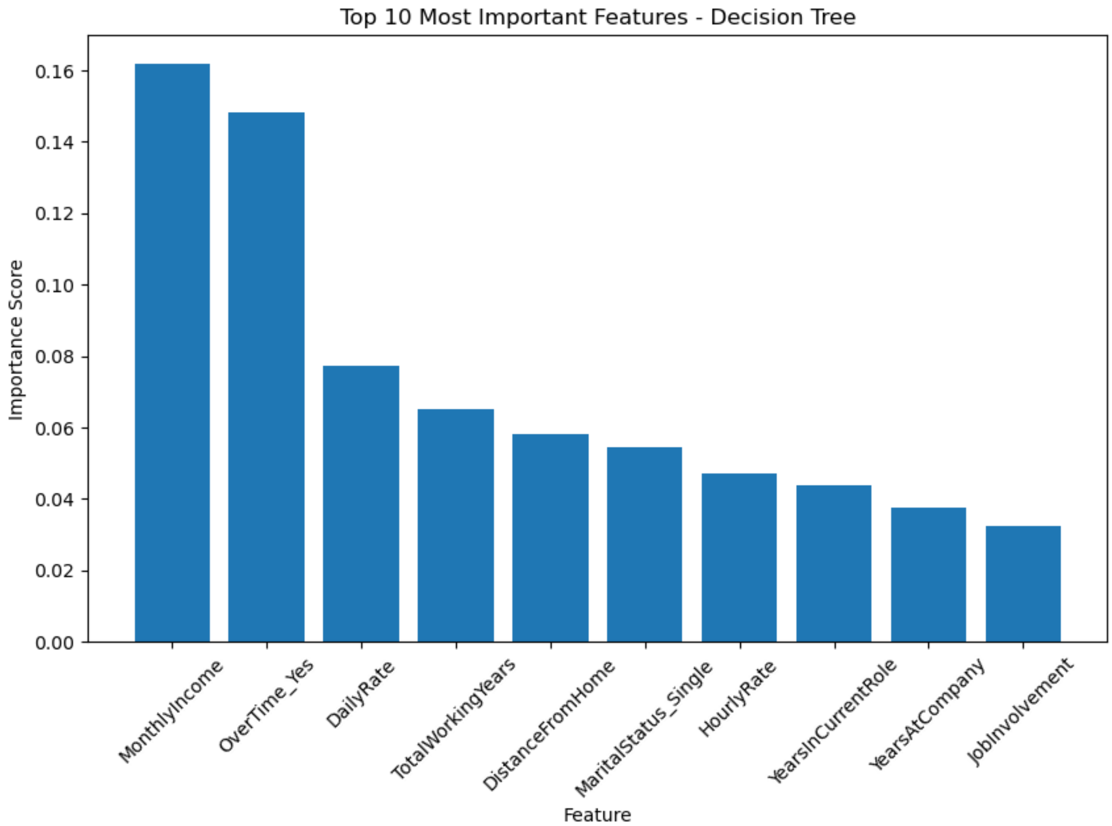
```
IN SAMPLE ACCURACY :  0.91
OUT OF SAMPLE ACCURACY :  0.83
```

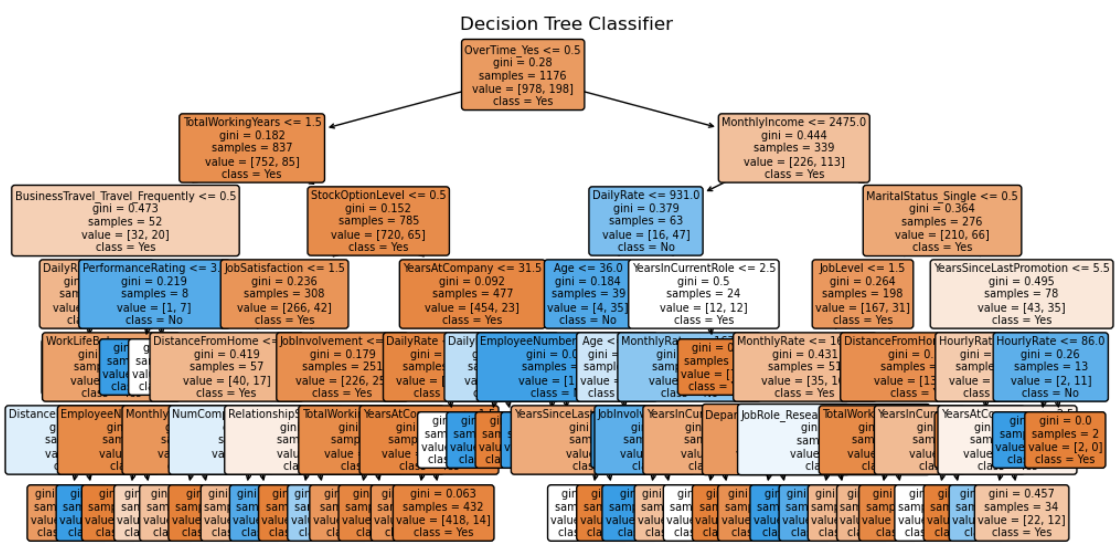
5 4.) Plot ……

[11]:
```python
# Make predictions on the test data
y_pred = clf.predict(x_test)
y_prob = clf.predict_proba(x_test)[:, 1]

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# 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()




feature_importance = clf.feature_importances_

# 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]
```

```
# 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()

# Plot the Decision Tree for better visualization of the selected features
plt.figure(figsize=(12, 6))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=["Yes", "No"],
    rounded=True, fontsize=7)
plt.title('Decision Tree Classifier')
plt.show()
```




Top 10 Most Important Features - Decision Tree
Importance Score
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
MonthlyIncome
OverTime_Yes
DailyRate
TotalWorkingYears
DistanceFromHome
MaritalStatus_Single
HourlyRate
YearsInCurrentRole
YearsAtCompany
JobInvolvement
Feature


Decision Tree Classifier
OverTime_Yes <= 0.5
gini = 0.28
samples = 1176
value = [978, 198]
class = Yes
TotalWorkingYears <= 1.5
gini = 0.182
samples = 837
value = [752, 85]
class = Yes
MonthlyIncome <= 2475.0
gini = 0.444
samples = 339
value = [226, 113]
class = Yes
BusinessTravel_Travel_Frequently <= 0.5
gini = 0.473
samples = 52
value = [32, 20]
class = Yes
StockOptionLevel <= 0.5
gini = 0.152
samples = 785
value = [720, 65]
class = Yes
DailyRate <= 931.0
gini = 0.379
samples = 63
value = [16, 47]
class = No
MaritalStatus_Single <= 0.5
gini = 0.364
samples = 276
value = [210, 66]
class = Yes
gini = 0.219
samples = 8
value = [1, 7]
class = No
JobSatisfaction <= 1.5
gini = 0.236
samples = 308
value = [266, 42]
class = Yes
YearsAtCompany <= 31.5
gini = 0.092
samples = 477
value = [454, 23]
class = Yes
Age <= 36.0
gini = 0.184
samples = 39
value = [4, 35]
class = No
YearsInCurrentRole <= 2.5
gini = 0.5
samples = 24
value = [12, 12]
class = Yes
JobLevel <= 1.5
gini = 0.264
samples = 198
value = [167, 31]
class = Yes
YearsSinceLastPromotion <= 5.5
gini = 0.495
samples = 78
value = [43, 35]
class = Yes
gini = 0.419
samples = 57
value = [40, 17]
class = Yes
gini = 0.179
gini = 0.431
HourlyRate <= 86.0
gini = 0.26
samples = 13
value = [2, 11]
class = No
gini = 0.0
samples = 2
value = [2, 0]
class = Yes
gini = 0.063
samples = 432
value = [418, 14]
class = Yes
gini = 0.457
samples = 34
value = [22, 12]
class = Yes

[ ]:

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

6.1 ANSWER :

[12]:
```
from scipy.stats import pearsonr

def calculate_correlation(X,feature_name,y):
    feature = X[feature_name]

    coef, _= pearsonr(feature,y)

    return(coef)
```

[13]:
```
np.corrcoef(np.array(X["OverTime_Yes"]),np.array(y["Attrition"]))
```

[13]:
```
array([[1.        , 0.24611799],
       [0.24611799, 1.        ]])
```

The correlation coefficient of 0.246 between overtime and attrition suggests a positive relationship. As overtime increases, attrition slightly increases as well. So introducing flexible working hours can help employees balance their work and personal lives, potentially reducing the need for overtime. Additional information such as employee feedback and cost of turnovers would be needed for a better plan.

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

[14]:
```
x_train_experiment = x_train.copy()
```

[15]:
```
x_train_experiment["OverTime_Yes"] = 0.
```

[16]:
```
y_pred = clf.predict(x_train)
y_pred_experiment = clf.predict(x_train_experiment)
```

[17]:
```
diff = sum(y_pred-y_pred_experiment)
print("The change in client retention would be",diff)
```

```
The change in client retention would be 59
```

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

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

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

11 What do you suggest to maximize company profits?

[18]:
```
x_train_experiment["Y"] = y_pred
x_train_experiment["Y_exp"] = y_pred_experiment
```

[19]:
```
x_train_experiment["RetChange"] = x_train_experiment["Y_exp"]-␣
↪x_train_experiment["Y"]
```

[20]:
```
sav = sum(-2.
↪8*x_train_experiment["RetChange"]*x_train_experiment["MonthlyIncome"])
```

[21]:
```
cost = len(x_train[x_train["OverTime_Yes"]==1])*2000
```

[22]:
```
sav - cost
```

[22]:
```
-117593.99999999977
```

The negative number of profits indicates that the cost of removing overtime and increasing training expenses exceeds the savings from reducing overtime expenses. This suggests that removing overtime may not be profitable due to the high cost of training new employees to maintain productivity levels. From a purely financial perspective, the company should keep overtime in order to maximize company profits.

12 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.

[23]:
```
raise_amount = 100
```

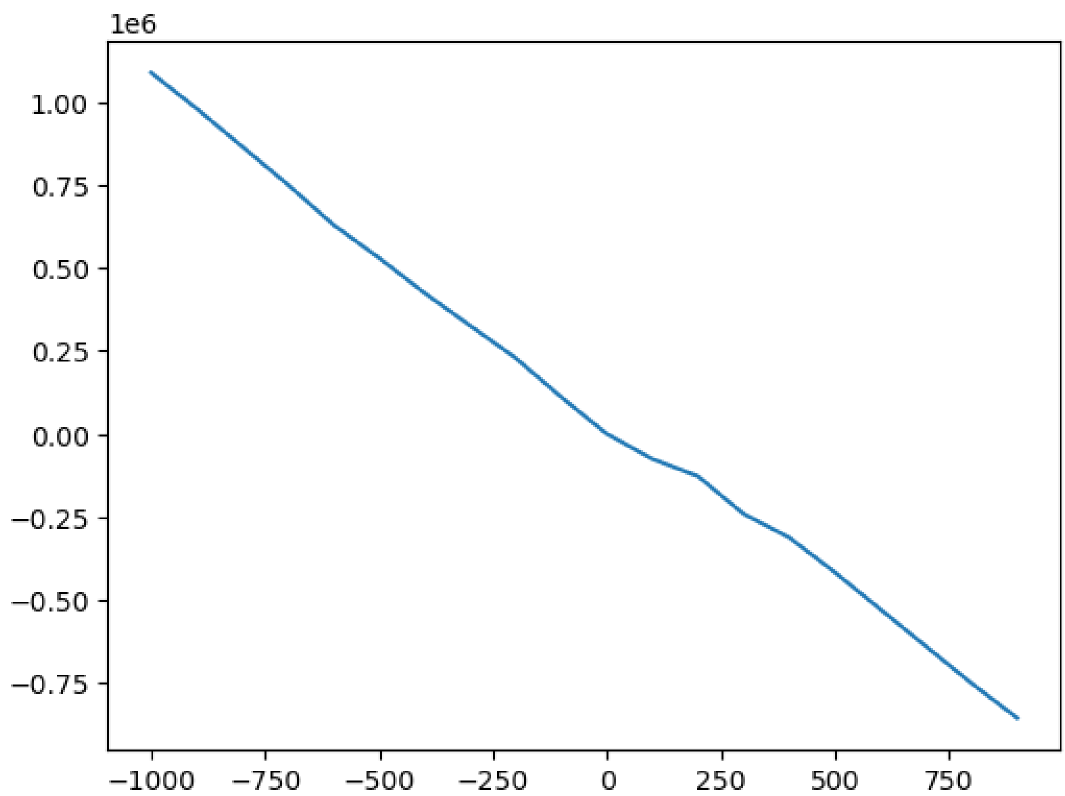
[24]:
```
profits = []

for raise_amount in range (-1000,1000,100):

    x_train_experiment = x_train.copy()
```

```
    x_train_experiment["MonthlyIncome"] = x_train_experiment["MonthlyIncome"] +␣
↪raise_amount

    y_pred = clf.predict(x_train)
    y_pred_experiment = clf.predict(x_train_experiment)

    diff = sum(y_pred-y_pred_experiment)
    print("change in attrition",diff)

    x_train_experiment["Y"] = y_pred
    x_train_experiment["Y_exp"] = y_pred_experiment

    x_train_experiment["RetChange"] = x_train_experiment["Y_exp"]-␣
↪x_train_experiment["Y"]
    sav = sum(-2.
↪8*x_train_experiment["RetChange"]*x_train_experiment["MonthlyIncome"])
    cost = len(x_train)*raise_amount

    print("Profit",sav- cost)
    profits.append(sav- cost)
```

```
change in attrition -16
Profit 1087584.4
change in attrition -14
Profit 979524.0
change in attrition -13
Profit 864992.8
change in attrition -12
Profit 750738.8
change in attrition -12
Profit 629778.8
change in attrition -9
Profit 530138.0
change in attrition -7
Profit 424200.0
change in attrition -4
Profit 326096.4
change in attrition -1
Profit 228440.8
change in attrition -1
Profit 110714.8
change in attrition 0
Profit 0.0
change in attrition 6
Profit -75328.40000000001
change in attrition 15
Profit -127503.60000000002
```

```
change in attrition 15
Profit -240914.8
change in attrition 21
Profit -311586.80000000005
change in attrition 22
Profit -416449.6000000001
change in attrition 22
Profit -527889.6000000001
change in attrition 22
Profit -639329.6000000001
change in attrition 22
Profit -750769.6000000001
change in attrition 23
Profit -854999.6000000001
```

[25]:
```
plt.plot(range(-1000,1000,100),profits)
plt.show()
```



Based on the result, when lowering people's income, the company might see an immediate increase in profit, but it could lead to increased attrition over time. When raising people's income, the immediate cost to the company also increases, which could lead to a decrease in profit. However, the profitability impact of these changes is influenced by the cost of the raises and the savings from reduced attrition. Initially, as income increases, the model predicts reduced attrition, but beyond a certain threshold, the additional cost of further raises leads to diminished profitability. I suggest company to optimize salary increases by using data analytics. The goal is to find the optimal range for salary increases that maximizes employee retention without disproportionately increasing costs.