**Homework #1 (Part B)**

# Neil Jiang, Hayley Huang, Foster Mosden, Icy Wang

Note: This is a team homework assignment. Discussing this homework with your classmates outside your MSBA team is a **violation** of the Honor Code. If you **borrow code** from somewhere else, please add a comment in your code to **make it clear** what the source of the code is (e.g., a URL would sufficient). If you borrow code and you don't provide the source, it is a violation of the Honor Code.

Total grade for Part 2: _____ out of ___50___ points

***ATTENTION: HW1 has two parts. Please first complete the Quiz "HW1_Part1" on Canvas. Then, proceed with Part 2 in the following page. You will need to submit (a) a PDF file with your answers and screenshots of Python code snippets as well as Rapidminer repositories and (b) the Python code and Rapidminer repositories.***

**(50 points) [Implement this exercise with both Python (30 points) RapidMiner(20 points). Use the decision tree classification technique on the *HW1* dataset. This dataset is provided on the course website and contains data about consumers and their decisions to terminate a contract (i.e., consumer churn problem).**

**Data description:**

| Col. | Var. Name | Var. Description |
|---|---|---|
| 1 | revenue | Mean monthly revenue in dollars |
| 2 | outcalls | Mean number of outbound voice calls |
| 3 | incalls | Mean number of inbound voice calls |
| 4 | months | Months in Service |
| 5 | eqpdays | Number of days the customer has had his/her current equipment |
| 6 | webcap | Handset is web capable |
| 7 | marryyes | Married (1=Yes; 0=No) |
| 8 | travel | Has traveled to non-US country (1=Yes; 0=No) |
| 9 | pcown | Owns a personal computer (1=Yes; 0=No) |
| 10 | creditcd | Possesses a credit card (1=Yes; 0=No) |
| 11 | retcalls | Number of calls previously made to retention team |
| 12 | churndep | Did the customer churn (1=Yes; 0=No) |

**30 Points (Python):**

Code mostly adapted from Python_BusinessAnalytics_DecisionTrees.ipynb by Prof.Todri
https://colab.research.google.com/drive/10aOYercpcwMVVGbnGMsRI9P76KM300T4?usp=sharing

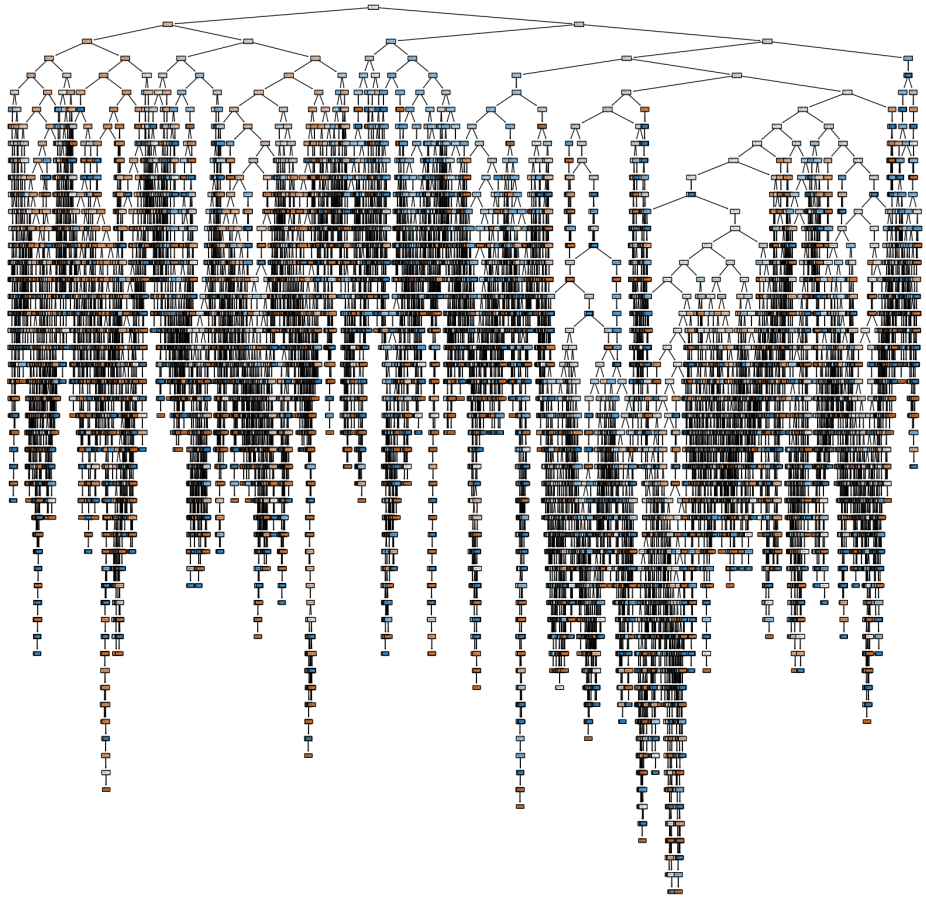
a) **(5 points) Build a decision tree model that predicts whether a consumer will terminate his/her contract. In particular, I would like for you to create a decision tree using entropy with no max depth. Some possible issues / hints to think about: using training vs. test datasets.**

```
def tree_targetting_churndep(data,criterion, t_size):
    ######################################## Split Training/Test Data ########################################
    # Split data into features (X) and the target variable (y)
    X = data.drop("churndep", axis=1)  # "churndep" is the target variable
    y = data["churndep"]

    # Split data into training and test sets (e.g., 70% train, 30% test)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=t_size, random_state=42)

    ######################################## Model Training ########################################
    # Create the decision tree classifier with no max depth and entropy as the criterion
    clf = DecisionTreeClassifier(criterion=criterion, random_state=42)

    # Fit the model to the training data
    clf.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = clf.predict(X_test)

    tree_depth = clf.get_depth()
    print("Decision Tree Depth:", tree_depth)

    # Calculate accuracy on the test set
    accuracy = accuracy_score(y_test, y_pred)

    # Calculate confusion matrix
    cnf_matrix = confusion_matrix(y_test, y_pred)

    # Calculate precision, recall, and F1-score
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Calculate error
    error = 1 - accuracy

    # Print the evaluation metrics
    print("Accuracy:", accuracy)
    print("Error:", error)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
    print("confusion matrix:")
    print(cnf_matrix)

    result_df = pd.DataFrame()
    result_df['y_test'] = y_test
    result_df['y_pred'] = y_pred

    return(result_df)
```

```
#decision tree model with no max depth, criterion = 'entropy',aplit_test_size = 0.3
result_df = tree_targetting_churndep(data,'entropy',0.3)
```

```
Decision Tree Depth: 56
Accuracy: 0.5307724513292862
Error: 0.4692275486707138
Precision: 0.5362782757148954
Recall: 0.521044992743106
F1 Score: 0.5285518982017037
confusion matrix:
[[2558 2173]
 [2310 2513]]
```

Visualizing decision tree model with no max depth – which is overfitted:



**b) (5 points) Explore how well the decision trees perform for <u>several different parameter values</u> (e.g., for different splitting criteria).**

(Details explained in part d)

b.1) exploring criterion and split ratio holding no max_depth

```
[14] #decision tree model with no max depth, criterion = 'entropy',aplit_test_size = 0.3
     result_df = tree_targetting_churndep(data,'entropy',0.3)
```

```
Decision Tree Depth: 56
Accuracy: 0.5307724513292862
Error: 0.4692275486707138
Precision: 0.5362782757148954
Recall: 0.521044992743106
F1 Score: 0.5285518982017037
confusion matrix:
[[2558 2173]
 [2310 2513]]
```

```
[15] result_df1 = tree_targetting_churndep(data,'gini',0.3)
```

```
Decision Tree Depth: 40
Accuracy: 0.5402972576931129
Error: 0.4597027423068871
Precision: 0.5453970929007794
Recall: 0.5368028198216878
F1 Score: 0.5410658307210031
confusion matrix:
[[2573 2158]
 [2234 2589]]
```

```
result_df2 = tree_targetting_churndep(data,'log_loss',0.3)
```

```
Decision Tree Depth: 56
Accuracy: 0.5307724513292862
Error: 0.4692275486707138
Precision: 0.5362782757148954
Recall: 0.521044992743106
F1 Score: 0.5285518982017037
confusion matrix:
[[2558 2173]
 [2310 2513]]
```

```
[18] result_df3 = tree_targetting_churndep(data,'gini',0.25)
```

```
Decision Tree Depth: 44
Accuracy: 0.5385001884185404
Error: 0.4614998115814596
Precision: 0.5432129514321296
Recall: 0.5424023874658045
F1 Score: 0.5428073668491787
confusion matrix:
[[2106 1834]
 [1840 2181]]
```

b.1) exploring different max_depth

```
from sklearn.model_selection import cross_val_score

# Create an array of different max_depth values to test
max_depth_values = range(1,20)

# Store the cross-validation scores for each max_depth value
cv_scores = []

X = data.drop("churndep", axis=1)  # "churndep" is the target variable
y = data["churndep"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=42)

for max_depth in max_depth_values:
    # Create and fit a decision tree classifier with the current max_depth value
    clf = DecisionTreeClassifier(criterion='entropy', random_state=42, max_depth=max_depth)

    # Perform cross-validation with 5 folds (adjust as needed)
    scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy')

    # Calculate the mean accuracy score across the folds
    mean_accuracy = scores.mean()

    # Store the mean accuracy score for this max_depth value
    cv_scores.append(mean_accuracy)

# Find the max_depth that gives the highest cross-validation accuracy
best_max_depth = max_depth_values[cv_scores.index(max(cv_scores))]

# Create and fit the final decision tree classifier with the best max_depth
final_clf = DecisionTreeClassifier(criterion='entropy', random_state=42, max_depth=best_max_depth)
final_clf.fit(X_train, y_train)

# Print the best max_depth
print("Best max_depth:", best_max_depth)

# Now you can use final_clf for predictions
```

```
Best max_depth: 4
```

```
def tree_targetting_churndep_wMax(data,t_size,criterion,max_depth):
    ######################################## Split Training/Test Data ########################################
    # Split  data into features (X) and the target variable (y)
    X = data.drop("churndep", axis=1)  # "churndep" is the target variable
    y = data["churndep"]

    # Split  data into training and test sets (e.g., 70% train, 30% test)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=t_size, random_state=42)

    ######################################## Model Training ########################################
    # Create the decision tree classifier with no max depth and entropy as the criterion
    clf = DecisionTreeClassifier(criterion=criterion, random_state=42, max_depth=max_depth)

    # Fit the model to the training data
    clf.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = clf.predict(X_test)

    tree_depth = clf.get_depth()
    print("Decision Tree Depth:", tree_depth)

    # Calculate accuracy on the test set
    accuracy = accuracy_score(y_test, y_pred)

    # Calculate confusion matrix
    cnf_matrix = confusion_matrix(y_test, y_pred)

    # Calculate precision, recall, and F1-score
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Calculate error
    error = 1 - accuracy

    # Print the evaluation metrics
    print("Accuracy:", accuracy)
    print("Error:", error)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
    print("confusion matrix:")
    print(cnf_matrix)

    result_df = pd.DataFrame()
    result_df['y_test'] = y_test
    result_df['y_pred'] = y_pred

    return(result_df)
```

When setting the max_depth to 4, different criterion agrees on the same model:

```
result_df6 = tree_targetting_churndep_wMax(data,0.3,'entropy',4)
```

```
Decision Tree Depth: 4
Accuracy: 0.5958760728490684
Error: 0.4041239271509316
Precision: 0.5765436028007639
Recall: 0.7511922040223927
F1 Score: 0.6523813811110111
confusion matrix:
[[2070 2661]
 [1200 3623]]
```

```
result_df7 = tree_targetting_churndep_wMax(data,0.3,'log_loss',4)
```

```
Decision Tree Depth: 4
Accuracy: 0.5958760728490684
Error: 0.4041239271509316
Precision: 0.5765436028007639
Recall: 0.7511922040223927
F1 Score: 0.6523813811110111
confusion matrix:
[[2070 2661]
 [1200 3623]]
```

```
result_df8 = tree_targetting_churndep_wMax(data,0.3,'gini',4)
```

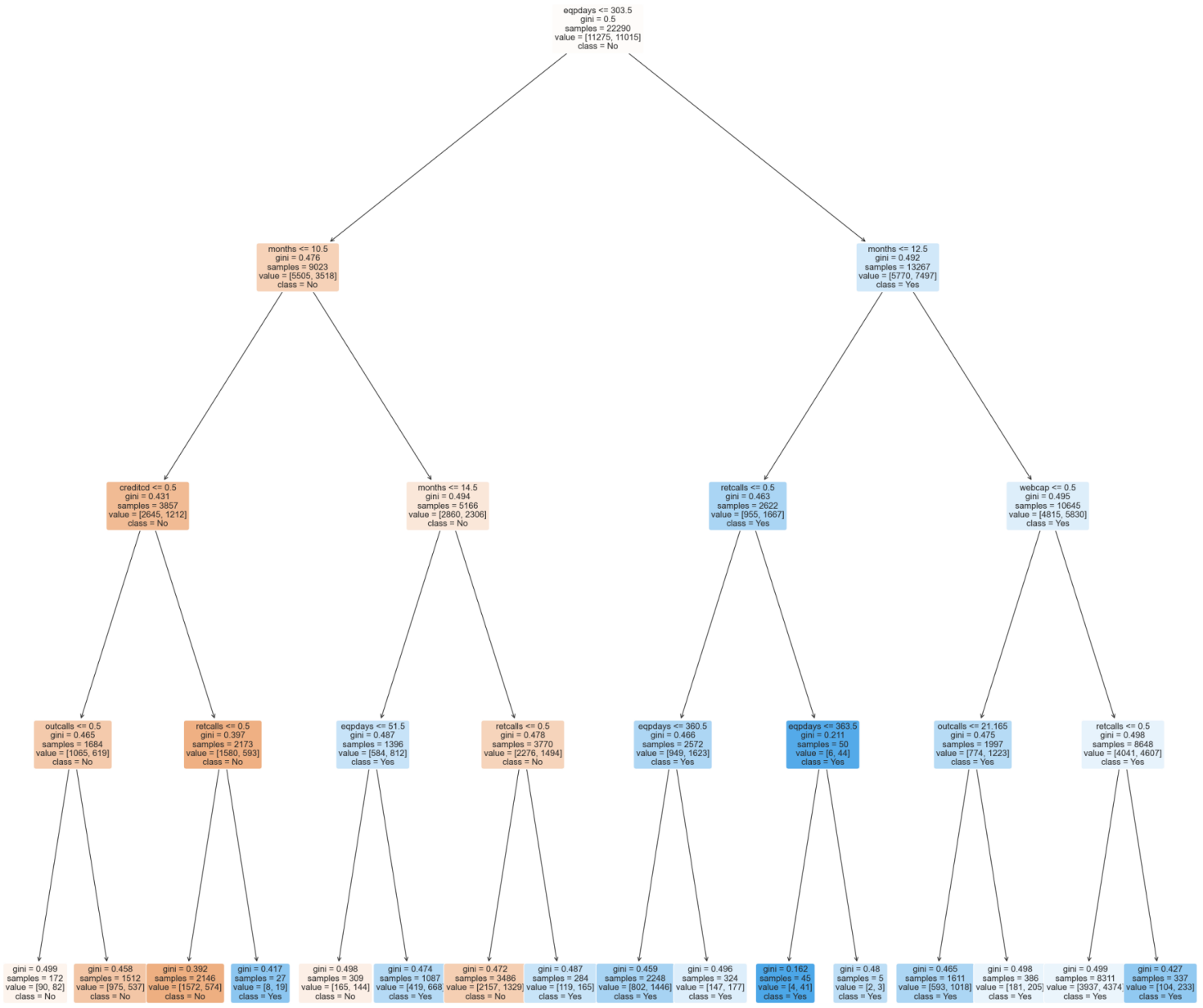
```
Decision Tree Depth: 4
Accuracy: 0.5958760728490684
Error: 0.4041239271509316
Precision: 0.5765436028007639
Recall: 0.7511922040223927
F1 Score: 0.6523813811110111
confusion matrix:
[[2070 2661]
 [1200 3623]]
```

**c) (5 points) Discuss the model (decision tree) that provides the best predictive performance from experimenting with different parameter values in question (b).**

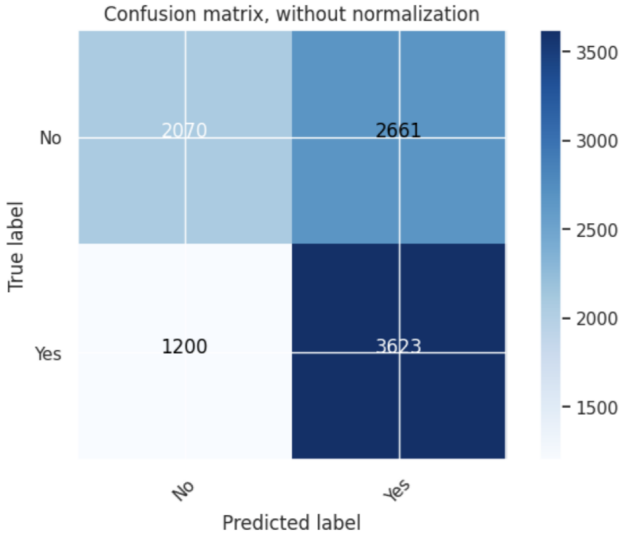
```
result_df8 = tree_targetting_churndep_wMax(data,0.3,'gini',4)
```

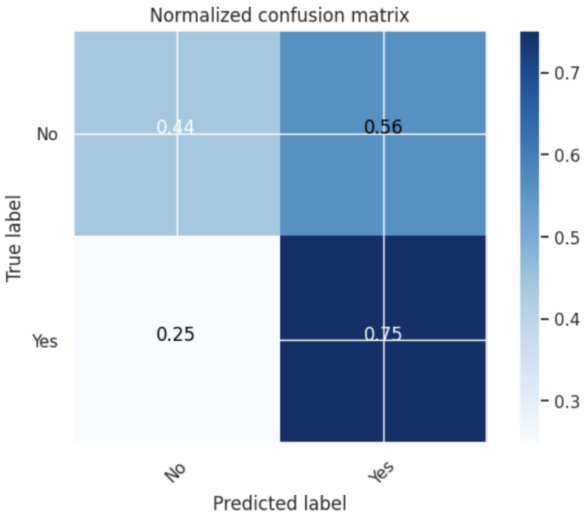
Targeting [churndep] the consumers' decisions to terminate a contract (i.e., Did the customer churn), our model with the best predictive performance has a maximum depth of 4, which limits its complexity but makes it easier to interpret. The most significant attributes in our tree includes [months] Months in Service, [creditcd] Possesses a credit card or not, [retcalls] Number of calls previously made to retention team, and [webcap] Handset is web capable or not. See bellow for a detailed interpretation of its predictive performance:

- Accuracy: ~0.596: About 59.6% of all predictions were correct. This is better than random guessing but may not be sufficient depending on the problem's criticality and the industry standard.
- Error: ~0.404: About 40.4% of predictions were incorrect. This is the complement of accuracy.
- Precision: ~0.577: Of the instances predicted as churn, about 57.7% were actually churn. This might be an important metric if false positives have a high cost.
- Recall: ~0.751: Of the actual churn instances, about 75.1% were caught by the model. This is fairly good and important if the cost of false negatives is high.
- F1 Score: ~0.652: The F1 Score is the harmonic mean of Precision and Recall and takes both false positives and false negatives into account. An F1 Score of around 0.652 suggests that the model is reasonably balanced in terms of precision and recall.


eqpdays <= 303.5
gini = 0.5
samples = 22290
value = [11275, 11015]
class = No
months <= 10.5
gini = 0.476
samples = 9023
value = [5505, 3518]
class = No
months <= 12.5
gini = 0.492
samples = 13267
value = [5770, 7497]
class = Yes
creditcd <= 0.5
gini = 0.431
samples = 3857
value = [2645, 1212]
class = No
months <= 14.5
gini = 0.494
samples = 5166
value = [2860, 2306]
class = No
retcalls <= 0.5
gini = 0.463
samples = 2622
value = [955, 1667]
class = Yes
webcap <= 0.5
gini = 0.495
samples = 10645
value = [4815, 5830]
class = Yes
outcalls <= 0.5
gini = 0.465
samples = 1684
value = [1065, 619]
class = No
retcalls <= 0.5
gini = 0.397
samples = 2173
value = [1580, 593]
class = No
eqpdays <= 51.5
gini = 0.487
samples = 1396
value = [584, 812]
class = Yes
retcalls <= 0.5
gini = 0.478
samples = 3770
value = [2276, 1494]
class = No
eqpdays <= 360.5
gini = 0.466
samples = 2572
value = [949, 1623]
class = Yes
eqpdays <= 363.5
gini = 0.211
samples = 50
value = [6, 44]
class = Yes
outcalls <= 21.165
gini = 0.475
samples = 1997
value = [774, 1223]
class = Yes
retcalls <= 0.5
gini = 0.498
samples = 8648
value = [4041, 4607]
class = Yes
gini = 0.499
samples = 172
value = [90, 82]
class = No
gini = 0.458
samples = 1512
value = [975, 537]
class = No
gini = 0.392
samples = 2146
value = [1572, 574]
class = No
gini = 0.417
samples = 27
value = [8, 19]
class = Yes
gini = 0.498
samples = 309
value = [165, 144]
class = No
gini = 0.474
samples = 1087
value = [419, 668]
class = Yes
gini = 0.472
samples = 3486
value = [2157, 1329]
class = No
gini = 0.487
samples = 284
value = [119, 165]
class = Yes
gini = 0.459
samples = 2248
value = [802, 1446]
class = Yes
gini = 0.496
samples = 324
value = [147, 177]
class = Yes
gini = 0.162
samples = 45
value = [4, 41]
class = Yes
gini = 0.48
samples = 5
value = [2, 3]
class = Yes
gini = 0.465
samples = 1611
value = [593, 1018]
class = Yes
gini = 0.498
samples = 386
value = [181, 205]
class = Yes
gini = 0.499
samples = 8311
value = [3937, 4374]
class = Yes
gini = 0.427
samples = 337
value = [104, 233]
class = Yes

Decision Tree Depth: 4 Accuracy: 0.5958760728490684 Error: 0.4041239271509316 Precision: 0.5765436028007639 Recall: 0.7511922040223927 F1 Score: 0.6523813811110111



**d) (15 points) Present a brief overview of your predictive modeling process, explorations, and discuss your results. That is, you need to lay out the steps you have taken in order to build and evaluate the decision tree model. For instance, how did you explore the data set before you built the model? Write this report in a way that the upper level management of the team would understand what you are doing. Why is the decision tree an appropriate model for this problem? How can we evaluate the predictive ability of the decision tree? If you build decision trees with different splitting criteria, which decision tree would you prefer to use in practice? Make sure you present information about the model "goodness" (please report the confusion matrix, predictive accuracy, classification error, precision, recall, f-measure).**

**D.** *Present a brief overview of your predictive modeling process, explorations, and discuss your results. That is, you need to lay out the steps you have taken in order to build and evaluate the decision tree model.*

**D1. How did you explore the data set before you built the model? Write this report in a way that the upper level management of the team would understand what you are doing.**

1. To explore the data, we reviewed the first 15 rows of the data, reviewed the column names and data types, and got stats for each column of the data.
   1.1. `data.head(15)`: Provided the first 15 rows of data. From this we learned that *Revenue* is formatted as a dollar amount rounded to the second decimal place; *Outcalls* and *incalls* are formatted as a decimal rounded to the second decimal place; *Months, EQPDays*, and *retcalls* are all stored as integers; *WebCap, marryyes, travel, pcown, creditcd,* and *churndep* are all stored as integers, but are actually true/false values wherein 1 = true and 0 = false.
   1.2. `data.info()`: Provided the number of rows, columns, data type for each column, and the number of non-null entries present in each column. Most importantly, we learned from this command that there are no null values in any of our columns of the dataset. This means that every data point we have is reliable and complete, reducing the risk of errors or misleading results. There's no need to impute or guess missing information, and we can make confident and effective decision-making.
   1.3. `data.describe()`: Provided the count of entries, mean of all values, standard deviation, minimum value, 25th percentile, median (50th percentile), 75th percentile, and maximum value for every column in the dataset.

```
data.describe()
```

| | revenue | outcalls | incalls | months | eqpdays | webcap | marryyes | travel | pcown | creditcd | ret |
|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 31891.000000 | 31891.000000 | 31891.000000 | 31891.000000 | 31891.000000 | 31891.000000 | 31891.000000 | 31891.000000 | 31891.000000 | 31891.000000 | 31891.00( |
| mean | 58.665179 | 24.951385 | 8.065277 | 18.761908 | 391.222633 | 0.894704 | 0.363175 | 0.057163 | 0.184817 | 0.676931 | 0.04 |
| std | 44.163859 | 34.790147 | 16.610589 | 9.548019 | 254.998478 | 0.306939 | 0.480922 | 0.232158 | 0.388155 | 0.467656 | 0.22 |
| min | -5.860000 | 0.000000 | 0.000000 | 6.000000 | -5.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00( |
| 25% | 33.450000 | 3.000000 | 0.000000 | 11.000000 | 212.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00( |
| 50% | 48.380000 | 13.330000 | 2.000000 | 17.000000 | 341.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.00( |
| 75% | 71.040000 | 33.330000 | 9.000000 | 24.000000 | 530.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 0.00( |
| max | 861.110000 | 610.330000 | 404.000000 | 60.000000 | 1812.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 4.00( |

   1.3.1. This confirmed to us that our true/false columns are being correctly utilized, as the min = 0 and max = 1 for each of these columns. There is no erroneous data, where the stored value is anything other than 0 or 1, so we can confidently use these columns for modeling without pre-processing them.
   1.3.2. Perhaps the most important to our analysis was the observation from this function that the minimum value for *revenue* was -5.86, and the minimum value for *eqpdays* was -5. This brought to our attention that there are negative values in *revenue* and *eqpdays*, columns where we would not expect to see negative values. The negative *revenue* values are likely an

error and/or the result of dirty data. In addition, it's not possible for a customer to have had their equipment for a negative quantity of days.

1.3.3. Observing these abnormal negative values informed us that we would need to drop any rows from the dataset where *revenue* or *eqpdays* was less than 0. There were less than 50 rows with one of these anomalies, so we found it best to simply drop the rows from our table. The result of dropping these rows is a loss of less than 0.15% of our initial dataset. Our alternative option was to impute or guess values for these rows. However, given the minimal data loss by deletion, and the fact that our data was otherwise complete (no null values), we felt it was best to keep the data clean by simply dropping these rows.

**D2.Why is the decision tree an appropriate model for this problem?**

2. A decision tree makes for a great model for predicting whether a customer will terminate their contract. Some reasons why are:
   2.1. **Ease of Interpretation:** Each branch and node in our decision tree will represent a decision being made on the basis of features of each customer. The categorization and decisions represented by the tree are easily understood.
   2.2. **Visual Ranking of Importance:** Decision trees highlight the most influential factors that contribute to the customers terminating their contract. This is extremely useful if the business is seeking to reduce customer churn, as it provides them with a visual ranking of the importance of each attribute in predicting whether or not a customer will terminate their contract. The company could use this data to give more attention to certain aspects of their business in hopes to reduce churn.
   2.3. **Non-Linear Modeling:** In many cases, the customer behavior is influenced by multiple factors which interact in intricate ways that cannot be explained by a linear model like Linear Regression or Logistic Regression. Our decision tree can effectively model and display these relationships through categorization via decisions.
   **2.4.** **Data Types:** Decision trees can effectively handle both our categorical (true/false features) and numerical (revenue, etc.) data when building a model. This makes the decision tree optimal for predicting churn, as our dataset has a mix of attribute types.
   2.5. **Implementation:** Requires minimal data pre-processing to build, which aids in interpretability of the final model as well. It's much easier to interpret the final model when the values that decisions are being based on represent the real-world values initially found in the dataset.

**D3. How can we evaluate the predictive ability of the decision tree?**

3.1 For quick evaluation of predictive ability when exploring around parameters, we used predictive accuracy, classification error, precision, recall, f-measure, and confusion matrix available in sklearn.metrics:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

- **Accuracy:** Accuracy measures the overall correctness of a model's predictions. It's the ratio of correctly predicted instances to the total number of instances.
- **Precision:** Precision measures the proportion of true positive predictions (correctly predicted positive cases) out of all positive predictions made by the model.
- **Recall (Sensitivity):** Recall measures the proportion of true positive predictions out of all actual positive cases. It's useful when you want to minimize false negatives.
- **F1-Score:** The F1-score is the harmonic mean of precision and recall. It balances both precision and recall and provides a single metric for evaluating model performance.
- **Confusion Matrix:** The confusion matrix provides a detailed view of true positives, true negatives, false positives, and false negatives.

```python
# Calculate accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)

# Calculate confusion matrix
cnf_matrix = confusion_matrix(y_test, y_pred)

# Calculate precision, recall, and F1-score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Calculate error
error = 1 - accuracy

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Error:", error)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("confusion matrix:")
print(cnf_matrix)
```

```
[8] #decision tree model with no max depth, criterion = 'entropy',aplit_test_size = 0.3
    result_df = tree_targetting_churndep(data,'entropy',0.3)

    Decision Tree Depth: 56
    Accuracy: 0.5307724513292862
    Error: 0.4692275486707138
    Precision: 0.5362782757148954
    Recall: 0.521044992743106
    F1 Score: 0.5285518982017037
    confusion matrix:
    [[2558 2173]
     [2310 2513]]

[9] result_df1 = tree_targetting_churndep(data,'gini',0.3)

    Decision Tree Depth: 40
    Accuracy: 0.5402972576931129
    Error: 0.4597027423068871
    Precision: 0.5453970929007794
    Recall: 0.5368028198216878
    F1 Score: 0.5410658307210031
    confusion matrix:
    [[2573 2158]
     [2234 2589]]

    result_df2 = tree_targetting_churndep(data,'log_loss',0.3)

    Decision Tree Depth: 56
    Accuracy: 0.5307724513292862
    Error: 0.4692275486707138
    Precision: 0.5362782757148954
    Recall: 0.521044992743106
    F1 Score: 0.5285518982017037
    confusion matrix:
    [[2558 2173]
     [2310 2513]]

    result_df3 = tree_targetting_churndep(data,'gini',0.25)

    Decision Tree Depth: 44
    Accuracy: 0.5385001884185404
    Error: 0.4614998115814596
    Precision: 0.5432129514321296
    Recall: 0.5424023874658045
    F1 Score: 0.5428073668491787
    confusion matrix:
    [[2106 1834]
     [1840 2181]]
```

3.2 We used more advanced evaluation of predictive ability when finalizing the model to use:

3.2.1 when choosing the best max_depth parameter (which we found highly influential) – we performed **cross-validation with 5 folds** using cross_val_score() from sklearn.model_selection.

Cross-validation is a method that assesses a model's performance by splitting the data into subsets, training the model on one subset, and testing it on another. This cycle repeats several times with different subsets, and the model's overall performance is computed as the average across these iterations.

We used the most common type of cross-validation k-fold cross-validation (5-fold), where the data is divided into k equally-sized subsets. The model is trained on k-1 of these subsets and tested on the remaining one. This process repeats k times, each time using a different subset for testing. The model's performance is then determined as the average across all k iterations.

Using cross-validation for max_depth testing helps **reduce the chance of over-fitting**.

```
from sklearn.model_selection import cross_val_score

# Create an array of different max_depth values to test
max_depth_values = range(1,20)

# Store the cross-validation scores for each max_depth value
cv_scores = []

X = data.drop("churndep", axis=1)  # "churndep" is the target variable
y = data["churndep"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=42)

for max_depth in max_depth_values:
    # Create and fit a decision tree classifier with the current max_depth value
    clf = DecisionTreeClassifier( random_state=42, max_depth=max_depth)

    # Perform cross-validation with 5 folds (adjust as needed)
    scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy')

    # Calculate the mean accuracy score across the folds
    mean_accuracy = scores.mean()

    # Store the mean accuracy score for this max_depth value
    cv_scores.append(mean_accuracy)

# Find the max_depth that gives the highest cross-validation accuracy
best_max_depth = max_depth_values[cv_scores.index(max(cv_scores))]

# Print the best max_depth
print("Best max_depth:", best_max_depth)
```

```
Best max_depth: 4
```

**D4. If you build decision trees with different splitting criteria, which decision tree would you prefer to use in practice? Make sure you present information about the model "goodness" (please report the confusion matrix, predictive accuracy, classification error, precision, recall, f-measure).**

Splitting criteria in decision trees refer to the metric used to decide how to split a node into two or more child nodes. This decision aims to increase purity or homogeneity in each child node for the target variable. Common splitting criteria are:

a. Gini Impurity: Measures how often a randomly chosen sample would be incorrectly labeled.
b. Information Gain: Based on the concept of entropy, measures the reduction in uncertainty.
c. Chi-Square: Tests the statistical significance of differences between sub-nodes and parent node.

In the above steps, we used the information gain method to build the decision tree. However, each splitting method has its relative pros and cons. Here is the comparison of each method:

a. Gini Impurity measures the disorder or impurity of a set. A Gini Impurity of 0 means all elements are of the same class. Advantages: Fast to compute. Good for continuous attributes. Disadvantages: May produce slightly biased trees for imbalanced classes.
b. Information Gain (IG) measures the effectiveness of an attribute in reducing uncertainty or entropy. High IG indicates better splitting attributes. Advantages: Good for handling missing values. Less

biased towards multi-level attributes compared to Gini. Disadvantages: Computationally intensive. May overfit with noisy data.

c. Chi-Square tests the statistical significance of observed distribution vs expected distribution in the case of a split. Advantages: Provides statistical inference (p-value). Good for categorical variables. Disadvantages: Not suitable for continuous variables without binning. May require a sufficiently large dataset for reliable p-values.

Therefore, in practice, we prefer the Gini Impurity method because the Gini Impurity method works well with continuous variables. In our dataset, most variables are continuous variables. In terms of the remaining variables that are boolean data types, boolean values are also straightforward to the Gini Impurity method because there are only two classes.

After changing the method from Information Gain method to Gini Impurity, we have then conducted the model "goodness" assessment for the new method.

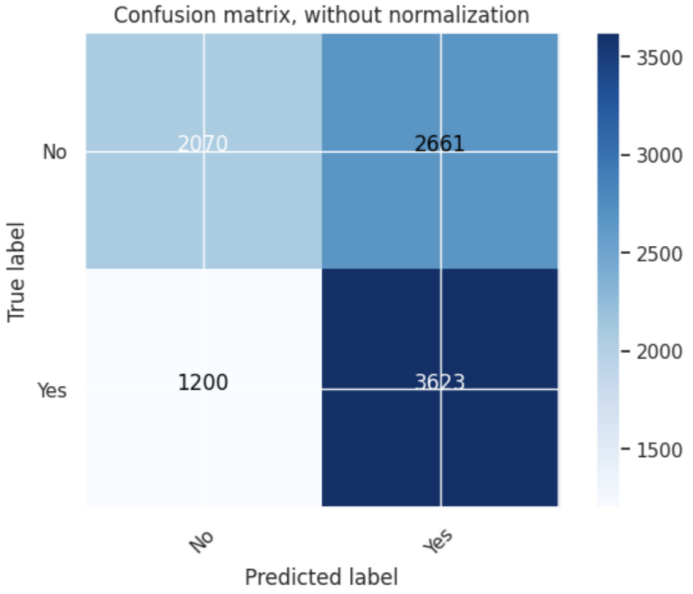
```
from sklearn.model_selection import cross_val_score

# Create an array of different max_depth values to test
max_depth_values = range(1,20)

# Store the cross-validation scores for each max_depth value
cv_scores = []

X = data.drop("churndep", axis=1)  # "churndep" is the target variable
y = data["churndep"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=42)

for max_depth in max_depth_values:
    # Create and fit a decision tree classifier with the current max_depth value
    clf = DecisionTreeClassifier(criterion='gini', random_state=42, max_depth=max_depth)

    # Perform cross-validation with 5 folds (adjust as needed)
    scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy')

    # Calculate the mean accuracy score across the folds
    mean_accuracy = scores.mean()

    # Store the mean accuracy score for this max_depth value
    cv_scores.append(mean_accuracy)

# Find the max_depth that gives the highest cross-validation accuracy
best_max_depth = max_depth_values[cv_scores.index(max(cv_scores))]

# Print the best max_depth
print("Best max_depth:", best_max_depth)

# Now you can use final_clf for predictions
```

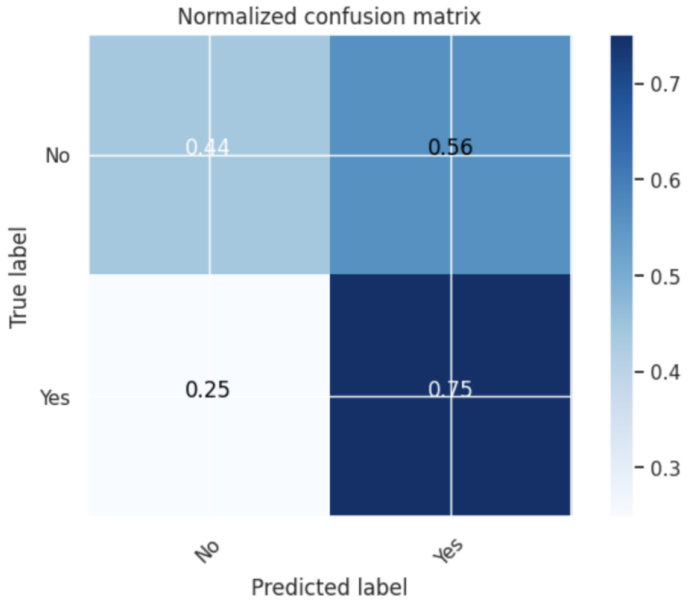
```
Best max_depth: 4
```

Using the Gini method, the best max depth is 4.

```
result_df8 = tree_targetting_churndep_wMax(data,0.3,'gini',4)
```

Decision Tree Depth: 4 Accuracy: 0.5958760728490684 Error: 0.4041239271509316 Precision: 0.5765436028007639 Recall: 0.7511922040223927 F1 Score: 0.6523813811110111



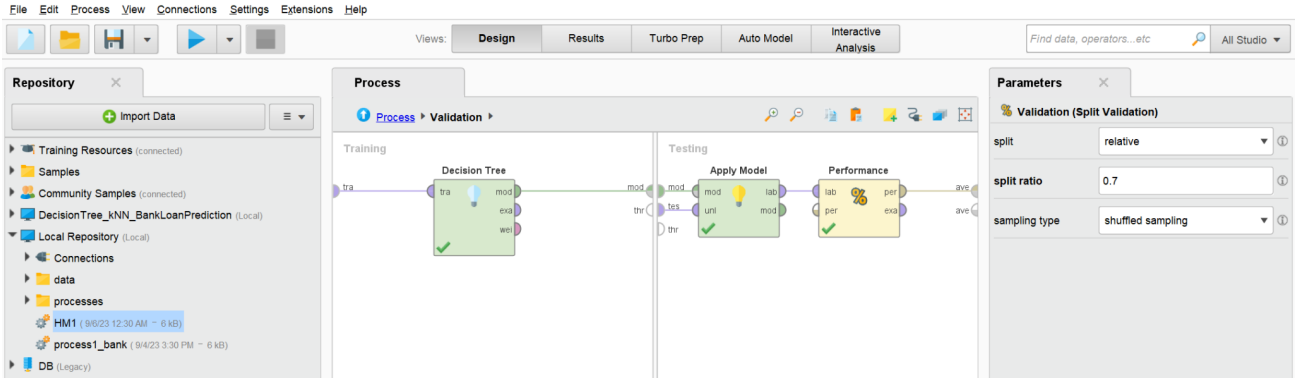
**20 Points (Rapidminer):**

**As you discuss the results please make sure you provide screenshots of your corresponding Python code at the same time. At the end, also please provide the Rapidminer screenshots as well (Screenshot on how you split the data, how you built and evaluated the model, the Parameters panel for the Decision tree operator, all the corresponding performance metrics as well as the visualization of the decision tree).**

**A. For the main model(decision tree using entropy with no max depth):**

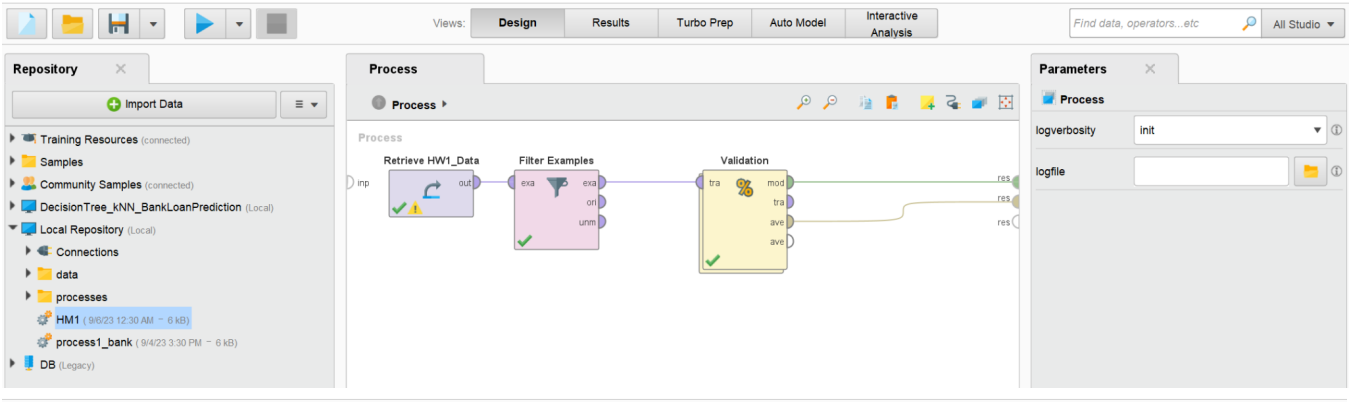
**a. Splitting data**

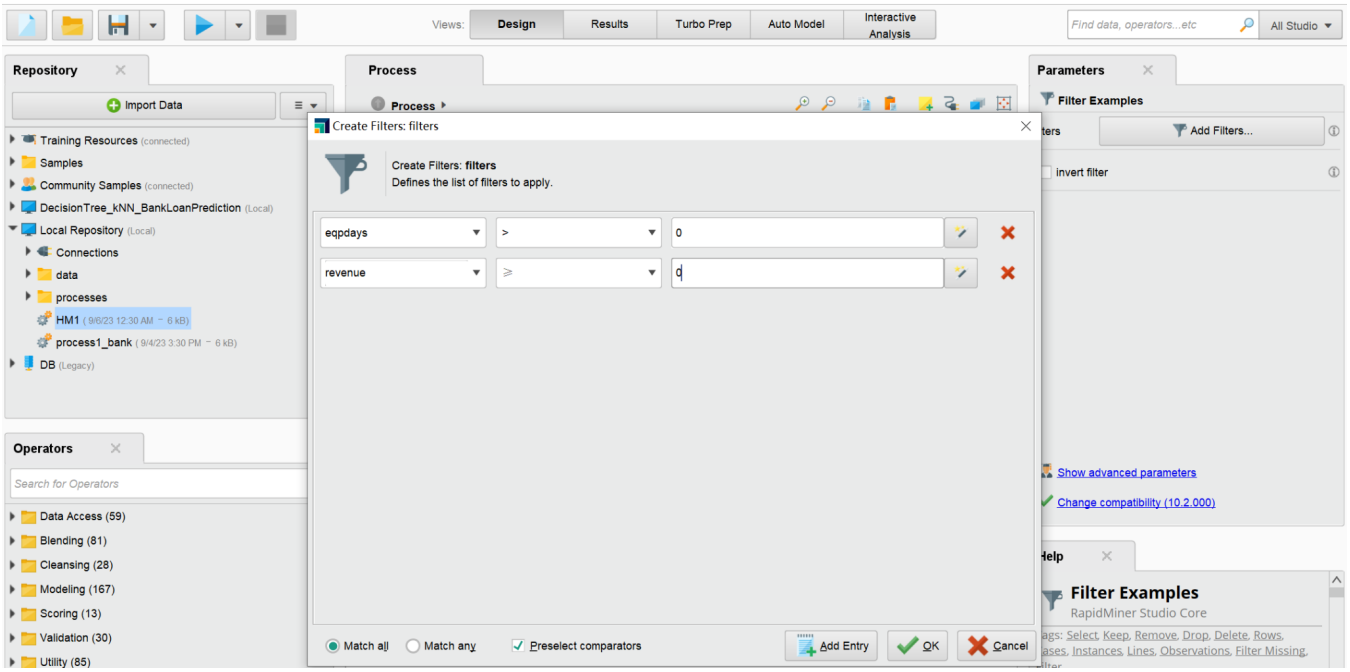
- Apply "Split Validation" operator
- Set the split ratio to the default "0.7"
- Choose sampling type as "shuffled sampling"




**b. Building and evaluating the model**

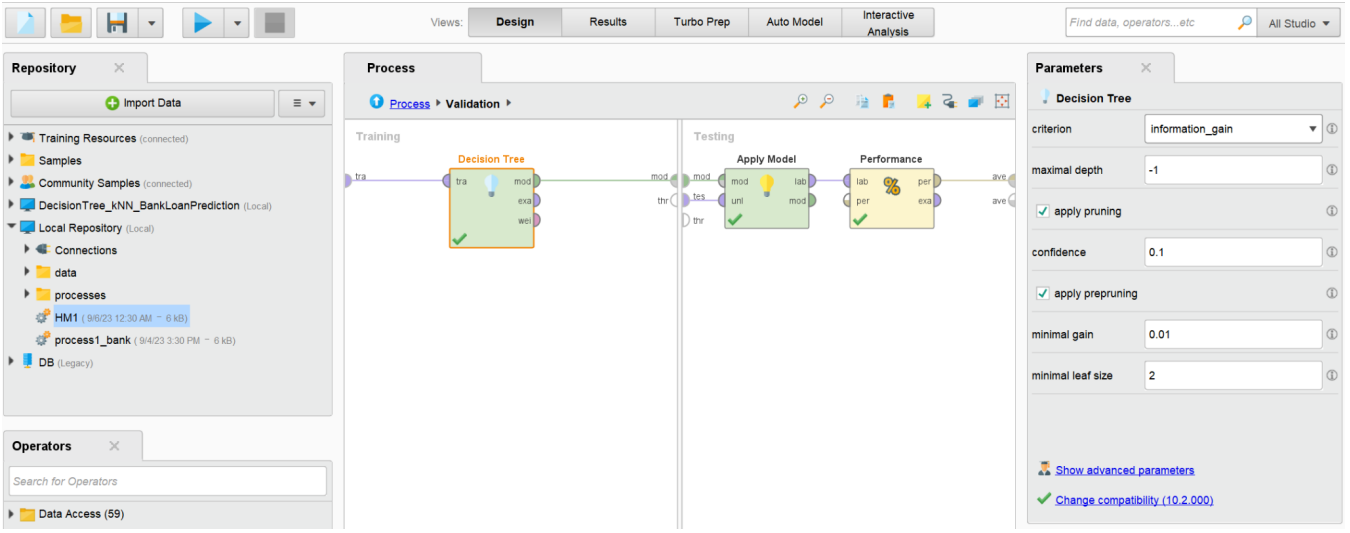
- Retrieve Data
- Apply "Filter Examples" operator, filtering out the negative values in the ***eqpdays*** and ***revenue*** columns
- Apply "Split Validation" operator




**c. Parameters panel for the Decision tree operator**

- **For crition, I chose information_gain**
- **For the main model, choose no max depth, set the maximal depth to "-1"**
- **Other Parameters leave to default**




**d. Corresponding performance metrics**

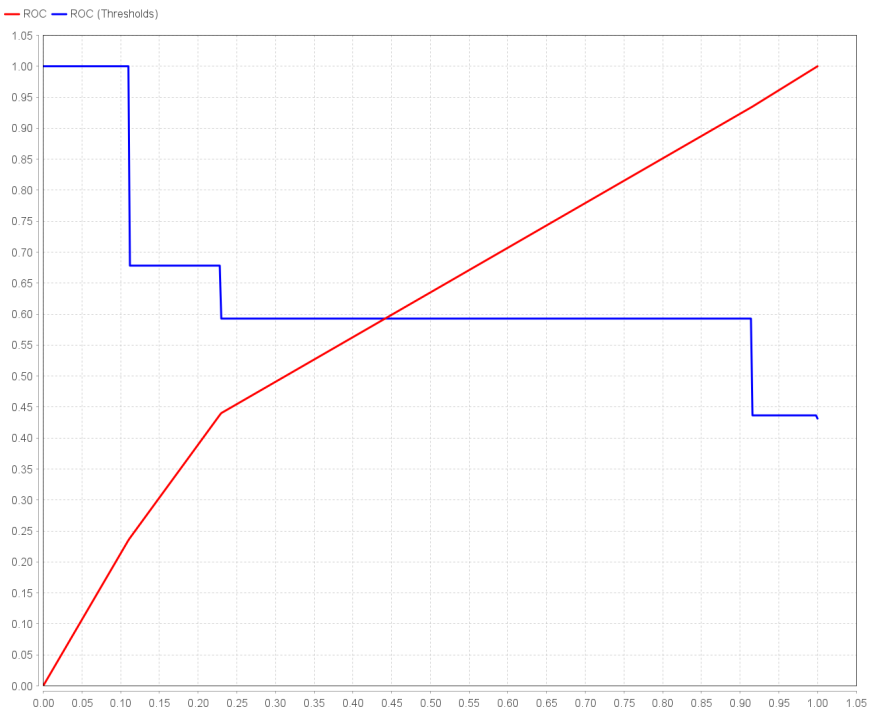
## accuracy: 60.43%

**accuracy: 60.43%**

| | true 1 | true 0 | class precision |
|---|---|---|---|
| pred. 1 | 3659 | 2689 | 57.64% |
| pred. 0 | 1091 | 2113 | 65.95% |
| class recall | 77.03% | 44.00% | |

## AUC: 0.606

**AUC: 0.606 (positive class: 0)**




## precision: 65.95%

**precision: 65.95% (positive class: 0)**

| | true 1 | true 0 | class precision |
|---|---|---|---|
| pred. 1 | 3659 | 2689 | 57.64% |
| pred. 0 | 1091 | 2113 | 65.95% |
| class recall | 77.03% | 44.00% | |

## recall: 44.00%

**recall: 44.00% (positive class: 0)**

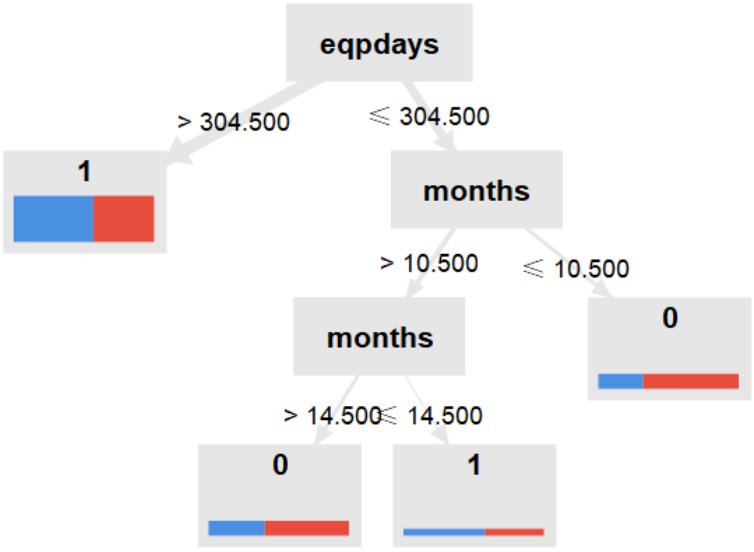
| | true 1 | true 0 | class precision |
|---|---|---|---|
| pred. 1 | 3659 | 2689 | 57.64% |
| pred. 0 | 1091 | 2113 | 65.95% |
| class recall | 77.03% | 44.00% | |

## f-measure: 52.79%

**f_measure: 52.79% (positive class: 0)**

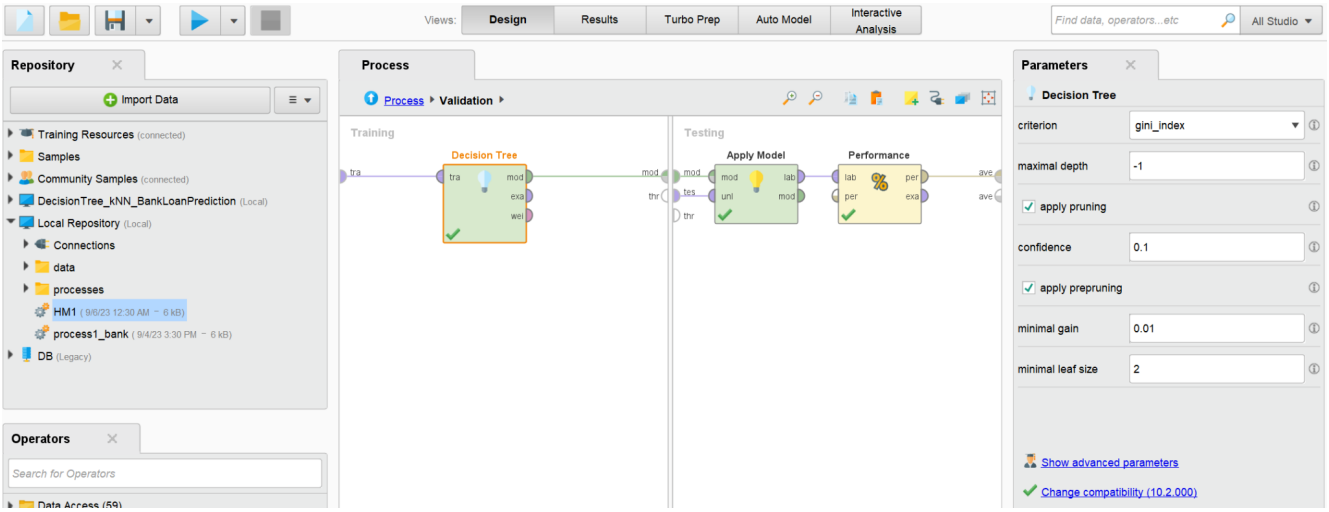
| | true 1 | true 0 | class precision |
|---|---|---|---|
| pred. 1 | 3659 | 2689 | 57.64% |
| pred. 0 | 1091 | 2113 | 65.95% |
| class recall | 77.03% | 44.00% | |

**e. Visualization of the decision tree**




**B. Explore how well the decision trees perform for different parameter values**

**a. change decision tree criterion to "gini_index", holding other parameters still**




**Results:**

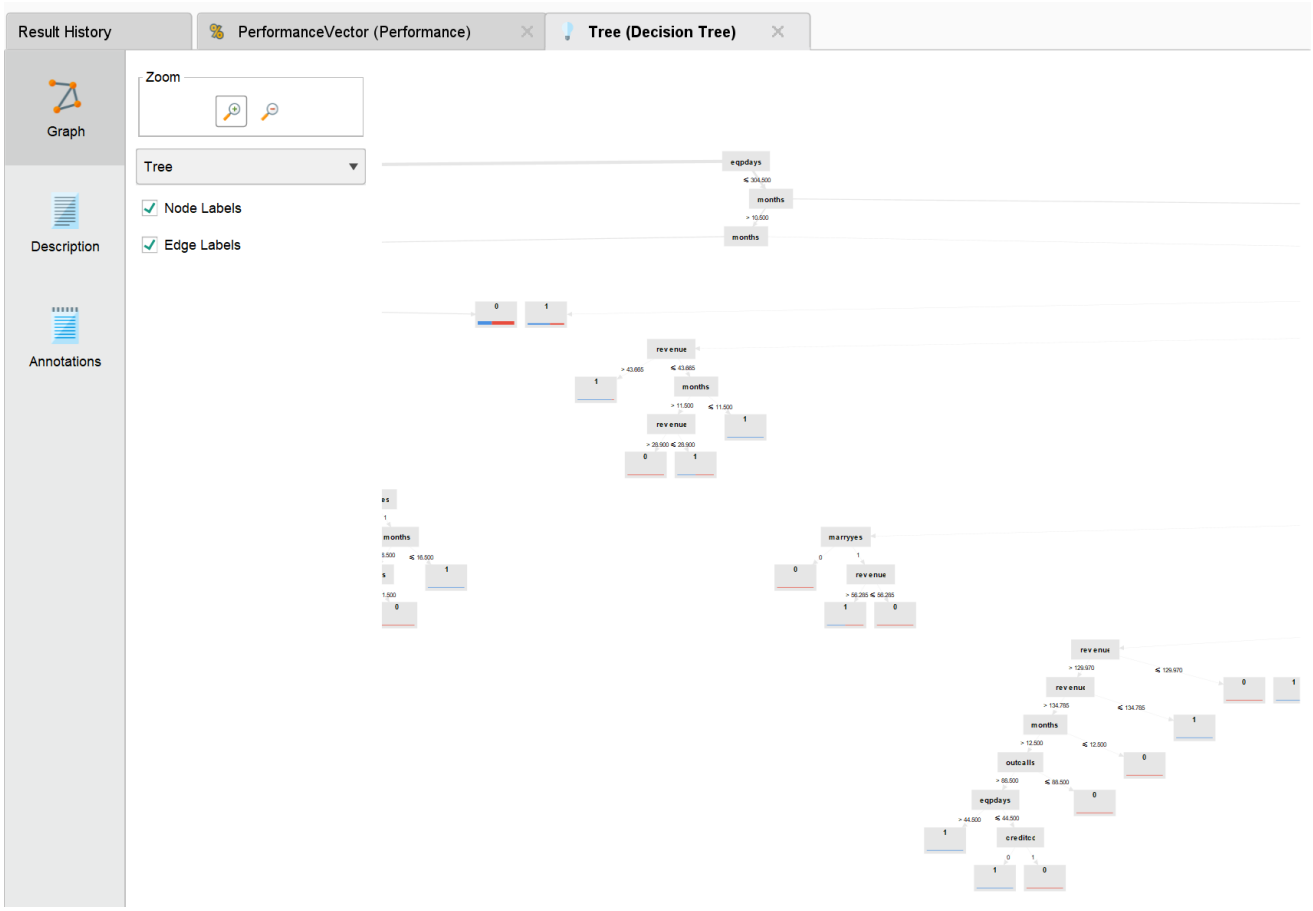
Comparing to main model,
the model accuracy increased, from 60.43% to 60.56%,
the f-measure increased as well, from 52.79% to 53.12%,
the decision tree has much more branches than the main model.

**accuracy: 60.56%**

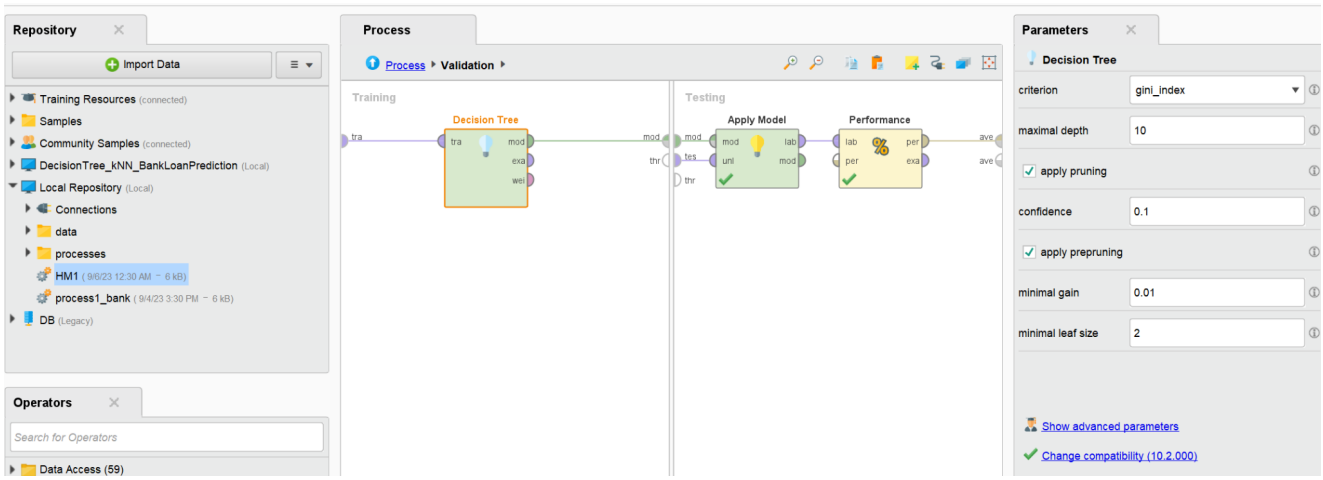
| | true 1 | true 0 | class precision |
|---|---|---|---|
| pred. 1 | 3651 | 2668 | 57.78% |
| pred. 0 | 1099 | 2134 | 66.01% |
| class recall | 76.86% | 44.44% | |

**f_measure: 53.12% (positive class: 0)**

| | true 1 | true 0 | class precision |
|---|---|---|---|
| pred. 1 | 3651 | 2668 | 57.78% |
| pred. 0 | 1099 | 2134 | 66.01% |
| class recall | 76.86% | 44.44% | |



**b. change decision tree criterion to “gini_index”, and change max depth to “10”, holding other parameters still**




**Results:**

Comparing to main model,
the model accuracy increased, from 60.43% to 60.7%,
the f-measure increased as well, from 52.79% to 53.45%.

**accuracy: 60.70%**

| | true 1 | true 0 | class precision |
|---|---|---|---|
| pred. 1 | 3643 | 2647 | 57.92% |
| pred. 0 | 1107 | 2155 | 66.06% |
| class recall | 76.69% | 44.88% | |

**f_measure: 53.45% (positive class: 0)**

| | true 1 | true 0 | class precision |
|---|---|---|---|
| pred. 1 | 3643 | 2647 | 57.92% |
| pred. 0 | 1107 | 2155 | 66.06% |
| class recall | 76.69% | 44.88% | |