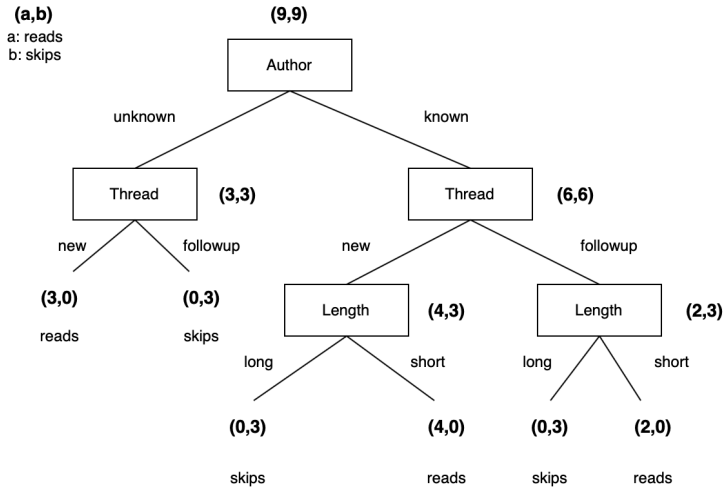
Z5142003 Tong Zheng

Question 1

a)

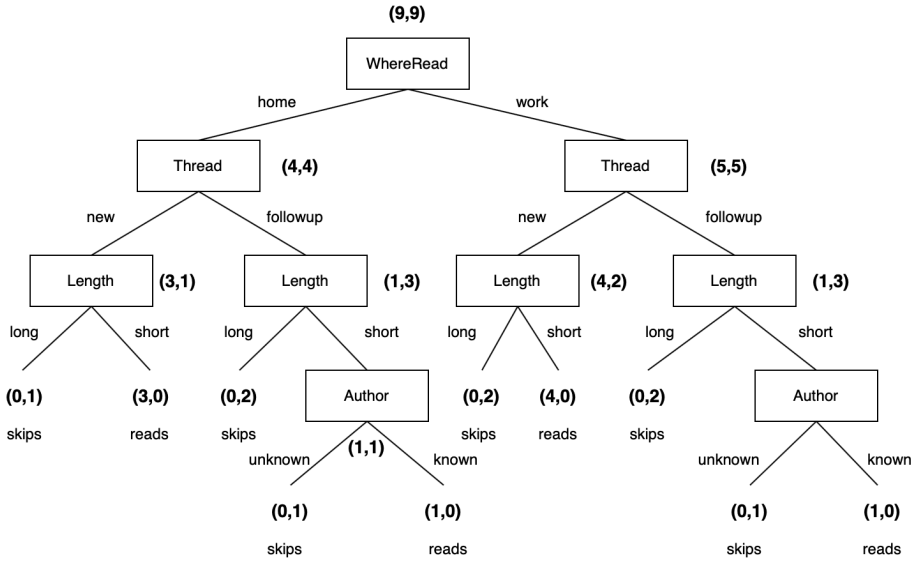


Above is the decision tree found by applying order of *[Author,Thread,Length,WhereRead]*. Comparing to the decision tree with the maximum information gain split, it has a **different** function obviously.

Maximum information gain split:
- skips: long
- reads: short and new
- skips: short and followup and unknown
- reads: short and followup and known

First element split:
- skips: unknown and followup
- reads: unknown and new
- skips: known and new and long
- reads: known and new and short
- skips: known and followup and long
- reads: known and followup and short

b)



Comparing to preceding splits, this tree decision has the same function with the maximum information gain split.
Current split function:

skips: home and new and long
skips: home and followup and long
skips: home and followup and short and unknown
skips: work and new and long
skips: work and followup and short and unknown

reads: home and new and short
reads: home and followup and short and known
reads: work and new and short
reads: work and followup and short and known

Do simplification here:

(skips: home and new and long,
skips: home and followup and long,
skips: work and new and long) can be merge into
skips: long

(skips: home and followup and short and unknown,
skips: work and followup and short and unknown) can be merge into
skips: followup and short and unknown

(reads: home and new and short,
reads: work and new and short) can be merge into
reads: new and short

(reads: home and followup and short and known,
reads: work and followup and short and known) can be merge into
reads: followup and short and known

After simplifications:

skips: long
skips: followup and short and unknown
reads: new and short
reads: followup and short and known

Which is the **same** with the **maximum information gain split**:

skips: long
reads: short and new
skips: short and followup and unknown
reads: short and followup and known

c)

No, there isn't.

Because other tree decisions can all be transferred into two functions above.
According to a) and b), it can be inferred that "Where_read" will **not** affect the classification result.
Combinations of *Author Length Thread* will be enough to determine the decision tree.

[Author Length Thread]
[Author Thread Length] first element
[Length Thread Author] maximum gain
[Length Author Thread]
[Thread Author Length]
[Thread Length Author] same as maximum gain split

After calculating all possible solutions, it can be concluded that all decision trees can be simplified into either first element split or maximum information gain split.

Z5142003 Tong Zheng

Question 2

By applying decision tree model

tree.py

```
from sklearn import datasets
import numpy as np
import pandas as pd
import graphviz
from sklearn import tree
from sklearn.tree.export import export_text
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn import metrics

dataset = pd.read_csv('adult.data', header=None)
X = dataset.iloc[: , :-1].values
# select until second last column of all rows
y = dataset.iloc[:, -1].values
# select last column of all rows
imp = SimpleImputer(missing_values=' ?', strategy="most_frequent")
# replace '?' with most frequent value

X[:, ::] = imp.fit_transform(X[:, ::])
labelencoder_X = LabelEncoder()
labelencoder_X.fit(X[: , 1])
# this step encodes target labels with value between 0 and n_classes-1
# transfer them from String to Integer to make it easier to be classified
X[: , 1] = labelencoder_X.fit_transform(X[: , 1])
X[: , 3] = labelencoder_X.fit_transform(X[: , 3])
X[: , 5] = labelencoder_X.fit_transform(X[: , 5])
X[: , 6] = labelencoder_X.fit_transform(X[: , 6])
X[: , 7] = labelencoder_X.fit_transform(X[: , 7])
X[: , 8] = labelencoder_X.fit_transform(X[: , 8])
X[: , 9] = labelencoder_X.fit_transform(X[: , 9])
X[: , 13] = labelencoder_X.fit_transform(X[: , 13])

clf = tree.DecisionTreeClassifier(random_state=0, criterion='entropy',max_depth=8,
    splitter='best', min_samples_split=30)
clf = clf.fit(X, y)
tree.plot_tree(clf.fit(X, y))
r = export_text(clf)
print(r)

"""TESTING"""
test_dataset = pd.read_csv('adult.test', header=None ,skip_blank_lines=True, skiprows=1)
test_X = test_dataset.iloc[: , :-1].values
test_y = test_dataset.iloc[:, -1].values

test_X[:, ::] = imp.fit_transform(test_X[:, ::])
labelencoder = labelencoder_X
test_X[: , 1] = labelencoder_X.fit_transform(test_X[: , 1])
test_X[: , 3] = labelencoder_X.fit_transform(test_X[: , 3])
test_X[: , 5] = labelencoder_X.fit_transform(test_X[: , 5])
test_X[: , 6] = labelencoder_X.fit_transform(test_X[: , 6])
test_X[: , 7] = labelencoder_X.fit_transform(test_X[: , 7])
test_X[: , 8] = labelencoder_X.fit_transform(test_X[: , 8])
test_X[: , 9] = labelencoder_X.fit_transform(test_X[: , 9])
test_X[: , 13] = labelencoder_X.fit_transform(test_X[: , 13])
pred = clf.predict(test_X)
test_y = [i.replace(".", "") for i in test_y]
print("The accury is: " + str(metrics.accuracy_score(test_y, pred)))
```

sklearn.impute.SimpleImputer

class sklearn.impute.**SimpleImputer**(*missing_values=nan, strategy='mean', fill_value=None, verbose=0, copy=True, add_indicator=False*) [source]

Imputation transformer for completing missing values.

Read more in the User Guide.

Parameters:

missing_values : *number, string, np.nan (default) or None*
The placeholder for the missing values. All occurrences of `missing_values` will be imputed.

strategy : *string, default='mean'*
The imputation strategy.

- If "mean", then replace missing values using the mean along each column. Can only be used with numeric data.
- If "median", then replace missing values using the median along each column. Can only be used with numeric data.
- If "most_frequent", then replace missing using the most frequent value along each column. Can be used with strings or numeric data.
- If "constant", then replace missing values with fill_value. Can be used with strings or numeric data.

sklearn.preprocessing.LabelEncoder

class sklearn.preprocessing.**LabelEncoder** [source]

Encode target labels with value between 0 and n_classes-1.

This transformer should be used to encode target values, *i.e.* `y`, and not the input `X`.

Read more in the User Guide.

New in version 0.12.

Attributes:

classes_ : *array of shape (n_class,)*
Holds the label for each class.

See also:

sklearn.preprocessing.OrdinalEncoder
Encode categorical features using an ordinal encoding scheme.

```
|   |   |   |   |   |   |   |   |--- feature_10 >  7669.50
|   |   |   |   |   |   |   |   |    |--- class:  >50K
|   |   |   |   |   |   |   |--- feature_5 >  1.00
|   |   |   |   |   |   |   |    |--- class:  >50K
|   |   |   |--- feature_4 >  10.50
|   |   |   |    |--- class:  >50K

The accury is: 0.8569498188071986
```

Import the dataset by using read_csv() from panda. By using SimpleImputer, I did preprocessing as replacing missing value like '?' with other values from dataset under specific strategy. Because of the String values in the dataset, I used labelEncoder() to change them into integers so that classifications can be done easier. And I used adult.data as training_set, and adult.test as testing_set.