Tugas Besar IF3170 - Aplikasi Web Prediksi Income Per Tahun

Kelompok : Markingat

Vigor Akbar - 13515031 (K-01)

Muhamad Irfan Maulana - 13515037 (K-01)

Luthfi Fadillah - 13515072 (K-03)

Aya Aurora Rimbamorani - 13515098 (K-02)

Iftitakhul Zakiah - 13515114 (K-03)

Reading External Files

In [45]:

```
#Reading External Files
import pandas as pd

#Reading column names
cencus_income_names=["age",
                     "workclass",
                     "fnlwgt",
                     "education",
                     "education-num",
                     "marital-status",
                     "occupation",
                     "relationship",
                     "race",
                     "sex",
                     "capital-gain",
                     "capital-loss",
                     "hours-per-week",
                     "native-country"]

#Reading data attribute from external file
cencus_income_learning_data = pd.read_csv("CencusIncome.data.txt",
                                 header=None,
                                 na_values=["?"],
                                 skipinitialspace=True,
                                 usecols=list(range(0,14)))
print("Learning Data")
print(cencus_income_learning_data)
print()

#Reading data target from external file
cencus_income_learning_target = pd.read_csv("CencusIncome.data.txt",
                                   header=None,
                                   na_values=["?"],
                                   skipinitialspace=True,
                                   usecols=[14])
print("Learning Target")
print(cencus_income_learning_target)
print()

#reading test atrribute from external file
cencus_income_test_data = pd.read_csv("CencusIncome.test.txt",
                                      header=None,
                                      na_values=["?"],
                                      skipinitialspace=True,
                                      usecols=list(range(0,14)),
                                      comment='|')
print("Test Data")
print(cencus_income_test_data)

#Reading data target from external file
cencus_income_test_target = pd.read_csv("CencusIncome.test.txt",
                                        header=None,
                                        na_values=["?"],
                                        skipinitialspace=True,
                                        usecols=[14],
                                        comment='|')
print("Test Target")
print(cencus_income_test_target)
print()
```

```
#Convert to numpy array
cencus_income_learning_data = cencus_income_learning_data.values
cencus_income_learning_target = cencus_income_learning_target.values
cencus_income_test_data = cencus_income_test_data.values
cencus_income_test_target = cencus_income_test_target.values
print ("Convert Done!")

#print numpy array
print()
print(cencus_income_learning_data)
print()
print(cencus_income_learning_target)
print()
print(cencus_income_test_data)
print()
print(cencus_income_test_target)
```

```
6      29               NaN  227026       HS-grad   9       Never-marrie
d
7      63  Self-emp-not-inc  104626   Prof-school  15  Married-civ-spous
e
8      24           Private  369667  Some-college  10       Never-marrie
d
9      55           Private  104996       7th-8th   4  Married-civ-spous
e
10     65           Private  184454       HS-grad   9  Married-civ-spous
e
11     36       Federal-gov  212465     Bachelors  13  Married-civ-spous
e
12     26           Private   82091       HS-grad   9       Never-marrie

d
13     58               NaN  299831       HS-grad   9  Married-civ-spous
e
14     48           Private  279724       HS-grad   9  Married-civ-spous
e
15     43           Private  346189       Masters  14  Married-civ-spous
```

Import Library

In [62]:

```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.model_selection import train_test_split, cross_val_predict, KFold
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

import pandas
from sklearn.tree import DecisionTreeClassifier, export_graphviz

def encode_target(df, target_column):
    df_mod = df.copy()
    targets = df_mod[target_column].unique()
    map_to_int = {name: n for n, name in enumerate(targets)}
    df_mod[target_column] = df_mod[target_column].replace(map_to_int)
    return (df_mod, map_to_int)

# Load data train
testData = pd.read_csv("CencusIncome.data.txt",
                                 header=None,
                                 na_values=["?"],
                                 skipinitialspace=True,
                                 usecols=list(range(0,14)))

for i in range(14):
    if( (i != 0) and (i != 2) and (i != 4) and (i != 10) and (i != 11) and (i != 12) ):
        testData, mappingTest = encode_target(testData, i)

testTarget = pd.read_csv("CencusIncome.data.txt",
                                   header=None,
                                   na_values=["?"],
                                   skipinitialspace=True,
                                   usecols=[14])

testTarget, mapping_target = encode_target(testTarget, 14)

target = [x[0] for x in testTarget.values]

# Load data test
test1 = pd.read_csv("CencusIncome.test.txt",
                                      header=None,
                                      na_values=["?"],
                                      skipinitialspace=True,
                                      usecols=list(range(0,14)),
                                      comment='|')

for i in range(14):
    if( (i != 0) and (i != 2) and (i != 4) and (i != 10) and (i != 11) and (i != 12) ):
        test1, mappingTest1 = encode_target(test1, i)


#Reading data target from external file
test2 = pd.read_csv("CencusIncome.test.txt",
                                        header=None,
                                        na_values=["?"],
                                        skipinitialspace=True,
                                        usecols=[14],
                                            comment='|')


test2, mappingTest2 = encode_target(test2, 14)
target2 = [x[0] for x in test2.values]
```

1. Naive Bayes Learning

In [19]:

```
from sklearn import metrics
from sklearn.model_selection import cross_val_predict
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
predicted = cross_val_predict(gnb, testData, target, cv=10)
print("accuracy GaussianNB = %f" %metrics.accuracy_score(target, predicted))

mnb = MultinomialNB()
predicted = cross_val_predict(mnb, testData, target, cv=10)
print("accuracy MultinomialNB = %f" %metrics.accuracy_score(target, predicted))

bnb = BernoulliNB()
predicted = cross_val_predict(bnb, testData, target, cv=10)
print("accuracy BernoulliNB = %f" %metrics.accuracy_score(target, predicted))
```

```
accuracy GaussianNB = 0.795492
accuracy MultinomialNB = 0.782623
accuracy BernoulliNB = 0.806148
```

2. K-Nearest Neighbours

In [50]:

```
x_full_training = testData.values
y_full_training = target
x_training, x_test, y_training, y_test = train_test_split(x_full_training, y_full_training,

neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(x_training, y_training)

y_pred = cross_val_predict(neigh, x_full_training, y_full_training, cv=10)

predict_accuracy = accuracy_score(y_full_training, y_pred)
print("accuracy kNN : %f" %predict_accuracy)
```

```
accuracy kNN : 0.761279
```

3. MLP

In [21]:

```
from sklearn.preprocessing import maxabs_scale

mlp = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(100,), random_state=1)

tD_scaled = maxabs_scale(testData, axis=0, copy=False)
tT_scaled = maxabs_scale(testTarget, axis=0, copy=False)

mlp.fit(tD_scaled, tT_scaled)

cencus_target_pred = cross_val_predict(mlp, tD_scaled, tT_scaled, cv=10)
predict_accuracy = accuracy_score(tT_scaled, cencus_target_pred)
print("accuracy MLP : %f" %predict_accuracy)
```

```
accuracy MLP : 0.850926
```

4. Decision Tree Learning

In [22]:

```
clf = tree.DecisionTreeClassifier()
predicted = cross_val_predict(clf, testData.values, testTarget.values, cv=10)
print("accuracy DTL = %f" %metrics.accuracy_score(testTarget.values, predicted))
```

```
accuracy DTL = 0.812475
```

Confusion Matrix pada Model dengan Akurasi tertinggi (MLP)

In [28]:

```
conf_matrix = confusion_matrix(testTarget, cencus_target_pred)
print("confusion matrix:")
display(pd.DataFrame(conf_matrix, index= ['>50','<=50'], columns=['>50','<=50']))
```

```
confusion matrix:
```

	>50	<=50
>50	23081	1639
<=50	3215	4626

Full Training MLP menggunakan Data Test

In [33]:

```
mlp = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(100,), random_state=1)

tD_scaled = maxabs_scale(testData, axis=0, copy=False)
tT_scaled = maxabs_scale(testTarget, axis=0, copy=False)

mlp_full = mlp.fit(tD_scaled, tT_scaled)
target_pred = mlp_full.predict(tD_scaled)
predict_accuracy = accuracy_score(tT_scaled, target_pred)
print("accuracy MLP : %f" %predict_accuracy)
```

```
accuracy MLP : 0.852247
```

Save Model

In [30]:

```
from sklearn.externals import joblib
import numpy as np
joblib.dump(mlp_full, 'MLP.model')
```

Out[30]:

```
['MLP.model']
```

Read Model

In [31]:

```
MLP = joblib.load('MLP.model')
```

In [61]:

```
y_test1_pred = MLP.predict(test1)

print(y_test1_pred)
print(target2)
```

```
[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,
0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
```

In [ ]: