Baca Data

In [30]:
```
from time import time
from sklearn import preprocessing
from sklearn import metrics
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import maxabs_scale
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.externals import joblib
import pandas as pd
import bisect
import numpy as np

with open('CensusIncome.names.txt', 'r') as fname:
    sname = fname.read()
names=np.array(sname[sname.find("age"):].split(".\n"))
#print(names)

#Encode Data Workclass
workclass = preprocessing.LabelEncoder()
workclass.fit(np.array(names[1][names[1].find(":")+2:].split(", ")))
#Handle Unknown Data : '?'
le_classes = workclass.classes_.tolist()
bisect.insort_left(le_classes, '?')
workclass.classes_ = le_classes
#print(workclass.classes_)

#Encode Data Education
education = preprocessing.LabelEncoder()
education.fit(np.array(names[3][names[3].find(":")+2:].split(", ")))
le_classes = education.classes_.tolist()
bisect.insort_left(le_classes, '?')
education.classes_ = le_classes
#print(education.classes_)

marital_status = preprocessing.LabelEncoder()
marital_status.fit(np.array(names[5][names[5].find(":")+2:].split(", ")))
le_classes = marital_status.classes_.tolist()
bisect.insort_left(le_classes, '?')
marital_status.classes_ = le_classes
#print(marital_status.classes_)

occupation = preprocessing.LabelEncoder()
occupation.fit(np.array(names[6][names[6].find(":")+2:].split(", ")))
le_classes = occupation.classes_.tolist()
bisect.insort_left(le_classes, '?')
occupation.classes_ = le_classes
```

```
#print(occupation.classes_)

relationship = preprocessing.LabelEncoder()
relationship.fit(np.array(names[7][names[7].find(":")+2:].split(", ")))
le_classes = relationship.classes_.tolist()
bisect.insort_left(le_classes, '?')
relationship.classes_ = le_classes
#print(relationship.classes_)

race = preprocessing.LabelEncoder()
race.fit(np.array(names[8][names[8].find(":")+2:].split(", ")))
le_classes = race.classes_.tolist()
bisect.insort_left(le_classes, '?')
race.classes_ = le_classes
#print(race.classes_)

sex = preprocessing.LabelEncoder()
sex.fit(np.array(names[9][names[9].find(":")+2:].split(", ")))
le_classes = sex.classes_.tolist()
bisect.insort_left(le_classes, '?')
sex.classes_ = le_classes
#print(sex.classes_)

native_country = preprocessing.LabelEncoder()
native_country.fit(np.array(names[13][names[13].find(":")+2:].split(", ")))
le_classes = native_country.classes_.tolist()
bisect.insort_left(le_classes, '?')
native_country.classes_ = le_classes
#print(native_country.classes_)

#Open Data Cencus
with open('CencusIncome.data.txt', 'r') as fdata:
    s = fdata.read()

raw = s.split("\n")
A = []
#print(raw[32560])
length = 32560
for i in range(length):
    A.append(raw[i].split(", "))

#Mapping from raw data to
data_length = 14
cencus_data = []
cencus_target = []
for i in range(length):
    temp = []
    for j in range(data_length):
        if j == 7:
            temp.append(relationship.transform([A[i][j]])[0])
        elif j == 1:
            temp.append(workclass.transform([A[i][j]])[0])
        elif j == 3:
            temp.append(education.transform([A[i][j]])[0])
        elif j == 5:
            temp.append(marital_status.transform([A[i][j]])[0])
        elif j == 6:
```

```
            temp.append(occupation.transform([A[i][j]])[0])
        elif j == 8:
            temp.append(race.transform([A[i][j]])[0])
        elif j == 9:
            temp.append(sex.transform([A[i][j]])[0])
        elif j == 13:
            temp.append(native_country.transform([A[i][j]])[0])
        else:
            temp.append(int(A[i][j]))
    cencus_data.append(temp)
    cencus_target.append(A[i][data_length])

#Encode Target
target = preprocessing.LabelEncoder()
target.fit(cencus_target)

#Ready to Use Data
y = target.transform(cencus_target)
X = np.array(cencus_data)

#Scaling
y = maxabs_scale(y, axis=0, copy=False)
X = maxabs_scale(X, axis=0, copy=False)
```

k-Nearest Neighbour

In [31]:
```
kf = KFold(n_splits=10, shuffle=True)

table = []
training_times = []
prediction_times = []
scores = []
i = 0
for train, test in kf.split(X, y):
    x_train = [X[i] for i in train]
    y_train = [y[i] for i in train]
    x_test = [X[i] for i in test]
    y_test = [y[i] for i in test]

    t0 = time()
    model = KNeighborsClassifier(n_neighbors=41).fit(x_train, y_train)
    training_time = '{:.6}'.format(time() - t0)

    t0 = time()
    pred = model.predict(x_test)
    prediction_time = '{:.6}'.format(time() - t0)

    score = accuracy_score(pred, y_test)
    cm = confusion_matrix(y_test, pred)
    i = i+1
    print ("Confusion Matrix Iterasi", i)
    display(pd.DataFrame(confusion_matrix(y_test, pred), columns=['>50K','<=50
K'], index=['>50K','<=50K']))

    table += [[training_time, prediction_time, score]]
    training_times += [training_time]
    prediction_times += [prediction_time]
    scores += [score]

display(pd.DataFrame(table, columns=['training time', 'prediction time', 'scor
e'], index=range(1,len(table)+1)))
```

```
Confusion Matrix Iterasi 1
```

	>50K	**<=50K**
>50K	2227	219
<=50K	348	462

```
Confusion Matrix Iterasi 2
```

	>50K	**<=50K**
>50K	2296	211
<=50K	333	416

```
Confusion Matrix Iterasi 3
```

	>50K	**<=50K**
>50K	2302	187
<=50K	336	431

```
Confusion Matrix Iterasi 4
```

	>50K	**<=50K**
>50K	2251	217
<=50K	338	450

```
Confusion Matrix Iterasi 5
```

	>50K	**<=50K**
>50K	2257	203
<=50K	331	465

```
Confusion Matrix Iterasi 6
```

	>50K	**<=50K**
>50K	2246	192
<=50K	364	454

```
Confusion Matrix Iterasi 7
```

	>50K	<=50K
>50K	2257	199
<=50K	351	449

```
Confusion Matrix Iterasi 8
```

	>50K	<=50K
>50K	2271	205
<=50K	350	430

```
Confusion Matrix Iterasi 9
```

	>50K	<=50K
>50K	2299	191
<=50K	330	436

```
Confusion Matrix Iterasi 10
```

	>50K	<=50K
>50K	2298	192
<=50K	344	422

	training time	prediction time	score
1	1.25183	2.63625	0.825860
2	1.2017	2.40814	0.832924
3	1.23132	2.63175	0.839373
4	1.24483	2.46364	0.829545
5	1.2123	2.57773	0.835995
6	1.30187	2.5627	0.829238
7	1.3584	2.64776	0.831081
8	1.28385	2.48465	0.829545
9	1.25303	2.53324	0.839988
10	1.28285	2.38411	0.835381

Naive Bayes

In [32]:
```
#older vers: kf = StratifiedKFold(y, n_folds=10, shuffle=True)
#newer vers: kf = KFold(n_splits=10, shuffle=True)
kf = KFold(n_splits=10, shuffle=True)

table = []
sum = 0
i = 1
for train, test in kf.split(X,y):
    x_train = [X[i] for i in train]
    y_train = [y[i] for i in train]
    x_test = [X[i] for i in test]
    y_test = [y[i] for i in test]

    ts = time()
    model = GaussianNB().fit(x_train, y_train)
    trainingtime = time() - ts

    ts = time()
    prediction = model.predict(x_test)
    predictiontime = time() - ts

    score = accuracy_score(prediction, y_test)
    sum += score

    print(i)
    i += 1
    print(metrics.confusion_matrix(y_test,prediction))

    table += [[score, trainingtime, predictiontime]]

print()
print(pd.DataFrame(table, columns=['score', 'training time', 'prediction time'
], index=range(1,len(table)+1)))
print()
print("rata-rata score: ", sum/10)
```

```
1
[[2315  103]
 [ 557  281]]
2
[[2347  113]
 [ 533  263]]
3
[[2348  128]
 [ 519  261]]
4
[[2345  130]
 [ 515  266]]
5
[[2382  120]
 [ 491  263]]
6
[[2372  115]
 [ 518  251]]
7
[[2400  107]
 [ 495  254]]
8
[[2322  130]
 [ 502  302]]
9
[[2341  121]
 [ 527  267]]
10
[[2347  134]
 [ 515  260]]

       score  training time  prediction time
1   0.797297       0.044529         0.004003
2   0.801597       0.051535         0.003502
3   0.801290       0.043529         0.003502
4   0.801904       0.052036         0.004005
5   0.812346       0.043529         0.003502
6   0.805590       0.043529         0.003502
7   0.815111       0.049533         0.004003
8   0.805897       0.044030         0.003502
9   0.800983       0.044030         0.003502
10  0.800676       0.058539         0.003502

rata-rata score:  0.804269041769
```

Decision Tree Learning

In [33]:
```
kf = KFold(n_splits=10, shuffle=True)

table = []
training_times = []
prediction_times = []
scores = []
i = 0
for train, test in kf.split(X,y):
    x_train = [X[i] for i in train]
    y_train = [y[i] for i in train]
    x_test = [X[i] for i in test]
    y_test = [y[i] for i in test]

    t0 = time()
    model = tree.DecisionTreeClassifier().fit(x_train, y_train)
    training_time = '{:.6}'.format(time() - t0)

    t0 = time()
    y_pred = model.predict(x_test)
    prediction_time = '{:.6}'.format(time() - t0)

    score = accuracy_score(y_pred, y_test)

    cm = confusion_matrix(y_test, y_pred)
    i = i+1
    print("Confusion Matrix Iterasi =", i)
    display(pd.DataFrame(cm, columns=['>50K','<=50K'], index=['>50K','<=50K'
]))

    table += [[training_time, prediction_time, score]]
    training_times += [training_time]
    prediction_times += [prediction_time]
    scores += [score]

display(pd.DataFrame(table, columns=['training time', 'prediction time', 'scor
e'], index=range(1,len(table)+1)))
```

Confusion Matrix Iterasi = 1

	>50K	<=50K
>50K	2177	289
<=50K	297	493

Confusion Matrix Iterasi = 2

	>50K	<=50K
>50K	2154	312
<=50K	276	514

Confusion Matrix Iterasi = 3

	>50K	<=50K
>50K	2140	331
<=50K	313	472

Confusion Matrix Iterasi = 4

	>50K	<=50K
>50K	2141	328
<=50K	279	508

Confusion Matrix Iterasi = 5

	>50K	<=50K
>50K	2141	343
<=50K	312	460

Confusion Matrix Iterasi = 6

	>50K	<=50K
>50K	2168	313
<=50K	286	489

Confusion Matrix Iterasi = 7

	>50K	<=50K
>50K	2125	327
<=50K	294	510

```
Confusion Matrix Iterasi = 8
```

	>50K	<=50K
>50K	2162	319
<=50K	306	469

```
Confusion Matrix Iterasi = 9
```

	>50K	<=50K
>50K	2145	337
<=50K	298	476

```
Confusion Matrix Iterasi = 10
```

	>50K	<=50K
>50K	2151	317
<=50K	306	482

	training time	prediction time	score
1	0.21364	0.00350285	0.820025
2	0.213642	0.00300312	0.819410
3	0.215142	0.00300193	0.802211
4	0.223647	0.0035038	0.813575
5	0.226647	0.00300193	0.798833
6	0.220147	0.00300193	0.816032
7	0.215143	0.00300312	0.809275
8	0.217146	0.00300193	0.808047
9	0.214641	0.00300336	0.804975
10	0.216645	0.00300193	0.808661

Multi Layered Perceptron

In [34]:
```
mlp = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(100,), random_state=1)
kf = KFold(n_splits=10, shuffle=True)

table = []
training_times = []
prediction_times = []
scores = []
i = 0

for train, test in kf.split(X, y):
    x_train = [X[i] for i in train]
    y_train = [y[i] for i in train]
    x_test = [X[i] for i in test]
    y_test = [y[i] for i in test]

    #Xcoba = maxabs_scale(X, axis=0, copy=True)
    x_train = maxabs_scale(x_train, axis=0, copy=False)
    y_train = maxabs_scale(y_train, axis=0, copy=False)
    x_test = maxabs_scale(x_test, axis=0, copy=False)
    y_test = maxabs_scale(y_test, axis=0, copy=False)

    t0 = time()
    mlp.fit(x_train, y_train)
    training_time = '{:.6}'.format(time() - t0)

    t0 = time()
    pred = mlp.predict(x_test)
    prediction_time = '{:.6}'.format(time() - t0)

    score = accuracy_score(pred, y_test)
    cm = confusion_matrix(y_test, pred)
    i = i+1
    print ("Confusion Matrix Iterasi", i)
    display(pd.DataFrame(confusion_matrix(y_test, pred), columns=['>50K','<=50
K'], index=['>50K','<=50K']))

    table += [[training_time, prediction_time, score]]
    training_times += [training_time]
    prediction_times += [prediction_time]
    scores += [score]

display(pd.DataFrame(table, columns=['training time', 'prediction time', 'scor
e'], index=range(1,len(table)+1)))
```

Confusion Matrix Iterasi 1

	>50K	<=50K
>50K	2279	187
<=50K	346	444

Confusion Matrix Iterasi 2

	>50K	<=50K
>50K	2296	176
<=50K	325	459

Confusion Matrix Iterasi 3

	>50K	<=50K
>50K	2284	168
<=50K	338	466

Confusion Matrix Iterasi 4

	>50K	<=50K
>50K	2312	180
<=50K	305	459

Confusion Matrix Iterasi 5

	>50K	<=50K
>50K	2287	175
<=50K	322	472

Confusion Matrix Iterasi 6

	>50K	<=50K
>50K	2289	193
<=50K	332	442

Confusion Matrix Iterasi 7

	>50K	<=50K
>50K	2300	172
<=50K	314	470

Confusion Matrix Iterasi 8

	>50K	<=50K
>50K	2270	198
<=50K	304	484

Confusion Matrix Iterasi 9

	>50K	<=50K
>50K	2324	162
<=50K	321	449

Confusion Matrix Iterasi 10

	>50K	<=50K
>50K	2304	164
<=50K	347	441

	training time	prediction time	score
1	25.7612	0.0045042	0.836302
2	23.0972	0.00450301	0.846130
3	21.2907	0.00500202	0.844595
4	21.4704	0.00450516	0.851044
5	21.4719	0.00450373	0.847359
6	22.4472	0.00500369	0.838759
7	22.0498	0.00450373	0.850737
8	21.1423	0.00450325	0.845823
9	22.3873	0.00500417	0.851658
10	21.2605	0.004004	0.843059

Baca Test

In [35]:
```
#Open Test
with open('CencusIncome.test.txt', 'r') as fdata:
    stest = fdata.read()

test_raw = np.array(stest[stest.find("\n")+1:].split(".\n"))
B = []
length = len(test_raw)-1
for i in range(length):
    B.append(test_raw[i].split(", "))

#Mapping from raw data to
data_length = 14
cencus_data = []
cencus_target = []
for i in range(length):
    temp = []
    for j in range(data_length):
        if j == 7:
            temp.append(relationship.transform([B[i][j]])[0])
        elif j == 1:
            temp.append(workclass.transform([B[i][j]])[0])
        elif j == 3:
            temp.append(education.transform([B[i][j]])[0])
        elif j == 5:
            temp.append(marital_status.transform([B[i][j]])[0])
        elif j == 6:
            temp.append(occupation.transform([B[i][j]])[0])
        elif j == 8:
            temp.append(race.transform([B[i][j]])[0])
        elif j == 9:
            temp.append(sex.transform([B[i][j]])[0])
        elif j == 13:
            temp.append(native_country.transform([B[i][j]])[0])
        else:
            temp.append(int(B[i][j]))
    cencus_data.append(temp)
    cencus_target.append(B[i][data_length])

#Encode Target
target = preprocessing.LabelEncoder()
target.fit(cencus_target)

#Ready to Use Data
y_test = target.transform(cencus_target)
X_test = np.array(cencus_data)

#Scaling
y_test = maxabs_scale(y_test, axis=0, copy=False)
X_test = maxabs_scale(X_test, axis=0, copy=False)
```

Save, load, dan predict using best model

In [36]:
```
model = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(100,), random_state=1)
model.fit(X,y)
joblib.dump(model, 'Best.model')

bestmodel = joblib.load('Best.model')
prediction = bestmodel.predict(X_test)

display(pd.DataFrame(confusion_matrix(y_test, prediction), columns=['>50K','<=
50K'], index=['>50K','<=50K']))
```

	>50K	<=50K
>50K	11558	877
<=50K	1588	2258

In [37]:
```
print("Score: ", accuracy_score(prediction, y_test))
```

```
Score:  0.848596523555
```