Tugas Besar IF3170

Inteligensia Buatan 2017/2018

Eksperimen untuk mendapatkan model terbaik

Anggota Kelompok

- 13515019 - Candra Hesen Parera
- 13515024 - Abdurrahman
- 13515059 - Muthmainnah
- 13515090 - Annisa Muzdalifa
- 13515129 - Annisa Nurul Azhar

Baca Data

In [2]:

```
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 [3]:

```
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','<=50K'], inde
x=['>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', 'score'], inde
x=range(1,len(table)+1)))
```

```
Confusion Matrix Iterasi 1
```

	>50K	<=50K
>50K	2292	211
<=50K	328	425

```
Confusion Matrix Iterasi 2
```

	>50K	<=50K
>50K	2296	174
<=50K	337	449

```
Confusion Matrix Iterasi 3
```

	>50K	<=50K
>50K	2270	217
<=50K	346	423

```
Confusion Matrix Iterasi 4
```

	>50K	<=50K
>50K	2270	191
<=50K	335	460

```
Confusion Matrix Iterasi 5
```

	>50K	<=50K
>50K	2222	217
<=50K	369	448

```
Confusion Matrix Iterasi 6
```

	>50K	<=50K
>50K	2280	205
<=50K	332	439

```
Confusion Matrix Iterasi 7
```

	>50K	<=50K
>50K	2279	194
<=50K	333	450

```
Confusion Matrix Iterasi 8
```

	>50K	<=50K
>50K	2248	212
<=50K	332	464

```
Confusion Matrix Iterasi 9
```

	>50K	<=50K
>50K	2278	183
<=50K	359	436

```
Confusion Matrix Iterasi 10
```

	>50K	<=50K
>50K	2275	206
<=50K	349	426

	training time	prediction time	score
1	7.24724	6.34647	0.834459
2	5.21148	6.03431	0.843059
3	5.21375	6.01869	0.827088
4	5.1769	5.89727	0.838452
5	5.18501	5.97231	0.820025
6	5.1851	6.44159	0.835074
7	5.20247	5.95963	0.838145
8	5.21072	5.97181	0.832924
9	5.23347	6.14321	0.833538
10	5.17203	6.15882	0.829545

Naive Bayes

In [4]:

```
#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
[[2344  116]
 [ 533  263]]
2
[[2341  131]
 [ 502  282]]
3
[[2379  124]
 [ 486  267]]
4
[[2393  103]
 [ 525  235]]
5
[[2288  128]
 [ 553  287]]
6
[[2354  118]
 [ 520  264]]
7
[[2371  113]
 [ 522  250]]
8
[[2351  117]
 [ 525  263]]
9
[[2352  110]
 [ 498  296]]
10
[[2352  135]
 [ 511  258]]

       score  training time  prediction time
1   0.800676       0.098430         0.000000
2   0.805590       0.093757         0.015623
3   0.812654       0.093759         0.015624
4   0.807125       0.093755         0.000000
5   0.790848       0.093759         0.000000
6   0.804054       0.093754         0.015076
7   0.804975       0.102285         0.000000
8   0.802826       0.093756         0.000000
9   0.813268       0.082302         0.015626
10  0.801597       0.093754         0.000000

rata-rata score:  0.804361179361
```

Decision Tree Learning

In [5]:

```
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', 'score'], inde
x=range(1,len(table)+1)))
```

```
Confusion Matrix Iterasi = 1
```

	>50K	<=50K
>50K	2117	320
<=50K	321	498

```
Confusion Matrix Iterasi = 2
```

	>50K	<=50K
>50K	2128	302
<=50K	330	496

```
Confusion Matrix Iterasi = 3
```

	>50K	<=50K
>50K	2149	331
<=50K	288	488

```
Confusion Matrix Iterasi = 4
```

	>50K	<=50K
>50K	2130	299
<=50K	318	509

```
Confusion Matrix Iterasi = 5
```

	>50K	<=50K
>50K	2134	350
<=50K	306	466

```
Confusion Matrix Iterasi = 6
```

	>50K	<=50K
>50K	2163	347
<=50K	315	431

```
Confusion Matrix Iterasi = 7
```

	>50K	<=50K
>50K	2138	348
<=50K	257	513

```
Confusion Matrix Iterasi = 8
```

	>50K	<=50K
>50K	2165	339
<=50K	268	484

```
Confusion Matrix Iterasi = 9
```

	>50K	<=50K
>50K	2176	296
<=50K	307	477

```
Confusion Matrix Iterasi = 10
```

	>50K	<=50K
>50K	2204	284
<=50K	266	502

	training time	prediction time	score
1	0.621935	0.0	0.803133
2	0.453149	0.0	0.805897
3	0.46877	0.0	0.809889
4	0.453068	0.0156279	0.810504
5	0.484396	0.0	0.798526
6	0.466918	0.0	0.796683
7	0.453145	0.0156288	0.814189
8	0.450313	0.0156271	0.813575
9	0.472789	0.0	0.814803
10	0.468767	0.0156312	0.831081

Multi Layered Perceptron

In [6]:

```
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','<=50K'], inde
x=['>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', 'score'], inde
x=range(1,len(table)+1)))
```

```
Confusion Matrix Iterasi 1
```

	>50K	**<=50K**
>50K	2250	211
<=50K	320	475

```
Confusion Matrix Iterasi 2
```

	>50K	**<=50K**
>50K	2308	154
<=50K	307	487

```
Confusion Matrix Iterasi 3
```

	>50K	**<=50K**
>50K	2316	168
<=50K	309	463

```
Confusion Matrix Iterasi 4
```

	>50K	**<=50K**
>50K	2263	188
<=50K	365	440

```
Confusion Matrix Iterasi 5
```

	>50K	**<=50K**
>50K	2264	199
<=50K	328	465

```
Confusion Matrix Iterasi 6
```

	>50K	**<=50K**
>50K	2264	211
<=50K	285	496

```
Confusion Matrix Iterasi 7
```

	>50K	**<=50K**
>50K	2290	209
<=50K	318	439

```
Confusion Matrix Iterasi 8
```

	>50K	<=50K
>50K	2295	151
<=50K	342	468

```
Confusion Matrix Iterasi 9
```

	>50K	<=50K
>50K	2301	174
<=50K	286	495

```
Confusion Matrix Iterasi 10
```

	>50K	<=50K
>50K	2337	167
<=50K	314	438

	training time	prediction time	score
1	54.1628	0.0	0.836916
2	44.8927	0.0156298	0.858415
3	44.8549	0.0	0.853501
4	44.0849	0.0156293	0.830160
5	43.9472	0.0	0.838145
6	45.0115	0.0	0.847666
7	44.1756	0.0	0.838145
8	43.9954	0.0	0.848587
9	44.6645	0.015631	0.858722
10	44.4877	0.0156295	0.852273

Baca Test

In [7]:

```
#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 [8]:

```
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	11588	847
<=50K	1618	2228

In [9]:

```
print("Score: ", accuracy_score(prediction, y_test))
```

```
Score:  0.848596523555
```