This notebook is created for CS6741 - supervised learning

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import sys
import os
#from ggplot import *
%matplotlib inline
```

1. data glance

first, take a fast data glance on UCI_credit_Card dataset.

In [2]:

```
data = pd.read_csv('UCI_Credit_Card.csv')
data.sample(5)
```

Out[2]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_
3158	3159	150000.0	2	3	1	51	0	0	0	
27431	27432	230000.0	1	2	1	41	0	0	0	
1280	1281	200000.0	2	1	2	31	0	0	0	
21119	21120	160000.0	1	1	1	40	0	0	2	
19217	19218	200000.0	2	1	2	40	0	0	0	

5 rows × 25 columns

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
ID                            30000 non-null int64
LIMIT_BAL                     30000 non-null float64
SEX                           30000 non-null int64
EDUCATION                     30000 non-null int64
MARRIAGE                      30000 non-null int64
AGE                           30000 non-null int64
PAY_0                         30000 non-null int64
PAY_2                         30000 non-null int64
PAY_3                         30000 non-null int64
PAY_4                         30000 non-null int64
PAY_5                         30000 non-null int64
PAY_6                         30000 non-null int64
BILL_AMT1                     30000 non-null float64
BILL_AMT2                     30000 non-null float64
BILL_AMT3                     30000 non-null float64
BILL_AMT4                     30000 non-null float64
BILL_AMT5                     30000 non-null float64
BILL_AMT6                     30000 non-null float64
PAY_AMT1                      30000 non-null float64
PAY_AMT2                      30000 non-null float64
PAY_AMT3                      30000 non-null float64
PAY_AMT4                      30000 non-null float64
PAY_AMT5                      30000 non-null float64
PAY_AMT6                      30000 non-null float64
default.payment.next.month    30000 non-null int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

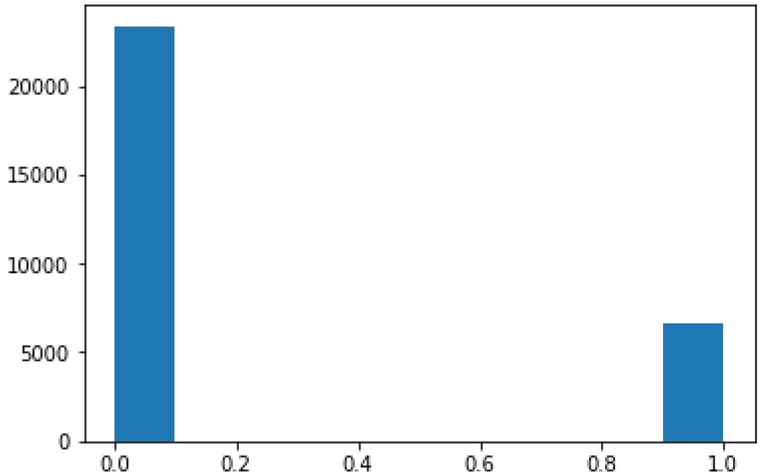
Here shows the distribution of target variable 'default_payment', 23364 v.s. 6636.

In [4]:

```
plt.hist(data['default.payment.next.month'])
```

Out[4]:

```
(array([23364.,      0.,      0.,      0.,      0.,      0.,      0.,      0.,
            0.,   6636.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```

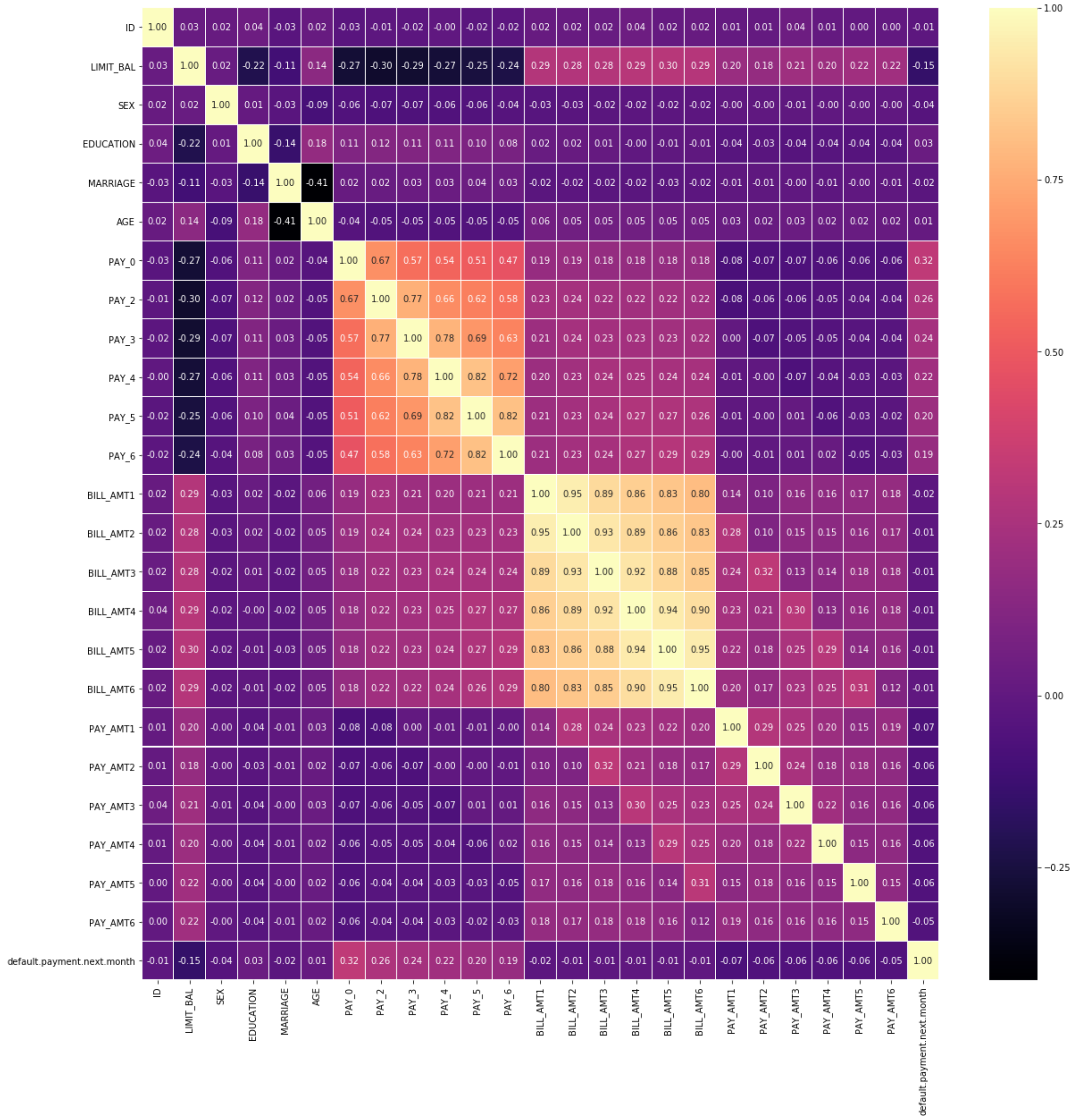


Here shows the correlation betweeen variable pairs. Light color means that the variables are highly-correlated, and dark color means versa. Note that the diagonal line says nothing.

According to the figure, 'PAY_0' to 'PAY_6' are highly correlated, so as 'BILL_AMT1' to 'BILL_AMT6'.

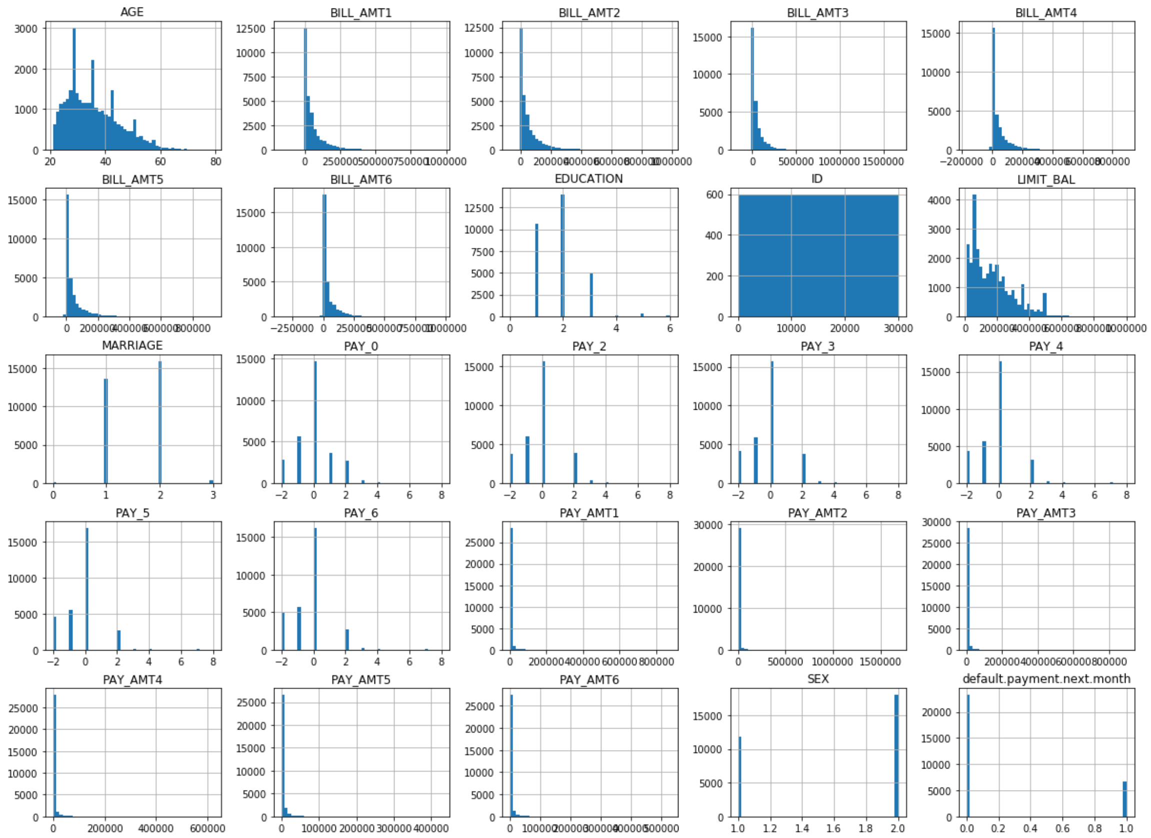
In [5]:

```
fig, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(data.corr(), ax=ax, annot=True, linewidths=0.05, fmt= '.2f',cmap="magma")
plt.show()
```



In [6]:

```
data.hist(bins=50, figsize=(20,15))
plt.show()
```



2. data cleaning

There is no missing data. However, some variables have undocumented value (see [https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients (https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients)](https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients) for full document).

- EDUCATION has category 5 and 6 that are unlabelled, moreover the category 0 is undocumented.

In the document, 1 = graduate school; 2 = university; 3 = high school; 4 = others, so 0, 5, 6 should be seen as 4.

- MARRIAGE has a label 0 that is undocumented

In the document, 1 = married; 2 = single; 3 = others, so 0 should be seen as 3.

- PAY_0 to PAY_6 all present an undocumented label -2.

In the document, 1,2,3, etc are the months of delay, 0 should be labeled 'pay duly'. So, every negative value should be seen as a 0.

In [7]:

```
data[['EDUCATION', 'MARRIAGE']].describe()
```

Out[7]:

	EDUCATION	MARRIAGE
count	30000.000000	30000.000000
mean	1.853133	1.551867
std	0.790349	0.521970
min	0.000000	0.000000
25%	1.000000	1.000000
50%	2.000000	2.000000
75%	2.000000	2.000000
max	6.000000	3.000000

In [8]:

```
data[['PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']].describe()
```

Out[8]:

	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	-0.016700	-0.133767	-0.166200	-0.220667	-0.266200	-0.291100
std	1.123802	1.197186	1.196868	1.169139	1.133187	1.149988
min	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000
25%	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000

In [9]:

```
tmp = data['EDUCATION'].values
tmp[tmp == 0] = 4
tmp[tmp == 5] = 4
tmp[tmp == 6] = 4
data['EDUCATION'] = tmp

tmp = data['MARRIAGE'].values
tmp[tmp == 0] = 3
data['EDUCATION'] = tmp

tmp = data['PAY_0'].values
tmp[tmp == -2] = 0
data['PAY_0'] = tmp
tmp = data['PAY_2'].values
tmp[tmp == -2] = 0
data['PAY_2'] = tmp
tmp = data['PAY_3'].values
tmp[tmp == -2] = 0
data['PAY_3'] = tmp
tmp = data['PAY_4'].values
tmp[tmp == -2] = 0
data['PAY_4'] = tmp
tmp = data['PAY_5'].values
tmp[tmp == -2] = 0
data['PAY_5'] = tmp
tmp = data['PAY_6'].values
tmp[tmp == -2] = 0
data['PAY_6'] = tmp
```

3. feature engineering

Some variables, such as EDUCATION, MARRIAGE and PAY_0 etc. are nomimal, i.e., their values are discrete, representing different meanings. However, some variables are continuous, such as AGE, we cannot use the same standard methods for them. Instead, we turn to quantilize the continouds variables.

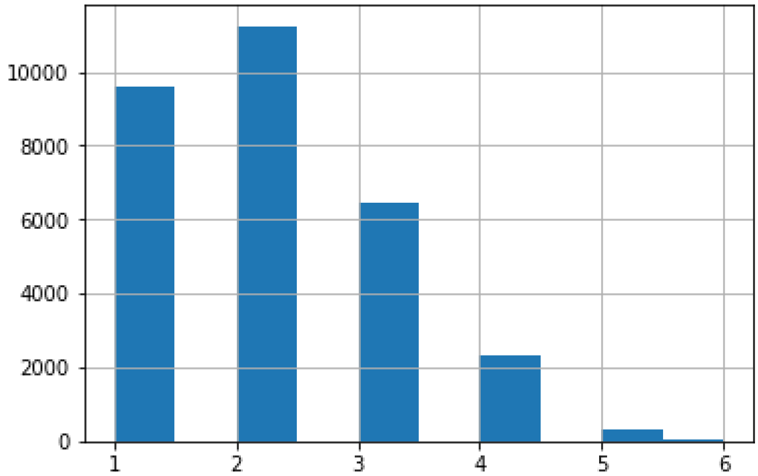
A way to examine whether our new feature will work is to check whether the correlation with the target value is larger (in absolute value) than before.

In [10]:

```
data['AgeBin'] = 0 #creates a column of 0
data.loc[((data['AGE'] > 20) & (data['AGE'] < 30)) , 'AgeBin'] = 1
data.loc[((data['AGE'] >= 30) & (data['AGE'] < 40)) , 'AgeBin'] = 2
data.loc[((data['AGE'] >= 40) & (data['AGE'] < 50)) , 'AgeBin'] = 3
data.loc[((data['AGE'] >= 50) & (data['AGE'] < 60)) , 'AgeBin'] = 4
data.loc[((data['AGE'] >= 60) & (data['AGE'] < 70)) , 'AgeBin'] = 5
data.loc[((data['AGE'] >= 70) & (data['AGE'] < 81)) , 'AgeBin'] = 6
data['AgeBin'].hist()
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15c98ff0>
```



In [11]:

```
print(data[['AgeBin', 'default.payment.next.month']].corr())
print(data[['AGE', 'default.payment.next.month']].corr())
```

```
                              AgeBin  default.payment.next.month
AgeBin                      1.000000                    0.014722
default.payment.next.month  0.014722                    1.000000
                                AGE  default.payment.next.month
AGE                         1.00000                     0.01389
default.payment.next.month  0.01389                     1.00000
```

We can see that the correlation is a bit larger (0.014722 v.s. 0.01389). And let's move on.

SEX (1 = male; 2 = female) Marital status (1 = married; 2 = single; 3 = others)

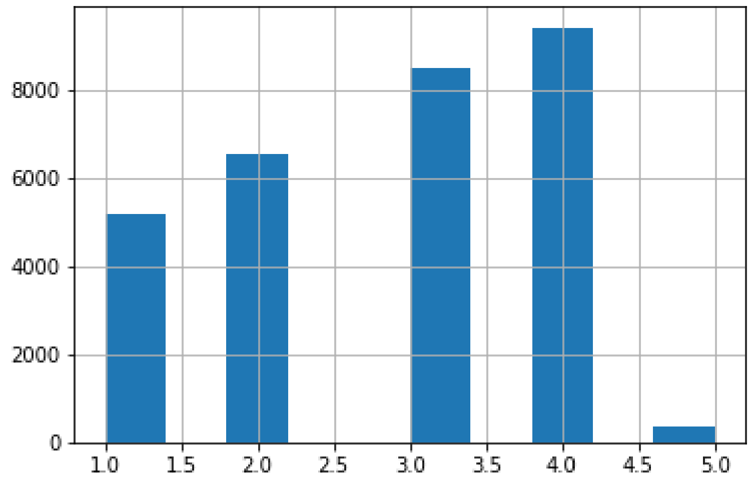
Here we define a new variable, representing for married man, single man, married woman, single woman and others

In [12]:

```
data['SE_MA'] = 5  # default: others
data.loc[((data.SEX == 1) & (data.MARRIAGE == 1)) , 'SE_MA'] = 1
data.loc[((data.SEX == 1) & (data.MARRIAGE == 2)) , 'SE_MA'] = 2
data.loc[((data.SEX == 2) & (data.MARRIAGE == 1)) , 'SE_MA'] = 3
data.loc[((data.SEX == 2) & (data.MARRIAGE == 2)) , 'SE_MA'] = 4
data['SE_MA'].hist()
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x162068b0>
```



In [13]:

```
print(data[['SE_MA', 'default.payment.next.month']].corr())
print(data[['SEX', 'default.payment.next.month']].corr())
print(data[['MARRIAGE', 'default.payment.next.month']].corr())
```

```
                               SE_MA  default.payment.next.month
SE_MA                       1.000000                   -0.046897
default.payment.next.month -0.046897                    1.000000
                                 SEX  default.payment.next.month
SEX                         1.000000                   -0.039961
default.payment.next.month -0.039961                    1.000000
                            MARRIAGE  default.payment.next.month
MARRIAGE                    1.000000                   -0.027575
default.payment.next.month -0.027575                    1.000000
```

A better variable is get now.

The last thing I am interersed in is the ratio of BILL_AMT to LIMIT_BAL. I think that if bill takes large part of credit, maybe the client will set default payment.

The following results shows that these variables will help to build a good model.

In [14]:

```
data['ratio1'] = data['BILL_AMT1'] / data['LIMIT_BAL']
data['ratio2'] = data['BILL_AMT2'] / data['LIMIT_BAL']
data['ratio3'] = data['BILL_AMT3'] / data['LIMIT_BAL']
data['ratio4'] = data['BILL_AMT4'] / data['LIMIT_BAL']
data['ratio5'] = data['BILL_AMT5'] / data['LIMIT_BAL']
data['ratio6'] = data['BILL_AMT6'] / data['LIMIT_BAL']
```

In [15]:

```
print(data[['ratio1', 'default.payment.next.month']]).corr()
print(data[['BILL_AMT1', 'default.payment.next.month']]).corr()
print()

print(data[['ratio2', 'default.payment.next.month']]).corr()
print(data[['BILL_AMT2', 'default.payment.next.month']]).corr()
print()

print(data[['ratio3', 'default.payment.next.month']]).corr()
print(data[['BILL_AMT3', 'default.payment.next.month']]).corr()
print()

print(data[['ratio4', 'default.payment.next.month']]).corr()
print(data[['BILL_AMT4', 'default.payment.next.month']]).corr()
print()

print(data[['ratio5', 'default.payment.next.month']]).corr()
print(data[['BILL_AMT5', 'default.payment.next.month']]).corr()
print()

print(data[['ratio6', 'default.payment.next.month']]).corr()
print(data[['BILL_AMT6', 'default.payment.next.month']]).corr()
print()
```

```
                              ratio1  default.payment.next.month
ratio1                      1.000000                    0.086168
default.payment.next.month  0.086168                    1.000000
                            BILL_AMT1  default.payment.next.month
BILL_AMT1                    1.000000                   -0.019644
default.payment.next.month  -0.019644                    1.000000
()
                              ratio2  default.payment.next.month
ratio2                      1.000000                    0.099039
default.payment.next.month  0.099039                    1.000000
                            BILL_AMT2  default.payment.next.month
BILL_AMT2                    1.000000                   -0.014193
default.payment.next.month  -0.014193                    1.000000
()
                              ratio3  default.payment.next.month
ratio3                      1.000000                    0.103902
default.payment.next.month  0.103902                    1.000000
                            BILL_AMT3  default.payment.next.month
BILL_AMT3                    1.000000                   -0.014076
default.payment.next.month  -0.014076                    1.000000
()
                              ratio4  default.payment.next.month
ratio4                      1.000000                    0.115925
default.payment.next.month  0.115925                    1.000000
                            BILL_AMT4  default.payment.next.month
BILL_AMT4                    1.000000                   -0.010156
default.payment.next.month  -0.010156                    1.000000
()
                              ratio5  default.payment.next.month
ratio5                      1.000000                    0.119156
default.payment.next.month  0.119156                    1.000000
                            BILL_AMT5  default.payment.next.month
BILL_AMT5                     1.00000                    -0.00676
default.payment.next.month   -0.00676                     1.00000
()
                              ratio6  default.payment.next.month
ratio6                      1.000000                    0.123373
default.payment.next.month  0.123373                    1.000000
                            BILL_AMT6  default.payment.next.month
BILL_AMT6                    1.000000                   -0.005372
default.payment.next.month  -0.005372                    1.000000
()
```

In [16]:

```
credit = data['default.payment.next.month'].values
features = data.drop(['default.payment.next.month'], axis=1).values
print ('Data is ready.')
print ('x shape', features.shape)
print ('y shape', credit.shape)
```

```
Data is ready.
('x shape', (30000, 32))
('y shape', (30000,))
```

3. model building

In [17]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import confusion_matrix
```

In [18]:

```
# use StratifiedKFold to sample according to the label distribution
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, roc_curve
from time import time

def draw_confusion_matrix(cm):
    cm = cm / float(np.sum(cm))
    df = pd.DataFrame({'0':cm[0,:],'1':cm[1,:]}, index=[0,1])
    plt.figure(figsize=(3, 3))
    plt.imshow(df, cmap=plt.get_cmap('gray_r'))
    plt.colorbar()
    plt.show()

def train_predict(clf, X_train, y_train, X_test, y_test, silent = False):
    if not silent:
        print('Building classifier: %s....'%(clf.__class__.__name__))

    # training
    t1 = time()
    clf.fit(X_train,y_train)
    t2 = time()

    # training result
    y_prob = clf.predict_proba(X_train)
    y_pred = np.argmax(y_prob, axis=1)
    acc_train, f1_train, auc_train = accuracy_score(y_train, y_pred), f1_score(y_train, y_pred),
roc_auc_score(y_train, y_prob[:,1])
    if not silent:
        print(' train set: acc=%.4f; f1=%.4f; auc=%.4f'%(acc_train, f1_train, auc_train))

    # testing result
    y_prob = clf.predict_proba(X_test)
    y_pred = np.argmax(y_prob, axis=1)
    acc_test, f1_test, auc_test = accuracy_score(y_test, y_pred), f1_score(y_test, y_pred), roc_
auc_score(y_test, y_prob[:,1])
    if not silent:
        print(' test set: acc=%.4f; f1=%.4f; auc=%.4f'%(acc_test, f1_test, auc_test))

    if not silent:
        cm = confusion_matrix(y_pred=y_pred, y_true=y_test)
        print(' test confusion martix:')
        print(cm)
        draw_confusion_matrix(cm)

    return y_prob[:,1], (acc_train, f1_train, auc_train), (acc_test, f1_test, auc_test), t2-t1
```

finetune parameters for kNN

k = 1, 3, 5, 10, 20, 100

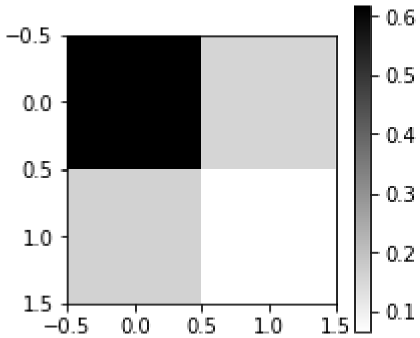
Note that when finetuning parameters, I use 5-fold average testing auc (area under roc curve) score as main metric.

Conclusion: according to the results below, we set optimal k to be 100.

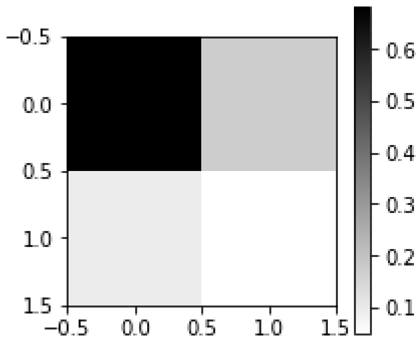
In [19]:

```
# roc curve for knn with different k choices
klist = [1, 3, 5, 10, 20, 100]
X_train, X_test, y_train, y_test = train_test_split(features, credit, stratify = credit)
acc_train, f1_train, auc_train = np.zeros(len(klist)), np.zeros(len(klist)), np.zeros(len(klist
))
acc_test, f1_test, auc_test = np.zeros(len(klist)), np.zeros(len(klist)), np.zeros(len(klist))
y_prob = []
for i, k in enumerate(klist):
    print('kNN k = %d'%k)
    clf = KNeighborsClassifier(n_neighbors=k)
    _y_prob, metric_train, metric_test, t = train_predict(clf, X_train, y_train, X_test, y_test)
    acc_train[i], f1_train[i], auc_train[i] = metric_train
    acc_test[i], f1_test[i], auc_test[i] = metric_test

    y_prob.append(_y_prob)

print('*******model complexity curve!!!*********')
plt.figure()
plt.plot(klist, auc_train, '*-')
plt.plot(klist, auc_test, 'o-')
plt.plot(klist, auc_test - auc_train)
plt.xlabel('k')
plt.legend(['auc score (training)', 'auc score (testing)', 'auc score (testing - training)'])

plt.figure()
plt.plot(klist, acc_train, '*-')
plt.plot(klist, acc_test, 'o-')
plt.plot(klist, acc_test - acc_train)
plt.xlabel('k')
plt.legend(['acc score (training)', 'acc score (testing)', 'acc score (testing - training)'])

# ROC curve of different k !!!
plt.figure()
for _y_prob in y_prob:
    fpr, tpr, thresholds = roc_curve(y_test, _y_prob)
    plt.plot(fpr, tpr,marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend([str(k) for k in klist], loc=0, fontsize='small')
plt.title('k nearest neighbors')
plt.show()
```

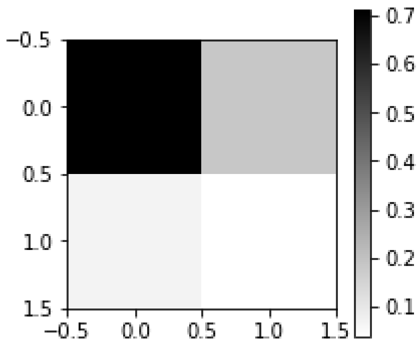
```
kNN k = 1
Building classifier: KNeighborsClassifier....
 train set: acc=1.0000; f1=1.0000; auc=1.0000
 test set: acc=0.6815; f1=0.2896; auc=0.5426
 test confusion martix:
[[4624 1217]
 [1172  487]]
```



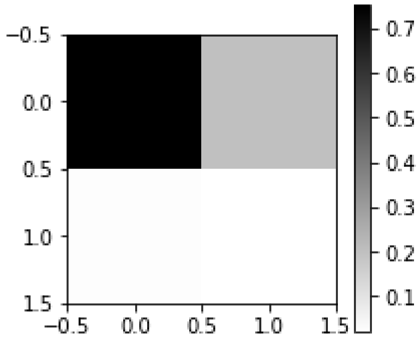
```
kNN k = 3
Building classifier: KNeighborsClassifier....
 train set: acc=0.8442; f1=0.5718; auc=0.8851
 test set: acc=0.7296; f1=0.2604; auc=0.5762
 test confusion martix:
[[5115  726]
 [1302  357]]
```



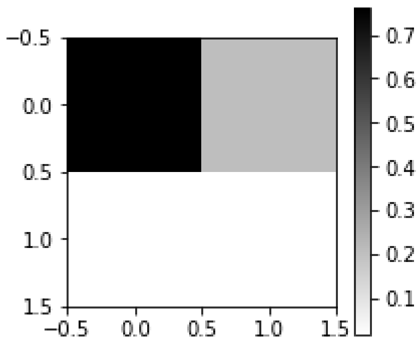
```
kNN k = 5
Building classifier: KNeighborsClassifier....
 train set: acc=0.8143; f1=0.4402; auc=0.8303
 test set: acc=0.7468; f1=0.2258; auc=0.5939
 test confusion martix:
[[5324  517]
 [1382  277]]
```



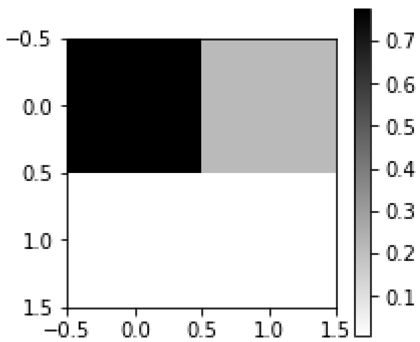
```
kNN k = 10
Building classifier: KNeighborsClassifier....
 train set: acc=0.7936; f1=0.2242; auc=0.7739
 test set: acc=0.7720; f1=0.1467; auc=0.6218
 test confusion martix:
[[5643  198]
 [1512  147]]
```



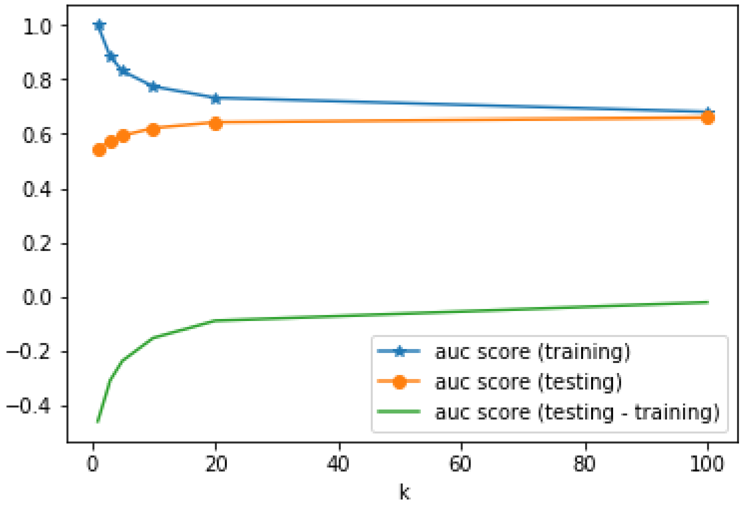
```
kNN k = 20
Building classifier: KNeighborsClassifier....
 train set: acc=0.7865; f1=0.1581; auc=0.7318
 test set: acc=0.7761; f1=0.1168; auc=0.6434
 test confusion martix:
[[5710  131]
 [1548  111]]
```

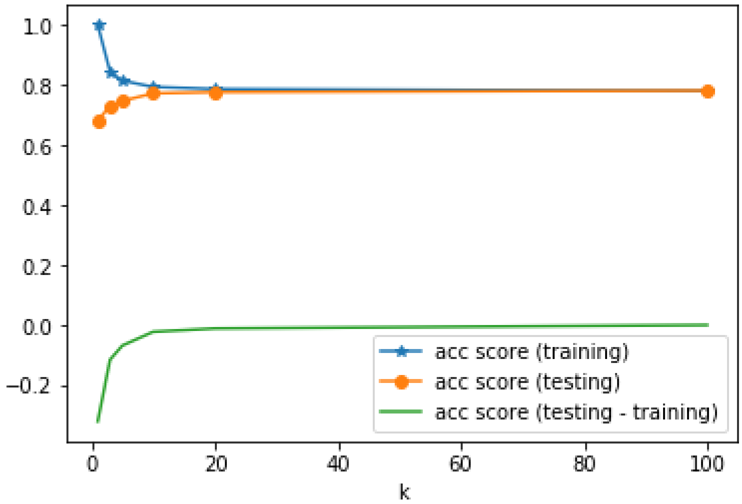


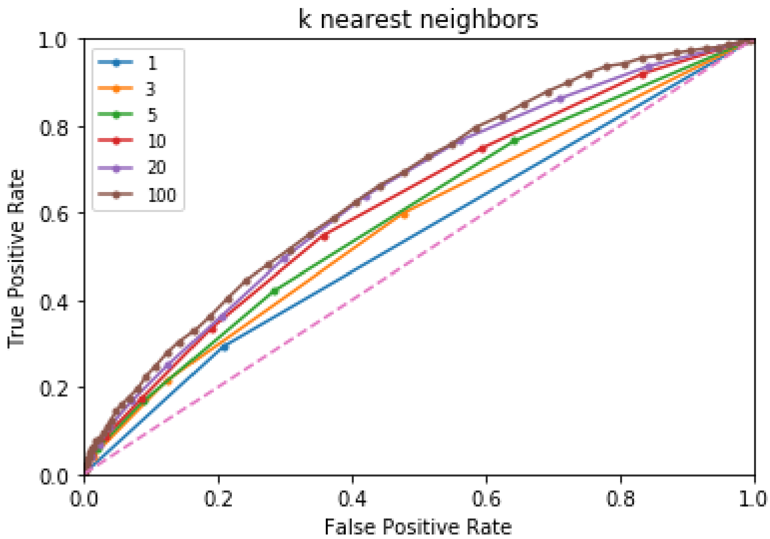
```
kNN k = 100
Building classifier: KNeighborsClassifier....
 train set: acc=0.7804; f1=0.0547; auc=0.6801
 test set: acc=0.7813; f1=0.0650; auc=0.6597
 test confusion martix:
[[5803   38]
 [1602   57]]
```



```
*******model complexity curve!!!*********
```


auc score (training)
auc score (testing)
auc score (testing - training)
k


acc score (training)
acc score (testing)
acc score (testing - training)
k


k nearest neighbors
1
3
5
10
20
100
True Positive Rate
False Positive Rate

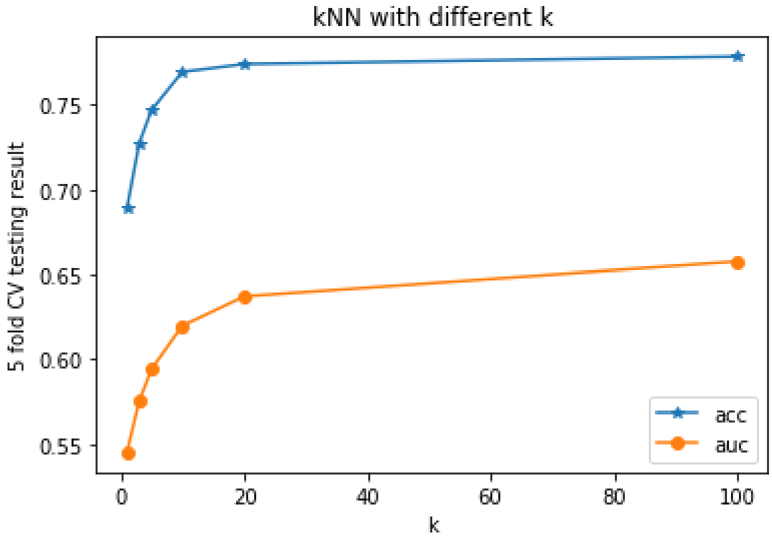
In [23]:

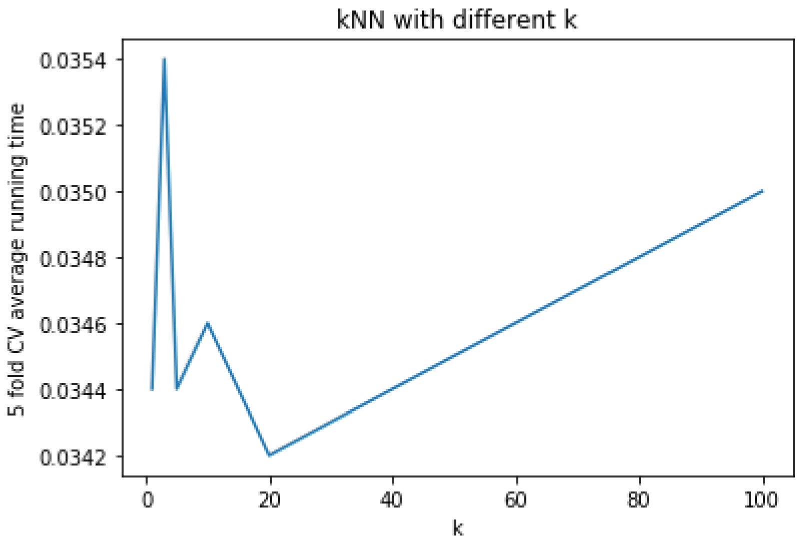
```
fold, n_fold = 0, 5
acc, f1, auc = np.zeros((len(klist), n_fold)), np.zeros((len(klist), n_fold)), np.zeros((len(kli
st), n_fold))
running_time = np.zeros((len(klist), n_fold))
for train_idx, test_idx in StratifiedKFold(n_splits=n_fold).split(features, credit):
    print('*** fold %d/%d'%(fold+1, n_fold))
    X_train, X_test = features[train_idx, :], features[test_idx, :]
    y_train, y_test = credit[train_idx], credit[test_idx]
    for i, k in enumerate(klist):
        _, train_metric, test_metric, t = train_predict(KNeighborsClassifier(n_neighbors=k), X_t
rain, y_train, X_test, y_test, silent=True)
        acc[i, fold], f1[i, fold], auc[i, fold] = test_metric
        running_time[i, fold] = t
    fold += 1
acc, f1, auc = np.mean(acc, axis=1), np.mean(f1, axis=1), np.mean(auc, axis=1)
running_time = np.mean(running_time, axis=1)

print('5 fold CV result of kNN')
for i, k in enumerate(klist):
    print(' k=%d, acc=%.4f; f1=%.4f; auc=%.4f'%(k, acc[i], f1[i], auc[i]))

print('*******kNN learning curve!!!*********')
plt.figure()
plt.plot(klist, acc, '*-')
plt.plot(klist, auc, 'o-')
plt.xlabel('k')
plt.legend(['acc', 'auc'])
plt.ylabel('5 fold CV testing result')
plt.title('kNN with different k')

plt.figure()
plt.plot(klist, running_time)
plt.xlabel('k')
plt.ylabel('5 fold CV average running time')
plt.title('kNN with different k')
```

```
5 fold CV result of kNN
 k=1, acc=0.6887; f1=0.2881; auc=0.5453
 k=3, acc=0.7272; f1=0.2487; auc=0.5755
 k=5, acc=0.7467; f1=0.2314; auc=0.5943
 k=10, acc=0.7693; f1=0.1439; auc=0.6200
 k=20, acc=0.7739; f1=0.1174; auc=0.6372
 k=100, acc=0.7783; f1=0.0589; auc=0.6576
*******kNN learning curve!!!*********
```

Out[23]:

```
Text(0.5,1,'kNN with different k')
```





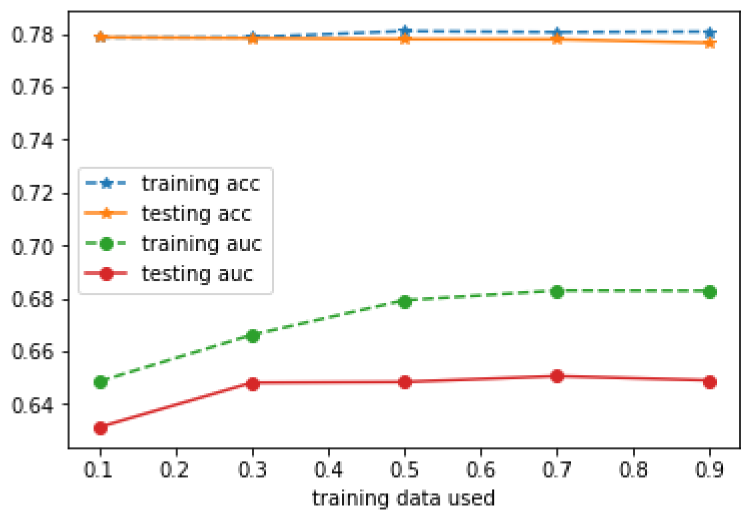
In [26]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
nfold = 5
clf = KNeighborsClassifier(n_neighbors=100) # optimal model

train_acc, train_f1, train_auc = np.zeros(len(train_size)), np.zeros(len(train_size)), np.zeros(
len(train_size))
test_acc, test_f1, test_auc = np.zeros(len(train_size)), np.zeros(len(train_size)), np.zeros(len
(train_size))

for i, size in enumerate(train_size):
    X_train, X_test, y_train, y_test = train_test_split(features, credit, test_size = 1-size, st
ratify = credit)
    _, train_metric, test_metric, t = train_predict(clf, X_train, y_train, X_test, y_test, silen
t=True)
    train_acc[i], train_f1[i], train_auc[i] = train_metric
    test_acc[i], test_f1[i], test_auc[i] = test_metric

plt.figure()
plt.plot(train_size, train_acc, '*--')
plt.plot(train_size, test_acc, '*-')
plt.plot(train_size, train_auc, 'o--')
plt.plot(train_size, test_auc, 'o-')
plt.xlabel('training data used')
plt.legend(['training acc', 'testing acc', 'training auc', 'testing auc'])
```

```
d:\python27\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMet
ricWarning: F-score is ill-defined and being set to 0.0 due to no predicted sample
s.
  'precision', 'predicted', average, warn_for)
```

Out[26]:

```
<matplotlib.legend.Legend at 0x15e44210>
```



finetune parameters for neural network

a single hidden layer is enough, for the number of features is not too large.

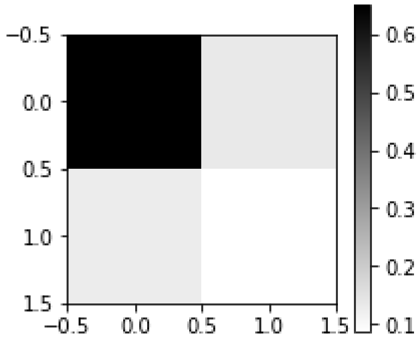
The number of the hidden neurons n = 10, 20, 50, 100

Conclusion: according to the results below, we set optimal n to be 50.

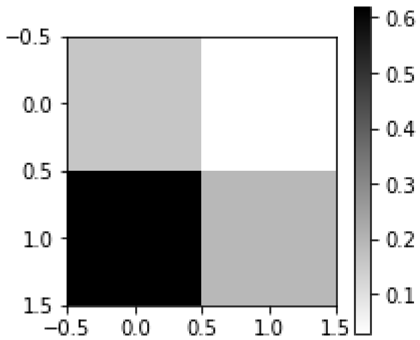
In [24]:

```
nlist = [10, 20, 50, 100]
X_train, X_test, y_train, y_test = train_test_split(features, credit, stratify = credit)
acc_train, f1_train, auc_train = np.zeros(len(nlist)), np.zeros(len(nlist)), np.zeros(len(nlist
))
acc_test, f1_test, auc_test = np.zeros(len(nlist)), np.zeros(len(nlist)), np.zeros(len(nlist))
y_prob = []
for i, n in enumerate(nlist):
    print(' neural networks (single hidden layer) n = %d'%n)
    clf = MLPClassifier(hidden_layer_sizes=n)
    _y_prob, metric_train, metric_test, t = train_predict(clf, X_train, y_train, X_test, y_test)
    acc_train[i], f1_train[i], auc_train[i] = metric_train
    acc_test[i], f1_test[i], auc_test[i] = metric_test

    y_prob.append(_y_prob)

print('******model complexity curve!!!*********')
plt.figure()
plt.plot(nlist, auc_train, '*-')
plt.plot(nlist, auc_test, 'o-')
plt.plot(nlist, auc_test - auc_train)
plt.xlabel('n')
plt.legend(['auc score (training)', 'auc score (testing)', 'auc score (testing - training)'])

plt.figure()
plt.plot(nlist, acc_train, '*-')
plt.plot(nlist, acc_test, 'o-')
plt.plot(nlist, acc_test - acc_train)
plt.xlabel('n')
plt.legend(['acc score (training)', 'acc score (testing)', 'acc score (testing - training)'])

# ROC curve of different k !!!
plt.figure()
for _y_prob in y_prob:
    fpr, tpr, thresholds = roc_curve(y_test, _y_prob)
    while len(fpr) > 100:
        fpr = [fpr[i] for i in range(len(fpr)) if i % 2 == 0]
        tpr = [tpr[i] for i in range(len(tpr)) if i % 2 == 0]
    plt.plot(fpr, tpr,marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend([str(n) for n in nlist], loc=0, fontsize='small')
plt.title('neural networks')
plt.show()
```

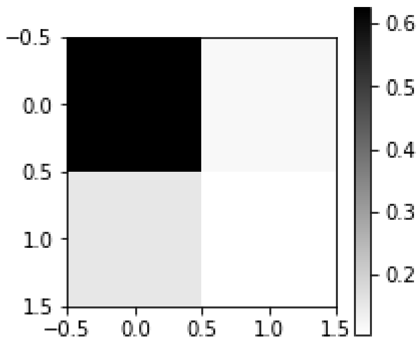
```
 neural networks (single hidden layer) n = 10
Building classifier: MLPClassifier....
 train set: acc=0.7441; f1=0.4042; auc=0.6656
 test set: acc=0.7367; f1=0.3951; auc=0.6530
 test confusion martix:
[[4880  961]
 [1014  645]]
```



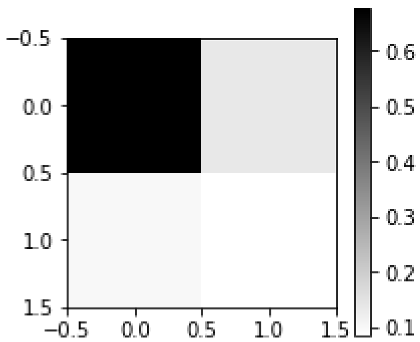

```
 neural networks (single hidden layer) n = 20
Building classifier: MLPClassifier....
 train set: acc=0.3626; f1=0.3837; auc=0.6003
 test set: acc=0.3535; f1=0.3747; auc=0.5886
 test confusion martix:
[[1198 4643]
 [ 206 1453]]
```



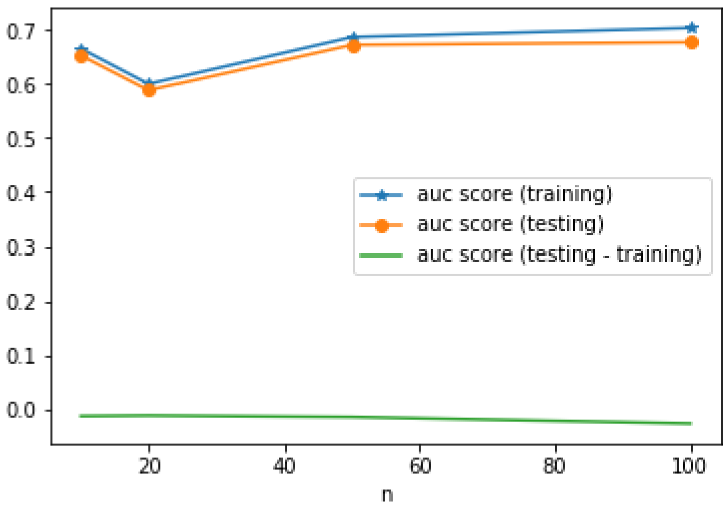

```
 neural networks (single hidden layer) n = 50
Building classifier: MLPClassifier....
 train set: acc=0.7361; f1=0.4340; auc=0.6870
 test set: acc=0.7280; f1=0.4311; auc=0.6725
 test confusion martix:
[[4687 1154]
 [ 886  773]]
```

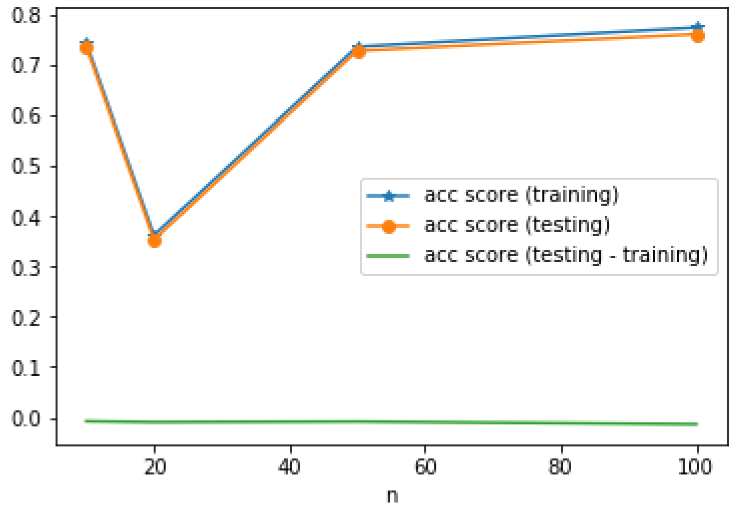


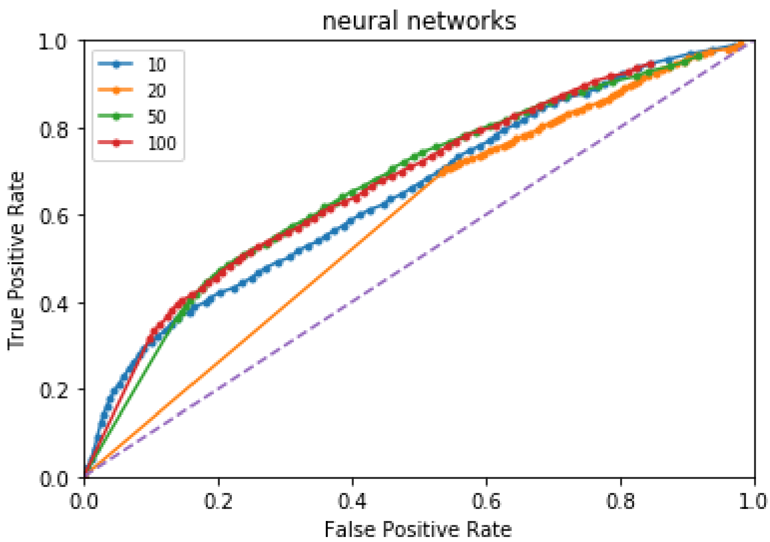

```
 neural networks (single hidden layer) n = 100
Building classifier: MLPClassifier....
 train set: acc=0.7744; f1=0.4379; auc=0.7038
 test set: acc=0.7611; f1=0.4117; auc=0.6773
 test confusion martix:
[[5081  760]
 [1032  627]]
```




```
*******model complexity curve!!!*********
```




0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
20
40
60
80
100
n
acc score (training)
acc score (testing)
acc score (testing - training)


neural networks
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
10
20
50
100

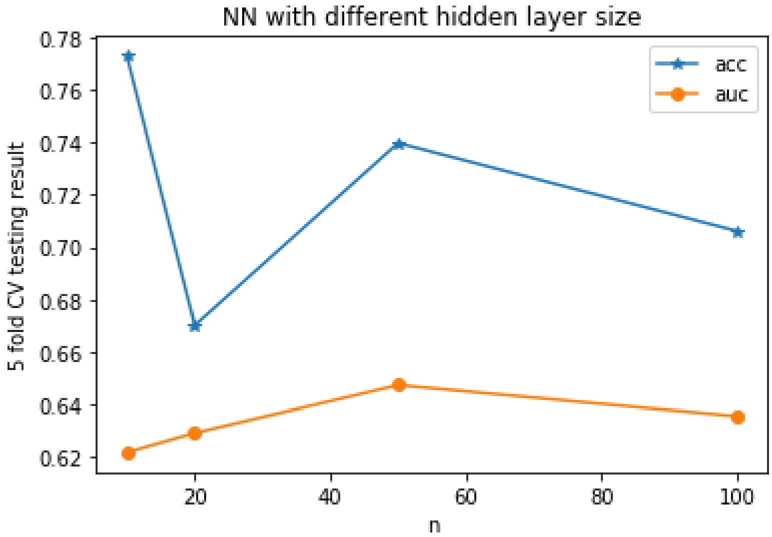
In [25]:

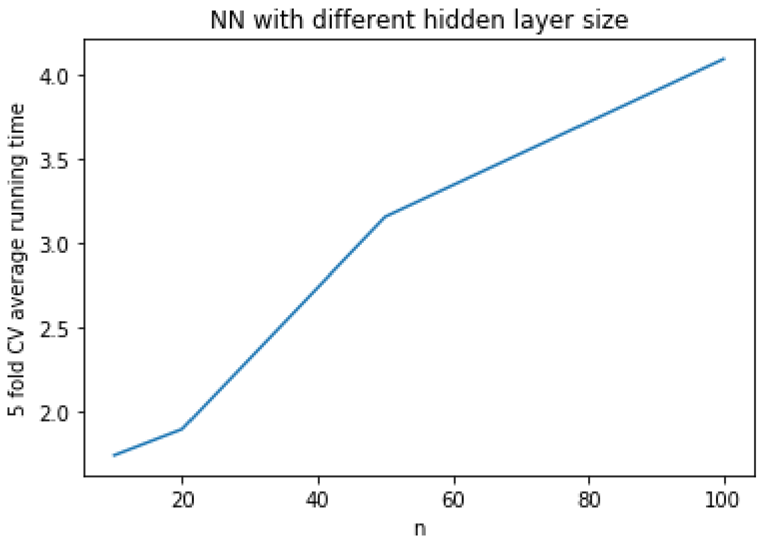
```
fold, n_fold = 0, 5
acc, f1, auc = np.zeros((len(nlist), n_fold)), np.zeros((len(nlist), n_fold)), np.zeros((len(nli
st), n_fold))
running_time = np.zeros((len(nlist), n_fold))
for train_idx, test_idx in StratifiedKFold(n_splits=n_fold).split(features, credit):
    print('*** fold %d/%d'%(fold+1, n_fold))
    X_train, X_test = features[train_idx, :], features[test_idx, :]
    y_train, y_test = credit[train_idx], credit[test_idx]
    for i, n in enumerate(nlist):
        _, train_metric, test_metric, t = train_predict(MLPClassifier(hidden_layer_sizes=n), X_t
rain, y_train, X_test, y_test, silent=True)
        acc[i, fold], f1[i, fold], auc[i, fold] = test_metric
        running_time[i, fold] = t
    fold += 1
acc, f1, auc = np.mean(acc, axis=1), np.mean(f1, axis=1), np.mean(auc, axis=1)
running_time = np.mean(running_time, axis=1)

print('5 fold CV result of neural network')
for i, n in enumerate(nlist):
    print(' n=%d, acc=%.4f; f1=%.4f; auc=%.4f'%(n, acc[i], f1[i], auc[i]))

print('******neural network learning curve!!!********')
plt.figure()
plt.plot(nlist, acc, '*-')
plt.plot(nlist, auc, 'o-')
plt.xlabel('n')
plt.legend(['acc', 'auc'])
plt.ylabel('5 fold CV testing result')
plt.title('NN with different hidden layer size')

plt.figure()
plt.plot(nlist, running_time)
plt.xlabel('n')
plt.ylabel('5 fold CV average running time')
plt.title('NN with different hidden layer size')
```

```
*** fold 1/5
*** fold 2/5
*** fold 3/5
*** fold 4/5
*** fold 5/5
5 fold CV result of neural network
 n=10, acc=0.7730; f1=0.1305; auc=0.6216
 n=20, acc=0.6702; f1=0.3633; auc=0.6289
 n=50, acc=0.7399; f1=0.2956; auc=0.6472
 n=100, acc=0.7062; f1=0.2989; auc=0.6353
*******neural network learning curve!!!*********
```

Out[25]:

```
Text(0.5,1,'NN with different hidden layer size')
```



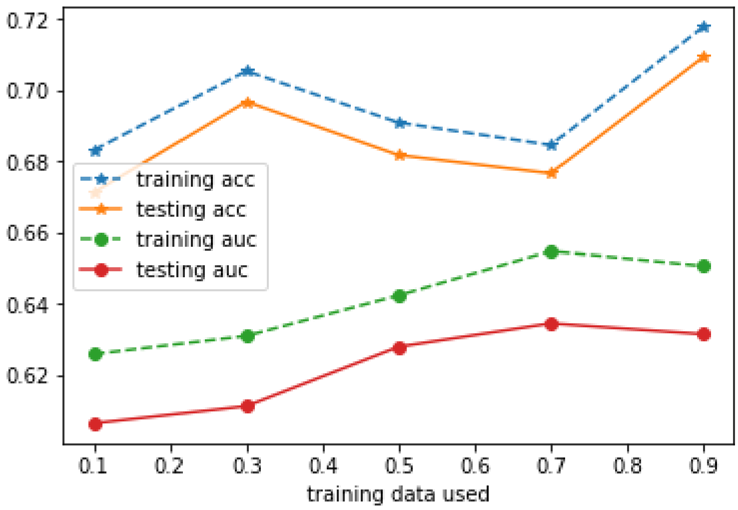


In [32]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
nfold = 5
clf = MLPClassifier(hidden_layer_sizes=50, max_iter=int(1e5)) # optimal model

train_acc, train_f1, train_auc = np.zeros([len(train_size),10]), np.zeros([len(train_size),10]),
np.zeros([len(train_size),10])
test_acc, test_f1, test_auc = np.zeros([len(train_size),10]), np.zeros([len(train_size),10]), np
.zeros([len(train_size),10])

for j in range(10):
    for i, size in enumerate(train_size):
        X_train, X_test, y_train, y_test = train_test_split(features, credit, test_size = 1-size
, stratify = credit)
        _, train_metric, test_metric, t = train_predict(clf, X_train, y_train, X_test, y_test, s
ilent=True)
        train_acc[i,j], train_f1[i,j], train_auc[i,j] = train_metric
        test_acc[i,j], test_f1[i,j], test_auc[i,j] = test_metric

train_acc, train_f1, train_auc = np.mean(train_acc, axis=1), np.mean(train_f1, axis=1), np.mean(
train_auc, axis=1)
test_acc, test_f1, test_auc = np.mean(test_acc, axis=1), np.mean(test_f1, axis=1), np.mean(test_
auc, axis=1)
plt.figure()
plt.plot(train_size, train_acc, '*--')
plt.plot(train_size, test_acc, '*-')
plt.plot(train_size, train_auc, 'o--')
plt.plot(train_size, test_auc, 'o-')
plt.xlabel('training data used')
plt.legend(['training acc', 'testing acc', 'training auc', 'testing auc'])
```

Out[32]:

<matplotlib.legend.Legend at 0x1673a8d0>



finetune parameters for decision tree

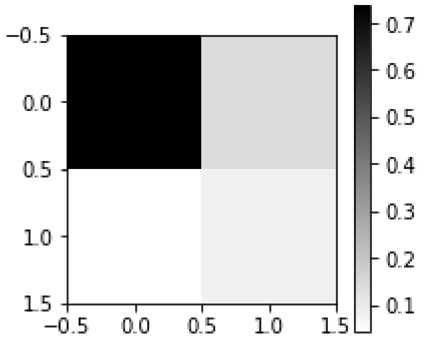
max_depth to be 3, 5, 10, 20, 30

Conclusion: according to the results below, we set the optimal depth to be 5

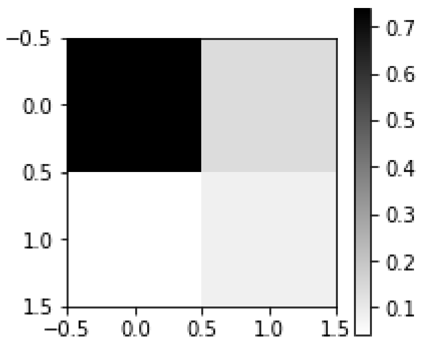
In [35]:

```
depth = [3, 5, 10, 20, 30]
X_train, X_test, y_train, y_test = train_test_split(features, credit, stratify = credit)
acc_train, f1_train, auc_train = np.zeros(len(depth)), np.zeros(len(depth)), np.zeros(len(depth
))
acc_test, f1_test, auc_test = np.zeros(len(depth)), np.zeros(len(depth)), np.zeros(len(depth))
y_prob = []
for i, d in enumerate(depth):
    print(' tree with depth = %d'%d)
    clf = DecisionTreeClassifier(max_depth=d)
    _y_prob, metric_train, metric_test, t = train_predict(clf, X_train, y_train, X_test, y_test)

    acc_train[i], f1_train[i], auc_train[i] = metric_train
    acc_test[i], f1_test[i], auc_test[i] = metric_test

    y_prob.append(_y_prob)

print('*******model complexity curve!!!*********')
plt.figure()
plt.plot(depth, auc_train, '*-')
plt.plot(depth, auc_test, 'o-')
plt.plot(depth, auc_test - auc_train)
plt.xlabel('max depth')
plt.legend(['auc score (training)', 'auc score (testing)', 'auc score (testing - training)'])

plt.figure()
plt.plot(depth, acc_train, '*-')
plt.plot(depth, acc_test, 'o-')
plt.plot(depth, acc_test - acc_train)
plt.xlabel('max depth')
plt.legend(['acc score (training)', 'acc score (testing)', 'acc score (testing - training)'])

# ROC curve of different k !!!
plt.figure()
for _y_prob in y_prob:
    fpr, tpr, thresholds = roc_curve(y_test, _y_prob)
    while len(fpr) > 100:
        fpr = [fpr[i] for i in range(len(fpr)) if i % 2 == 0]
        tpr = [tpr[i] for i in range(len(tpr)) if i % 2 == 0]
    plt.plot(fpr, tpr,marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend([str(d) for d in depth], loc=0, fontsize='small')
plt.title('decision tree')
plt.show()
```

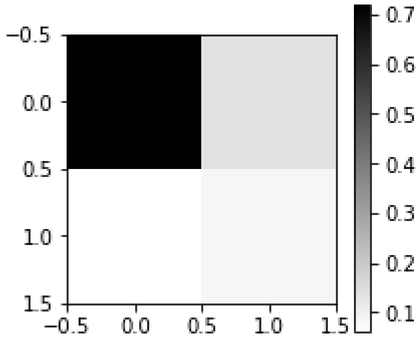
```
 tree with depth = 3
Building classifier: DecisionTreeClassifier....
 train set: acc=0.8219; f1=0.4765; auc=0.7362
 test set: acc=0.8209; f1=0.4821; auc=0.7353
 test confusion martix:
[[5532  309]
 [1034  625]]
```



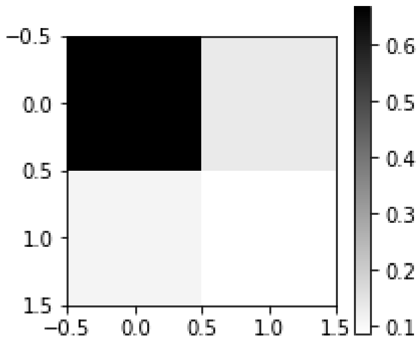

```
 tree with depth = 5
Building classifier: DecisionTreeClassifier....
 train set: acc=0.8249; f1=0.4846; auc=0.7672
 test set: acc=0.8211; f1=0.4807; auc=0.7517
 test confusion martix:
[[5537  304]
 [1038  621]]
```



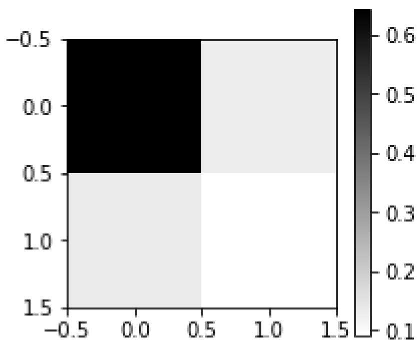

```
 tree with depth = 10
Building classifier: DecisionTreeClassifier....
 train set: acc=0.8544; f1=0.5994; auc=0.8365
 test set: acc=0.8027; f1=0.4602; auc=0.7180
 test confusion martix:
[[5389  452]
 [1028  631]]
```



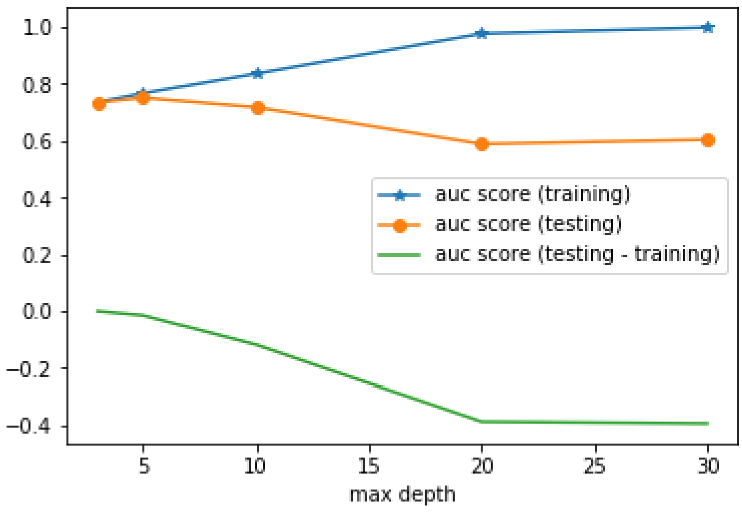

```
 tree with depth = 20
Building classifier：DecisionTreeClassifier....
 train set: acc=0.9561; f1=0.8931; auc=0.9775
 test set: acc=0.7533; f1=0.4097; auc=0.5895
 test confusion martix:
[[5008  833]
 [1017  642]]
```



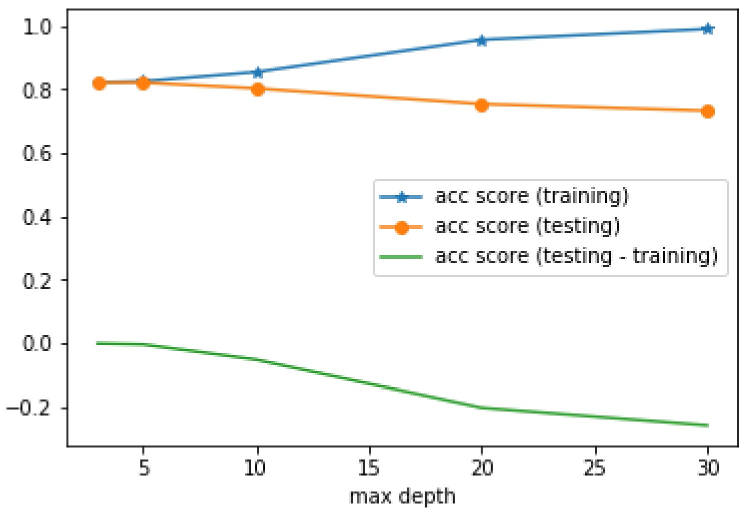

```
 tree with depth = 30
Building classifier：DecisionTreeClassifier....
 train set: acc=0.9901; f1=0.9773; auc=0.9989
 test set: acc=0.7321; f1=0.4033; auc=0.6048
 test confusion martix:
[[4812 1029]
 [ 980  679]]
```



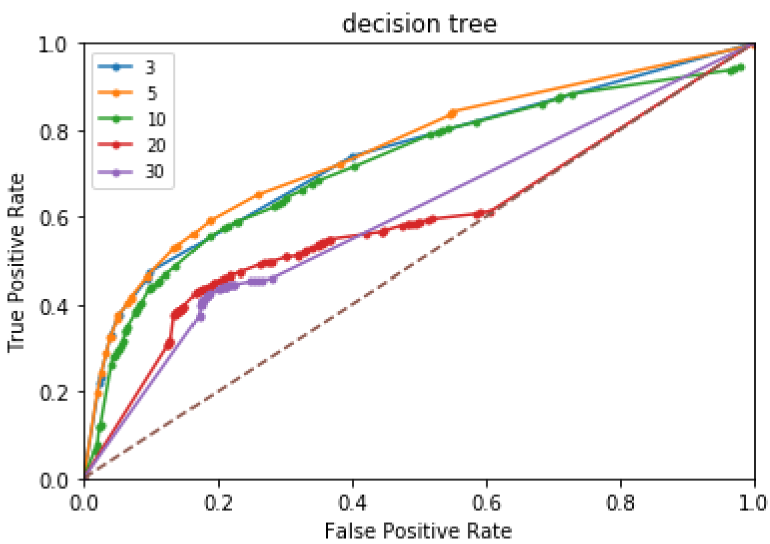

```
*******model complexity curve!!!*********
```



1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
5
10
15
20
25
30
max depth
auc score (training)
auc score (testing)
auc score (testing - training)



1.0
0.8
0.6
0.4
0.2
0.0
−0.2
5
10
15
20
25
30
max depth
acc score (training)
acc score (testing)
acc score (testing - training)



decision tree
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
3
5
10
20
30

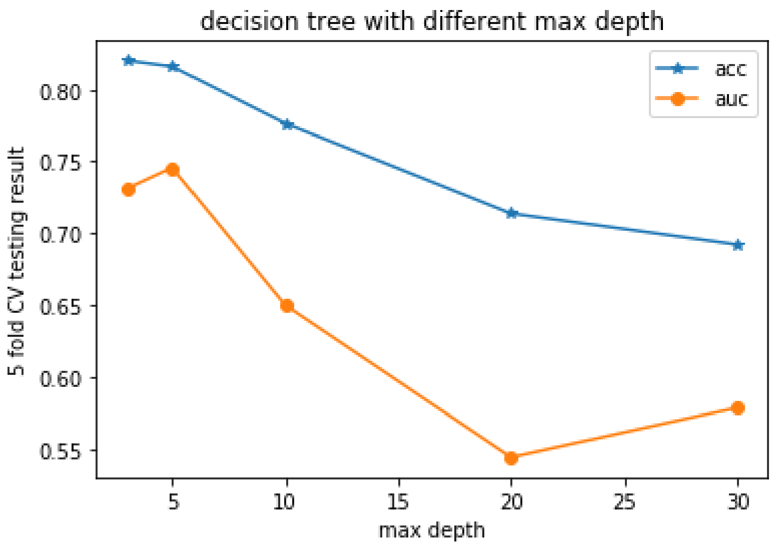
In [37]:

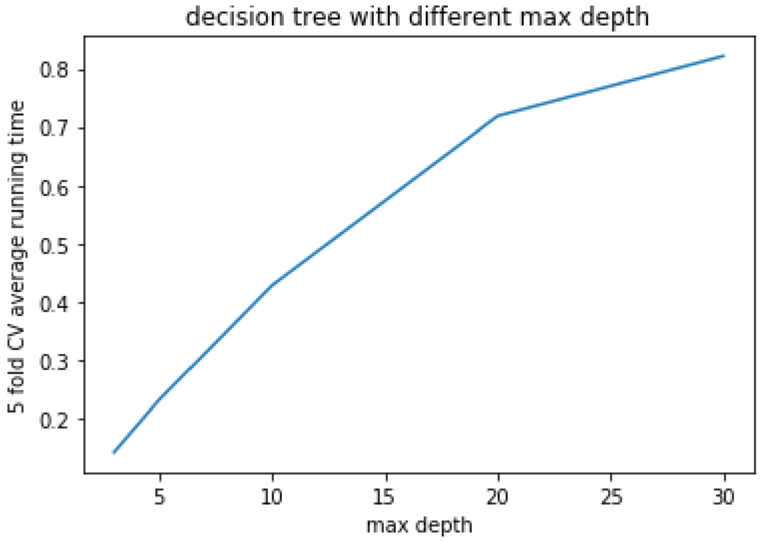
```
fold, n_fold = 0, 5
acc, f1, auc = np.zeros((len(depth), n_fold)), np.zeros((len(depth), n_fold)), np.zeros((len(dep
th), n_fold))
running_time = np.zeros((len(depth), n_fold))
for train_idx, test_idx in StratifiedKFold(n_splits=n_fold).split(features, credit):
    print('*** fold %d/%d'%(fold+1, n_fold))
    X_train, X_test = features[train_idx, :], features[test_idx, :]
    y_train, y_test = credit[train_idx], credit[test_idx]
    for i, d in enumerate(depth):
        _, train_metric, test_metric, t = train_predict(DecisionTreeClassifier(max_depth=d), X_t
rain, y_train, X_test, y_test, silent=True)
        acc[i, fold], f1[i, fold], auc[i, fold] = test_metric
        running_time[i, fold] = t
    fold += 1
acc, f1, auc = np.mean(acc, axis=1), np.mean(f1, axis=1), np.mean(auc, axis=1)
running_time = np.mean(running_time, axis=1)

print('5 fold CV result of decision tree')
for i, d in enumerate(depth):
    print(' max_depth=%d, acc=%.4f; f1=%.4f; auc=%.4f'%(d, acc[i], f1[i], auc[i]))

print('*******decision tree learning curve!!!*********')
plt.figure()
plt.plot(depth, acc, '*-')
plt.plot(depth, auc, 'o-')
plt.xlabel('max depth')
plt.legend(['acc', 'auc'])
plt.ylabel('5 fold CV testing result')
plt.title('decision tree with different max depth')

plt.figure()
plt.plot(depth, running_time)
plt.xlabel('max depth')
plt.ylabel('5 fold CV average running time')
plt.title('decision tree with different max depth')
```

```
*** fold 1/5
*** fold 2/5
*** fold 3/5
*** fold 4/5
*** fold 5/5
5 fold CV result of decision tree
 max_depth=3, acc=0.8201; f1=0.4789; auc=0.7312
 max_depth=5, acc=0.8160; f1=0.4430; auc=0.7456
 max_depth=10, acc=0.7765; f1=0.3893; auc=0.6499
 max_depth=20, acc=0.7138; f1=0.3624; auc=0.5441
 max_depth=30, acc=0.6922; f1=0.3581; auc=0.5785
*******decision tree learning curve!!!*********
```

Out[37]:

```
Text(0.5,1,'decision tree with different max depth')
```





for the tree with max-depth=5, there is no need to post prune it. So here I try to prune a tree with max_depth=20.

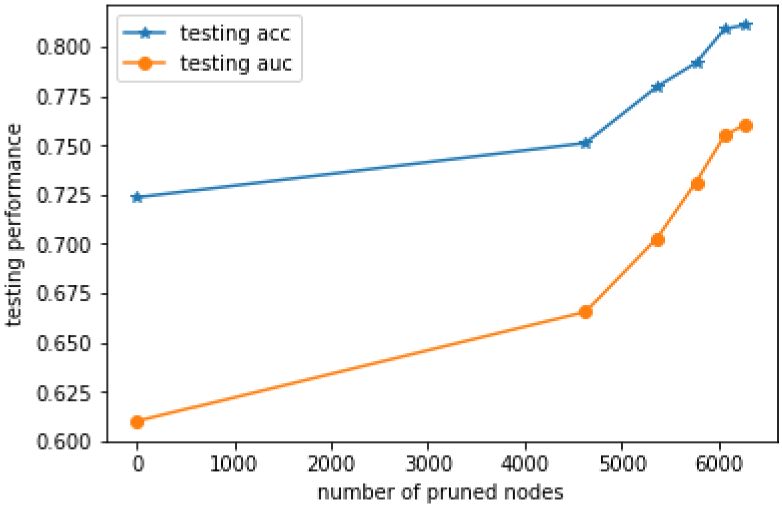
In [85]:

```
from sklearn.tree._tree import TREE_LEAF

def prune(decisiontree, min_samples_leaf = 1):
    if decisiontree.min_samples_leaf >= min_samples_leaf:
        print('Tree already more pruned')
    else:
        decisiontree.min_samples_leaf = min_samples_leaf
        tree = decisiontree.tree_
        n_prune = 0
        for i in range(tree.node_count):
            n_samples = tree.n_node_samples[i]
            if n_samples <= min_samples_leaf:
                n_prune += 1
                tree.children_left[i]=-1
                tree.children_right[i]=-1

        print('prune %d nodes'%n_prune)
        return n_prune

X_train, X_test, y_train, y_test = train_test_split(features, credit, stratify = credit)
clf = DecisionTreeClassifier(max_depth=50)
clf.fit(X_train, y_train)
y_prob = clf.predict_proba(X_test)
y_pred = np.argmax(y_prob, axis=1)
acc_test_full, f1_test_full, auc_test_full = accuracy_score(y_test, y_pred), f1_score(y_test, y_
pred), roc_auc_score(y_test, y_prob[:,1])
print(' test set: acc=%.4f; f1=%.4f; auc=%.4f'%(acc_test_full, f1_test_full, auc_test_full))

prune_threshold = [20, 50, 100, 200, 500, 100]
acc_test, f1_test, auc_test = np.zeros(len(prune_threshold)), np.zeros(len(prune_threshold)), np
.zeros(len(prune_threshold))
n_prune = []
for i, thres in enumerate(prune_threshold):
    n_prune.append(prune(clf, thres))
    y_prob = clf.predict_proba(X_test)
    y_pred = np.argmax(y_prob, axis=1)
    acc_test[i], f1_test[i], auc_test[i] = accuracy_score(y_test, y_pred), f1_score(y_test, y_pr
ed), roc_auc_score(y_test, y_prob[:,1])

n_prune.insert(0, 0)
acc_test = list(acc_test)
acc_test.insert(0, acc_test_full)
auc_test = list(auc_test)
auc_test.insert(0, auc_test_full)

plt.plot(n_prune, acc_test, '*-')
plt.plot(n_prune, auc_test, 'o-')
plt.xlabel('number of pruned nodes')
plt.ylabel('testing performance')
plt.legend(['testing acc', 'testing auc'])
```

```
 test set: acc=0.7236; f1=0.3940; auc=0.6100
prune 4628 nodes
prune 5367 nodes
prune 5768 nodes
prune 6064 nodes
prune 6281 nodes
Tree already more pruned
```

Out[85]:

```
<matplotlib.legend.Legend at 0x1622c570>
```

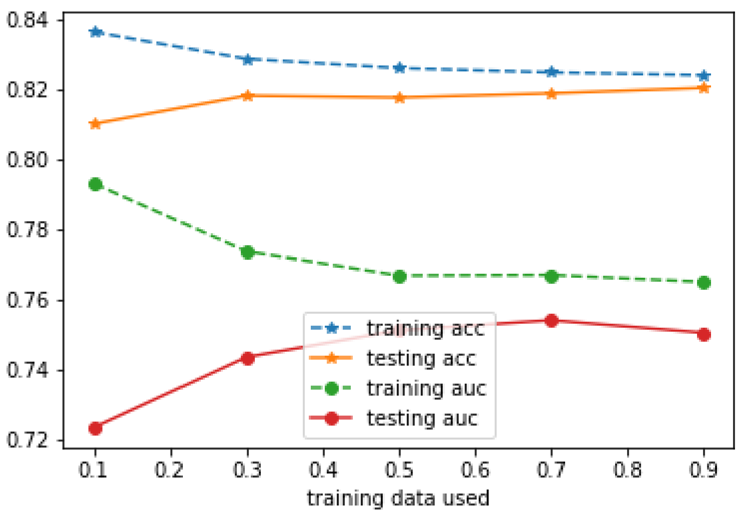


In [95]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
clf = DecisionTreeClassifier(max_depth=5) # optimal model

train_acc, train_f1, train_auc = np.zeros([len(train_size),10]), np.zeros([len(train_size),10]),
np.zeros([len(train_size),10])
test_acc, test_f1, test_auc = np.zeros([len(train_size),10]), np.zeros([len(train_size),10]), np
.zeros([len(train_size),10])

for j in range(10):
    for i, size in enumerate(train_size):
        X_train, X_test, y_train, y_test = train_test_split(features, credit, test_size = 1-size
, stratify = credit)
        _, train_metric, test_metric, t = train_predict(clf, X_train, y_train, X_test, y_test, s
ilent=True)
        train_acc[i,j], train_f1[i,j], train_auc[i,j] = train_metric
        test_acc[i,j], test_f1[i,j], test_auc[i,j] = test_metric

train_acc, train_f1, train_auc = np.mean(train_acc, axis=1), np.mean(train_f1, axis=1), np.mean(
train_auc, axis=1)
test_acc, test_f1, test_auc = np.mean(test_acc, axis=1), np.mean(test_f1, axis=1), np.mean(test_
auc, axis=1)
plt.figure()
plt.plot(train_size, train_acc, '*--')
plt.plot(train_size, test_acc, '*-')
plt.plot(train_size, train_auc, 'o--')
plt.plot(train_size, test_auc, 'o-')
plt.xlabel('training data used')
plt.legend(['training acc', 'testing acc', 'training auc', 'testing auc'])
```

Out[95]:

```
<matplotlib.legend.Legend at 0x1856d830>
```



finetune parameters for boosting

number of base classifiers = 10, 20, 50, 100, 200

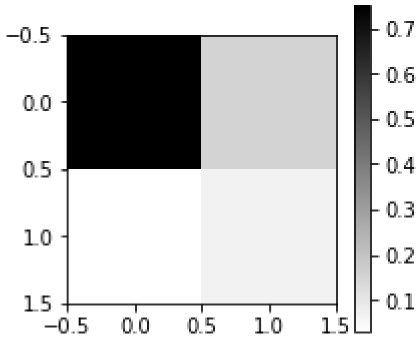
Conclusion: according to the results below, we set the optimal n to be 20

For there is no obvious differences among them, no need to run 5-fold CV here.

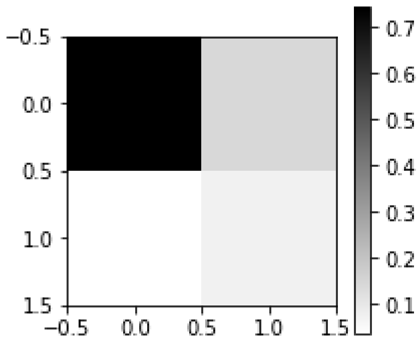
In [88]:

```
nlist = [10, 20, 50, 100, 200]
X_train, X_test, y_train, y_test = train_test_split(features, credit, stratify = credit)
acc_train, f1_train, auc_train = np.zeros(len(nlist)), np.zeros(len(nlist)), np.zeros(len(nlist
))
acc_test, f1_test, auc_test = np.zeros(len(nlist)), np.zeros(len(nlist)), np.zeros(len(nlist))
y_prob = []
for i, n in enumerate(nlist):
    print(' number of base models = %d'%n)
    clf = GradientBoostingClassifier(n_estimators=n)
    _y_prob, metric_train, metric_test, t = train_predict(clf, X_train, y_train, X_test, y_test)

    acc_train[i], f1_train[i], auc_train[i] = metric_train
    acc_test[i], f1_test[i], auc_test[i] = metric_test

    y_prob.append(_y_prob)

print('*******model complexity curve!!!*********')
plt.figure()
plt.plot(nlist, auc_train, '*-')
plt.plot(nlist, auc_test, 'o-')
plt.plot(nlist, auc_test - auc_train)
plt.xlabel('number of base models')
plt.legend(['auc score (training)', 'auc score (testing)', 'auc score (testing - training)'])

plt.figure()
plt.plot(nlist, acc_train, '*-')
plt.plot(nlist, acc_test, 'o-')
plt.plot(nlist, acc_test - acc_train)
plt.xlabel('number of base models')
plt.legend(['acc score (training)', 'acc score (testing)', 'acc score (testing - training)'])

# ROC curve of different k !!!
plt.figure()
for _y_prob in y_prob:
    fpr, tpr, thresholds = roc_curve(y_test, _y_prob)
    while len(fpr) > 100:
        fpr = [fpr[i] for i in range(len(fpr)) if i % 2 == 0]
        tpr = [tpr[i] for i in range(len(tpr)) if i % 2 == 0]
    plt.plot(fpr, tpr,marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend([str(n) for n in nlist], loc=0, fontsize='small')
plt.title('Gradient boosting tree')
plt.show()
```

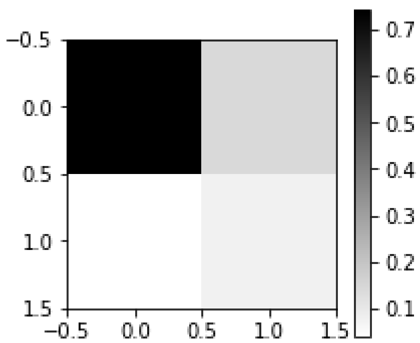
```
 number of base models = 10
Building classifier: GradientBoostingClassifier....
 train set: acc=0.8216; f1=0.4416; auc=0.7773
 test set: acc=0.8179; f1=0.4256; auc=0.7719
 test confusion martix:
[[5628  213]
 [1153  506]]
```



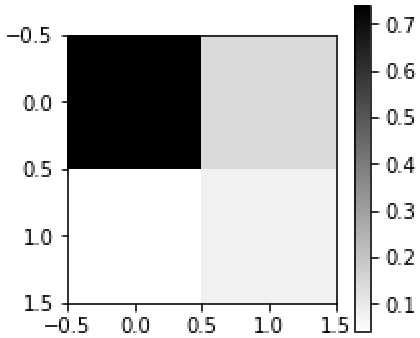

```
 number of base models = 20
Building classifier: GradientBoostingClassifier....
 train set: acc=0.8236; f1=0.4715; auc=0.7840
 test set: acc=0.8195; f1=0.4567; auc=0.7765
 test confusion martix:
[[5577  264]
 [1090  569]]
```



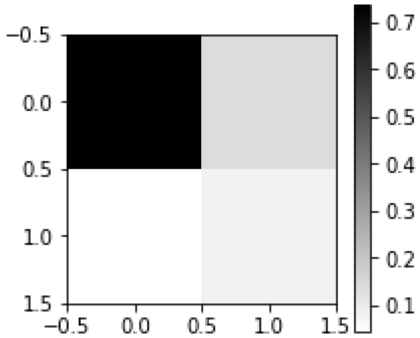

```
 number of base models = 50
Building classifier: GradientBoostingClassifier....
 train set: acc=0.8253; f1=0.4851; auc=0.7992
 test set: acc=0.8189; f1=0.4641; auc=0.7825
 test confusion martix:
[[5554  287]
 [1071  588]]
```



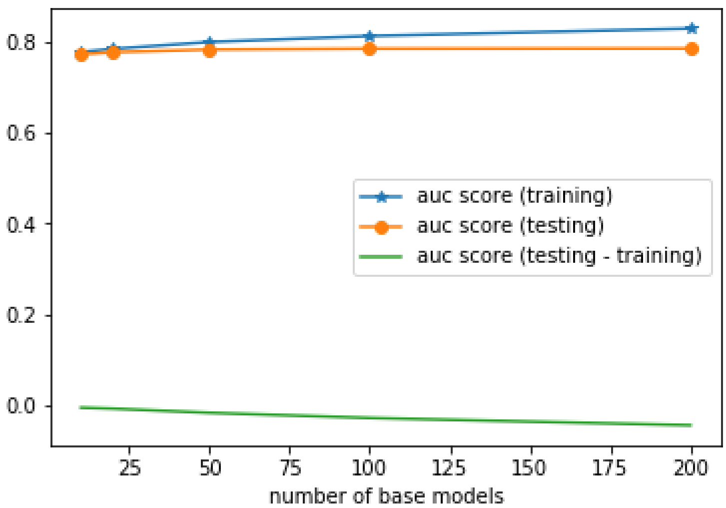
```
 number of base models = 100
Building classifier: GradientBoostingClassifier....
 train set: acc=0.8272; f1=0.4945; auc=0.8125
 test set: acc=0.8185; f1=0.4677; auc=0.7844
 test confusion martix:
[[5541  300]
 [1061  598]]
```

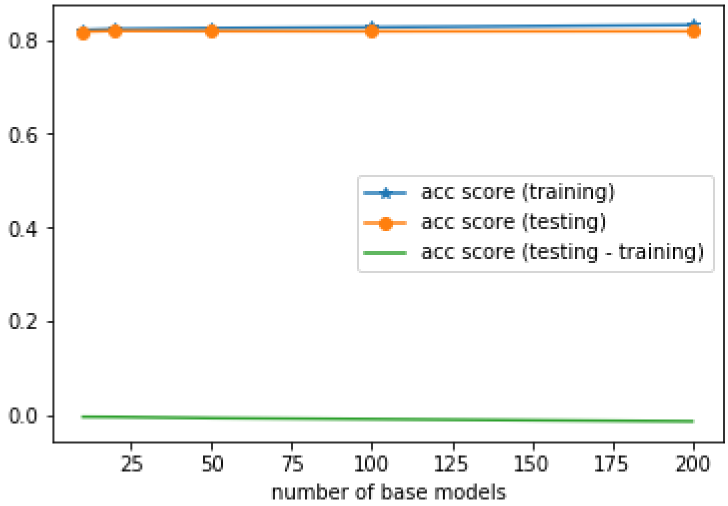



```
 number of base models = 200
Building classifier: GradientBoostingClassifier....
 train set: acc=0.8320; f1=0.5121; auc=0.8290
 test set: acc=0.8187; f1=0.4745; auc=0.7848
 test confusion martix:
[[5526  315]
 [1045  614]]
```

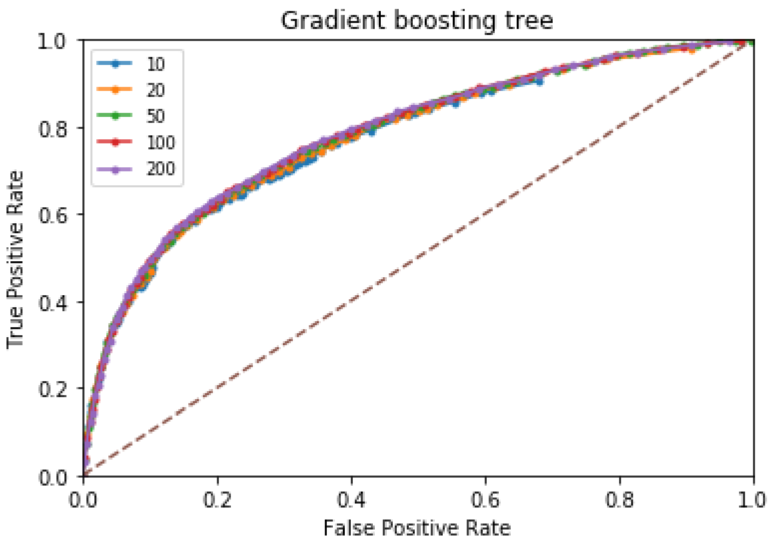



```
*******model complexity curve!!!*********
```





0.8
0.6
0.4
0.2
0.0
25
50
75
100
125
150
175
200
number of base models
acc score (training)
acc score (testing)
acc score (testing - training)



Gradient boosting tree
1.0
0.8
0.6
0.4
0.2
0.0
10
20
50
100
200
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

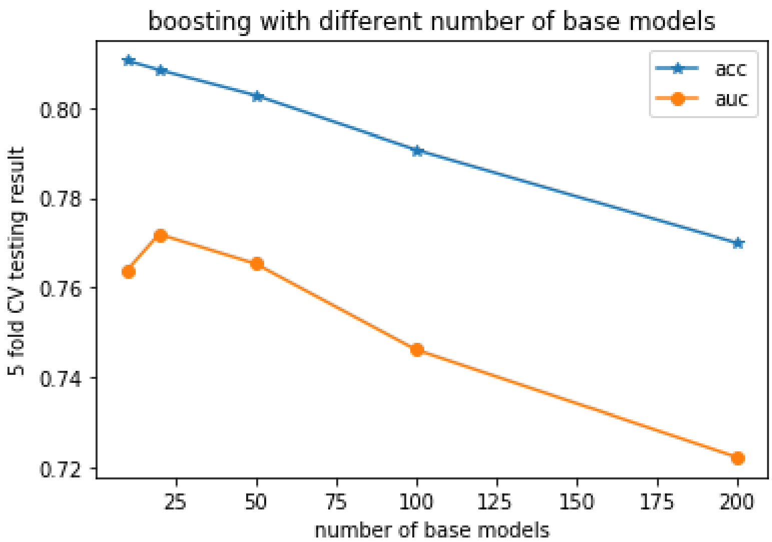
In [90]:

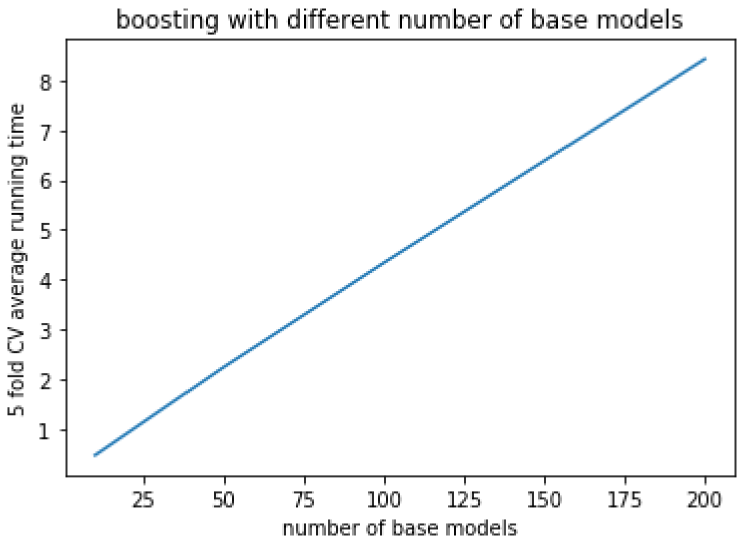
```
fold, n_fold = 0, 5
acc, f1, auc = np.zeros((len(nlist), n_fold)), np.zeros((len(nlist), n_fold)), np.zeros((len(nli
st), n_fold))
running_time = np.zeros((len(nlist), n_fold))
for train_idx, test_idx in StratifiedKFold(n_splits=n_fold).split(features, credit):
    print('*** fold %d/%d'%(fold+1, n_fold))
    X_train, X_test = features[train_idx, :], features[test_idx, :]
    y_train, y_test = credit[train_idx], credit[test_idx]
    for i, n in enumerate(nlist):
        _, train_metric, test_metric, t = train_predict(GradientBoostingClassifier(n_estimators=
n), X_train, y_train, X_test, y_test, silent=True)
        acc[i, fold], f1[i, fold], auc[i, fold] = test_metric
        running_time[i, fold] = t
    fold += 1
acc, f1, auc = np.mean(acc, axis=1), np.mean(f1, axis=1), np.mean(auc, axis=1)
running_time = np.mean(running_time, axis=1)

print('5 fold CV result of boosting')
for i, n in enumerate(nlist):
    print(' number of base models=%d, acc=%.4f; f1=%.4f; auc=%.4f'%(n, acc[i], f1[i], auc[i
]))

print('*******boosting learning curve!!!*********')
plt.figure()
plt.plot(nlist, acc, '*-')
plt.plot(nlist, auc, 'o-')
plt.xlabel('number of base models')
plt.legend(['acc', 'auc'])
plt.ylabel('5 fold CV testing result')
plt.title('boosting with different number of base models')

plt.figure()
plt.plot(nlist, running_time)
plt.xlabel('number of base models')
plt.ylabel('5 fold CV average running time')
plt.title('boosting with different number of base models')
```

```
*** fold 1/5
*** fold 2/5
*** fold 3/5
*** fold 4/5
*** fold 5/5
5 fold CV result of boosting
 number of base models=10, acc=0.8106; f1=0.3470; auc=0.7639
 number of base models=20, acc=0.8085; f1=0.3481; auc=0.7719
 number of base models=50, acc=0.8028; f1=0.3102; auc=0.7653
 number of base models=100, acc=0.7908; f1=0.2605; auc=0.7462
 number of base models=200, acc=0.7700; f1=0.2386; auc=0.7223
*******boosting learning curve!!!*********
```

Out[90]:

```
Text(0.5,1,'boosting with different number of base models')
```



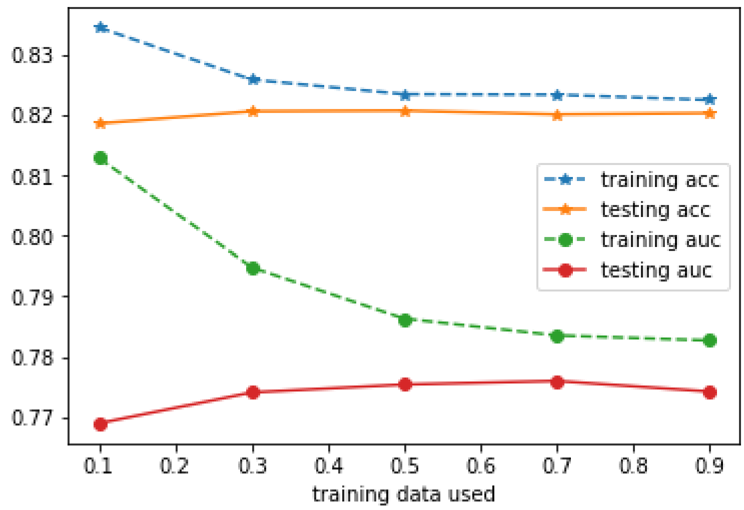


In [96]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
clf =GradientBoostingClassifier(n_estimators=20) # optimal model

train_acc, train_f1, train_auc = np.zeros([len(train_size),10]), np.zeros([len(train_size),10]),
np.zeros([len(train_size),10])
test_acc, test_f1, test_auc = np.zeros([len(train_size),10]), np.zeros([len(train_size),10]), np
.zeros([len(train_size),10])

for j in range(10):
    for i, size in enumerate(train_size):
        X_train, X_test, y_train, y_test = train_test_split(features, credit, test_size = 1-size
, stratify = credit)
        _, train_metric, test_metric, t = train_predict(clf, X_train, y_train, X_test, y_test, s
ilent=True)
        train_acc[i,j], train_f1[i,j], train_auc[i,j] = train_metric
        test_acc[i,j], test_f1[i,j], test_auc[i,j] = test_metric

train_acc, train_f1, train_auc = np.mean(train_acc, axis=1), np.mean(train_f1, axis=1), np.mean(
train_auc, axis=1)
test_acc, test_f1, test_auc = np.mean(test_acc, axis=1), np.mean(test_f1, axis=1), np.mean(test_
auc, axis=1)
plt.figure()
plt.plot(train_size, train_acc, '*--')
plt.plot(train_size, test_acc, '*-')
plt.plot(train_size, train_auc, 'o--')
plt.plot(train_size, test_auc, 'o-')
plt.xlabel('training data used')
plt.legend(['training acc', 'testing acc', 'training auc', 'testing auc'])
```

Out[96]:

<matplotlib.legend.Legend at 0x187c47d0>



finetune parameters for support vector machines

kernel = linear, rbf or polynomial

Conclusion: according to the results below, we set the kernel to be rbf

poor performance for a coarse finetuning, so we move on to finetune the other parameters with linear kernel. And we find out that the optimal C must be 0.01

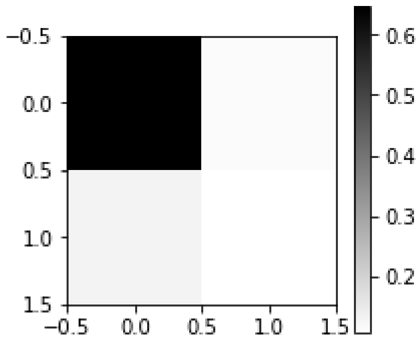
In [91]:

```
from sklearn.preprocessing import StandardScaler
features_svm = StandardScaler().fit_transform(features) # for svm, standard scaler will help imp
rove the performance
svm_list = [SVC(kernel='linear', probability=True, C=.1, max_iter=5000),
            SVC(kernel='rbf', probability=True, C=.1, max_iter=5000),
            SVC(kernel='poly', probability=True, C=.1, max_iter=5000)]
kernel_list = ['linear', 'rbf', 'poly']
X_train, X_test, y_train, y_test = train_test_split(features_svm, credit, stratify = credit)
y_prob = []
for clf, ker in zip(svm_list, kernel_list):
    print(' svm kerel = %s'%ker)
    _y_prob, metric_train, metric_test, t = train_predict(clf, X_train, y_train, X_test, y_test)
    y_prob.append(_y_prob)

plt.figure()
for _y_prob in y_prob:
    fpr, tpr, thresholds = roc_curve(y_test, _y_prob)
    while len(fpr) > 100:
        fpr = [fpr[i] for i in range(len(fpr)) if i % 2 == 0]
        tpr = [tpr[i] for i in range(len(tpr)) if i % 2 == 0]
    plt.plot(fpr, tpr,marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(kernel_list, loc=0, fontsize='small')
plt.title('SVM')
plt.show()
```

```
 svm kerel = linear
Building classifier：SVC....
```

```
d:\python27\lib\site-packages\sklearn\svm\base.py:244: ConvergenceWarning: Solver
terminated early (max_iter=5000).  Consider pre-processing your data with Standard
Scaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

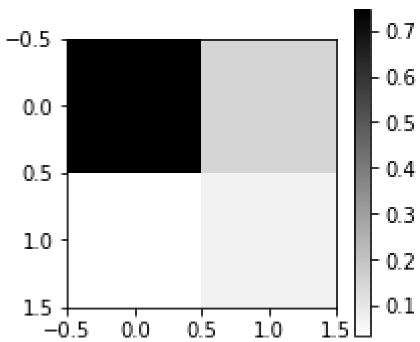
```
 train set: acc=0.7572; f1=0.4673; auc=0.7085
 test set: acc=0.7528; f1=0.4617; auc=0.6989
 test confusion martix:
[[4851  990]
 [ 864  795]]
```



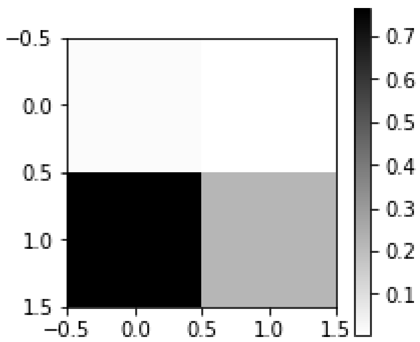

```
 svm kerel = rbf
Building classifier：SVC....
```

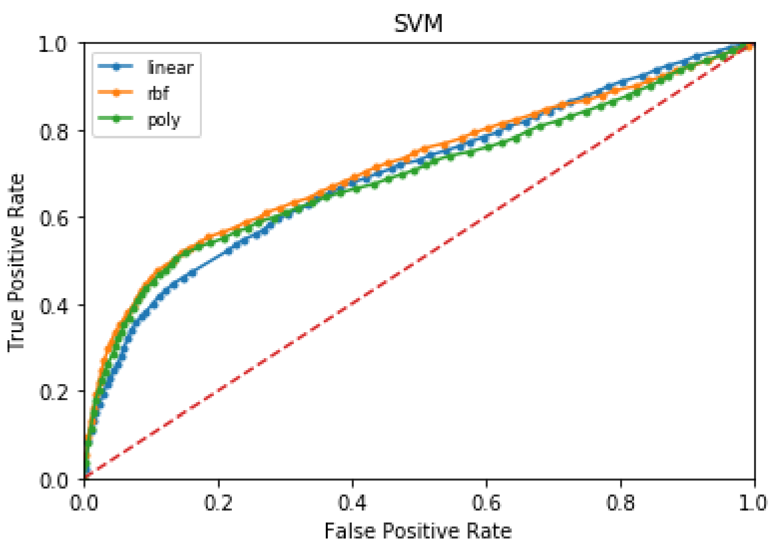
```
d:\python27\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default
value of gamma will change from 'auto' to 'scale' in version 0.22 to account bette
r for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this w
arning.
  "avoid this warning.", FutureWarning)
```

```
 train set: acc=0.8175; f1=0.4342; auc=0.7481
 test set: acc=0.8157; f1=0.4303; auc=0.7158
 test confusion martix:
[[5596  245]
 [1137  522]]
```



```
 svm kerel = poly
Building classifier: SVC....
 train set: acc=0.2358; f1=0.3648; auc=0.7162
 test set: acc=0.2337; f1=0.3641; auc=0.6955
 test confusion martix:
[[ 108 5733]
 [  14 1645]]
```

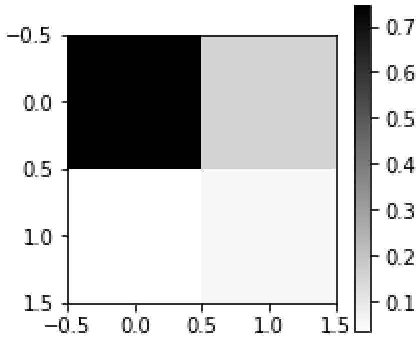




In [92]:

```
Clist = [.01, .1, .5, 1., 5., 10.]
X_train, X_test, y_train, y_test = train_test_split(features_svm, credit, stratify = credit)
acc_train, f1_train, auc_train = np.zeros(len(Clist)), np.zeros(len(Clist)), np.zeros(len(Clist
))
acc_test, f1_test, auc_test = np.zeros(len(Clist)), np.zeros(len(Clist)), np.zeros(len(Clist))
y_prob = []
for i, C in enumerate(Clist):
    print('rbf-svm  C= %f'%C)
    clf = SVC(kernel='rbf', probability=True, C=C, max_iter=10000)
    _y_prob, metric_train, metric_test, t = train_predict(clf, X_train, y_train, X_test, y_test)

    acc_train[i], f1_train[i], auc_train[i] = metric_train
    acc_test[i], f1_test[i], auc_test[i] = metric_test

    y_prob.append(_y_prob)

print('*******model complexity curve!!!*********')
plt.figure()
plt.plot(Clist, auc_train, '*-')
plt.plot(Clist, auc_test, 'o-')
plt.plot(Clist, auc_test - auc_train)
plt.xlabel('C')
plt.legend(['auc score (training)', 'auc score (testing)', 'auc score (testing - training)'])

plt.figure()
plt.plot(Clist, acc_train, '*-')
plt.plot(Clist, acc_test, 'o-')
plt.plot(Clist, acc_test - acc_train)
plt.xlabel('C')
plt.legend(['acc score (training)', 'acc score (testing)', 'acc score (testing - training)'])

plt.figure()
for _y_prob in y_prob:
    fpr, tpr, thresholds = roc_curve(y_test, _y_prob)
    while len(fpr) > 100:
        fpr = [fpr[i] for i in range(len(fpr)) if i % 2 == 0]
        tpr = [tpr[i] for i in range(len(tpr)) if i % 2 == 0]
    plt.plot(fpr, tpr,marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(['C=%f'%C for C in Clist], loc=0, fontsize='small')
plt.title('SVM with rbf kernel')
plt.show()
```

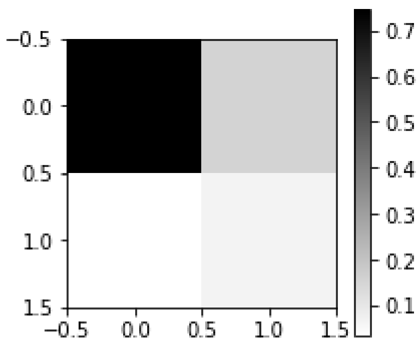
```
rbf-svm  C= 0.010000
Building classifier：SVC....
 train set: acc=0.8090; f1=0.3871; auc=0.7492
 test set: acc=0.8064; f1=0.3879; auc=0.7210
 test confusion martix:
[[5588  253]
 [1199  460]]
```



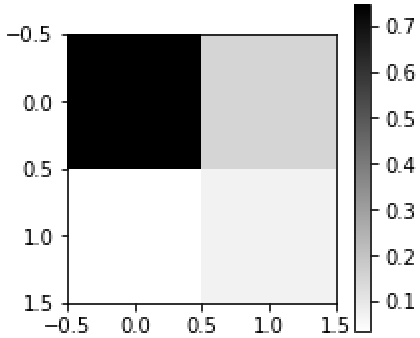

```
rbf-svm  C= 0.100000
Building classifier：SVC....
```

```
d:\python27\lib\site-packages\sklearn\svm\base.py:244: ConvergenceWarning: Solver
terminated early (max_iter=10000).  Consider pre-processing your data with Standar
dScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

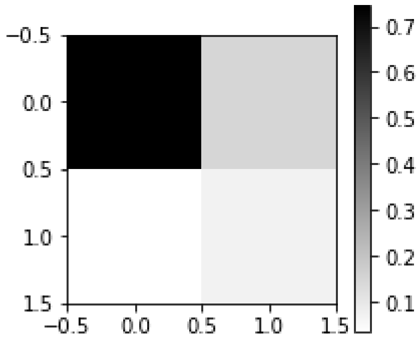
```
 train set: acc=0.8176; f1=0.4280; auc=0.7671
 test set: acc=0.8144; f1=0.4243; auc=0.7144
 test confusion martix:
[[5595  246]
 [1146  513]]
```



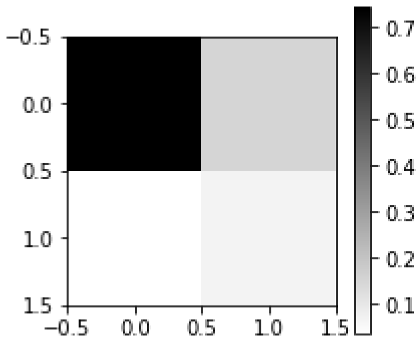

```
rbf-svm  C= 0.500000
Building classifier：SVC....
 train set: acc=0.8232; f1=0.4481; auc=0.7929
 test set: acc=0.8165; f1=0.4337; auc=0.7106
 test confusion martix:
[[5597  244]
 [1132  527]]
```



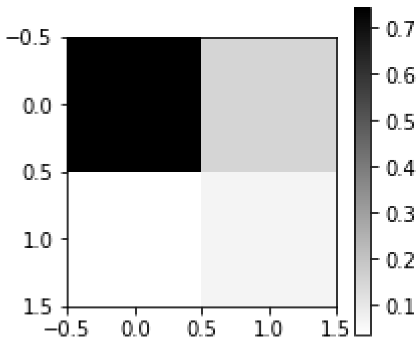

```
rbf-svm  C= 1.000000
Building classifier: SVC....
 train set: acc=0.8268; f1=0.4630; auc=0.7979
 test set: acc=0.8169; f1=0.4394; auc=0.7060
 test confusion martix:
[[5589  252]
 [1121  538]]
```



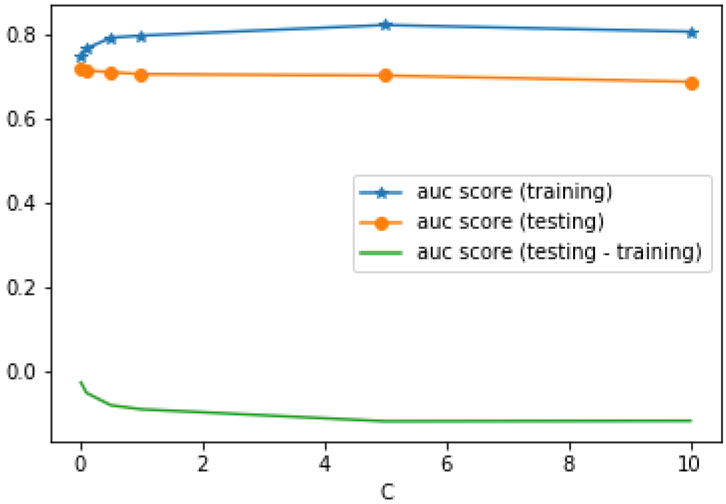

```
rbf-svm  C= 5.000000
Building classifier: SVC....
 train set: acc=0.8408; f1=0.5158; auc=0.8231
 test set: acc=0.8133; f1=0.4248; auc=0.7027
 test confusion martix:
[[5583  258]
 [1142  517]]
```

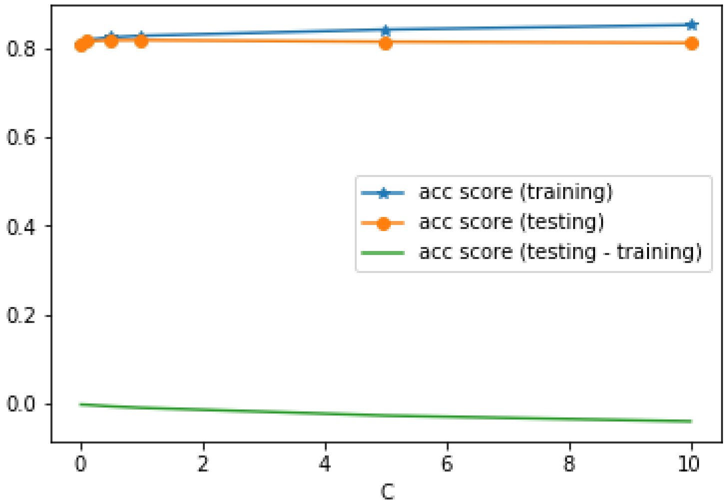


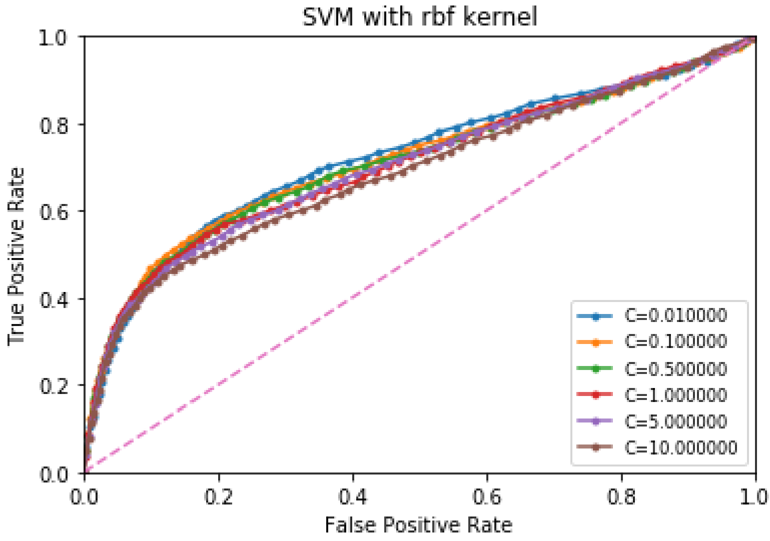

```
rbf-svm  C= 10.000000
Building classifier: SVC....
 train set: acc=0.8511; f1=0.5512; auc=0.8073
 test set: acc=0.8107; f1=0.4147; auc=0.6879
 test confusion martix:
[[5577  264]
 [1156  503]]
```




```
*******model complexity curve!!!*********
```







In [ ]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
clf = SVC(kernel='rbf', C=0.01, probability=True) # optimal model

train_acc, train_f1, train_auc = np.zeros([len(train_size),10]), np.zeros([len(train_size),10]),
np.zeros([len(train_size),10])
test_acc, test_f1, test_auc = np.zeros([len(train_size),10]), np.zeros([len(train_size),10]), np
.zeros([len(train_size),10])

for j in range(10):
    for i, size in enumerate(train_size):
        X_train, X_test, y_train, y_test = train_test_split(features, credit, test_size = 1-size
, stratify = credit)
        _, train_metric, test_metric, t = train_predict(clf, X_train, y_train, X_test, y_test, s
ilent=True)
        train_acc[i,j], train_f1[i,j], train_auc[i,j] = train_metric
        test_acc[i,j], test_f1[i,j], test_auc[i,j] = test_metric

train_acc, train_f1, train_auc = np.mean(train_acc, axis=1), np.mean(train_f1, axis=1), np.mean(
train_auc, axis=1)
test_acc, test_f1, test_auc = np.mean(test_acc, axis=1), np.mean(test_f1, axis=1), np.mean(test_
auc, axis=1)
plt.figure()
plt.plot(train_size, train_acc, '*--')
plt.plot(train_size, test_acc, '*-')
plt.plot(train_size, train_auc, 'o--')
plt.plot(train_size, test_auc, 'o-')
plt.xlabel('training data used')
plt.legend(['training acc', 'testing acc', 'training auc', 'testing auc'])
```

feature importance

some of my designed features are important, for exmaple, ratio2, ratio4 and ratio 1.

In [40]:

```
def get_feature_importance(clsf, feat):
    imp = model[2].feature_importances_.tolist()
    result = pd.DataFrame({'feat':feat,'score':imp})
    result = result.sort_values(by=['score'],ascending=False)
    return result

clf = DecisionTreeClassifier(max_depth=5)
clf.fit(features, credit)
feature_name = data.columns.values.tolist()
del feature_name[feature_name.index('default.payment.next.month')]
get_feature_importance(model[2], feature_name)
```

Out[40]:

	feat	score
6	PAY_0	0.685539
7	PAY_2	0.134716
19	PAY_AMT2	0.051279
8	PAY_3	0.023178
9	PAY_4	0.019968
27	ratio2	0.018275
12	BILL_AMT1	0.011965
10	PAY_5	0.009725
20	PAY_AMT3	0.006864
13	BILL_AMT2	0.006706
15	BILL_AMT4	0.005523
23	PAY_AMT6	0.004783
21	PAY_AMT4	0.004031
0	ID	0.003413
29	ratio4	0.003317
30	ratio5	0.002782
26	ratio1	0.002343
5	AGE	0.002220
22	PAY_AMT5	0.002007
1	LIMIT_BAL	0.001368
25	SE_MA	0.000000
24	AgeBin	0.000000
28	ratio3	0.000000
16	BILL_AMT5	0.000000
18	PAY_AMT1	0.000000
17	BILL_AMT6	0.000000
14	BILL_AMT3	0.000000
11	PAY_6	0.000000
4	MARRIAGE	0.000000
3	EDUCATION	0.000000
2	SEX	0.000000
31	ratio6	0.000000