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('voice.csv')
data.sample(5)
```

Out[2]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	
1046	0.206080	0.058981	0.222475	0.149010	0.258515	0.109505	1.528576	5.190499	0.
2851	0.227306	0.031154	0.229053	0.221053	0.240000	0.018947	4.994139	35.937231	0.
290	0.075701	0.089599	0.029361	0.002402	0.139066	0.136663	11.669905	166.260441	0.
695	0.157915	0.062103	0.176599	0.096479	0.196734	0.100255	3.564927	17.988527	0.
1703	0.192677	0.070494	0.215170	0.176886	0.241437	0.064551	2.166917	11.948492	0.

5 rows × 21 columns

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
meanfreq    3168 non-null float64
sd          3168 non-null float64
median      3168 non-null float64
Q25         3168 non-null float64
Q75         3168 non-null float64
IQR         3168 non-null float64
skew        3168 non-null float64
kurt        3168 non-null float64
sp.ent      3168 non-null float64
sfm         3168 non-null float64
mode        3168 non-null float64
centroid    3168 non-null float64
meanfun     3168 non-null float64
minfun      3168 non-null float64
maxfun      3168 non-null float64
meandom     3168 non-null float64
mindom      3168 non-null float64
maxdom      3168 non-null float64
dfrange     3168 non-null float64
modindx     3168 non-null float64
label       3168 non-null object
dtypes: float64(20), object(1)
memory usage: 507.4+ KB
```

In [4]:

```
data['label_binary'] = data['label']=='male'
data = data.drop(columns=['label'])
```

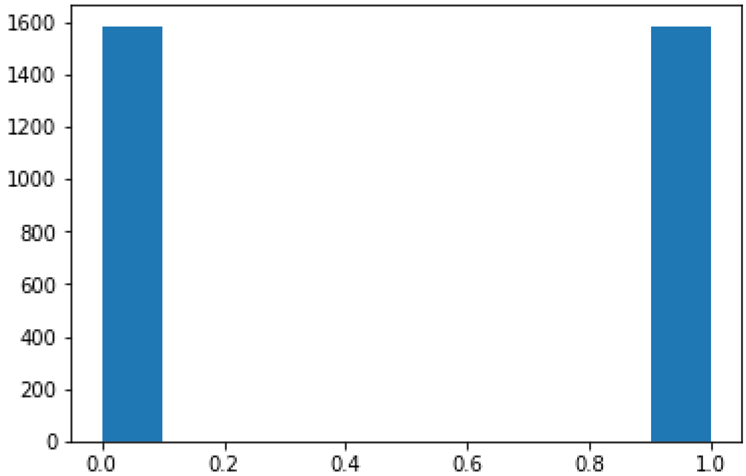
Here shows the distribution of target variable

In [5]:

```
plt.hist(data['label_binary'])
```

Out[5]:

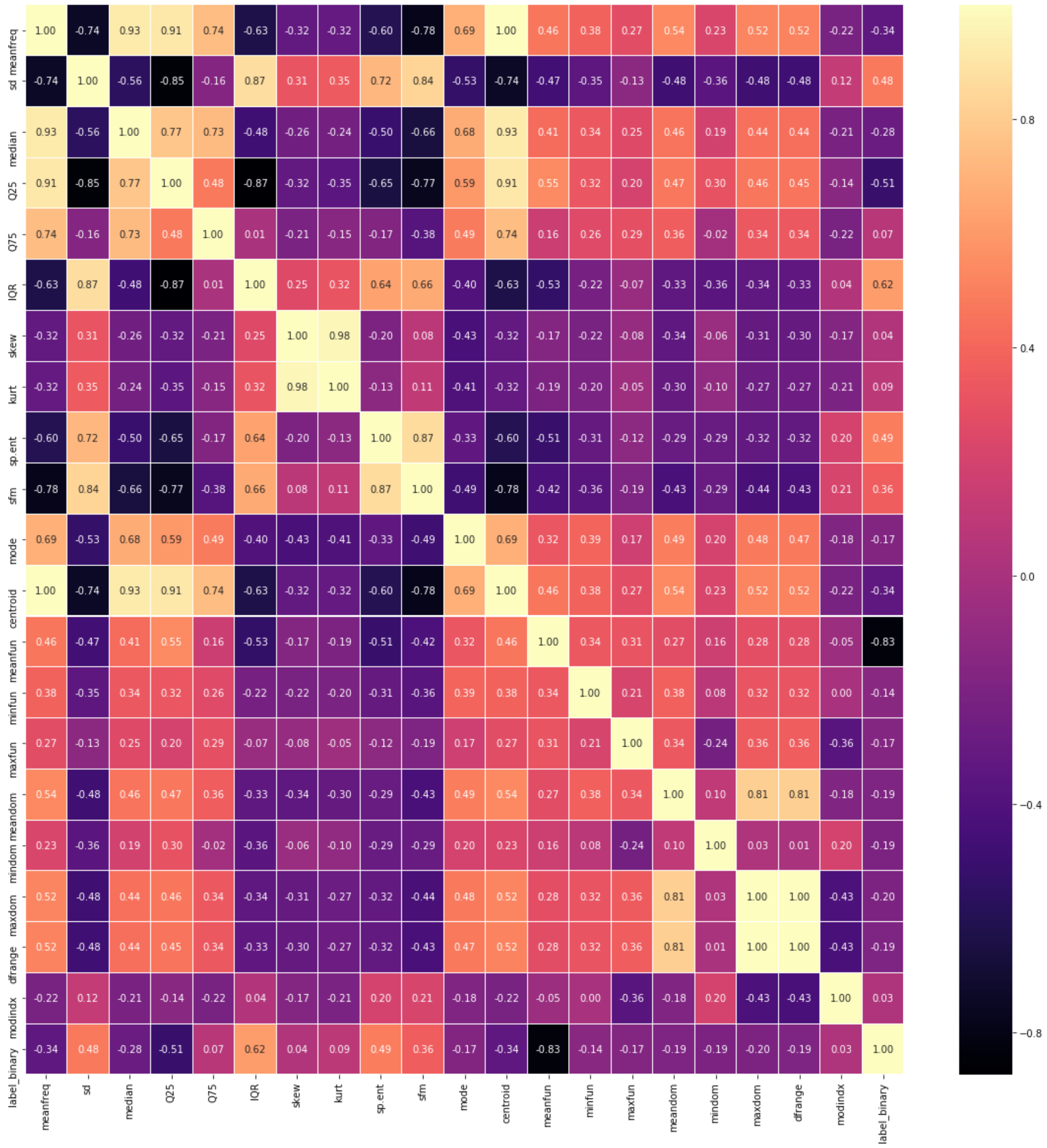
```
(array([1584.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
        1584.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



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.

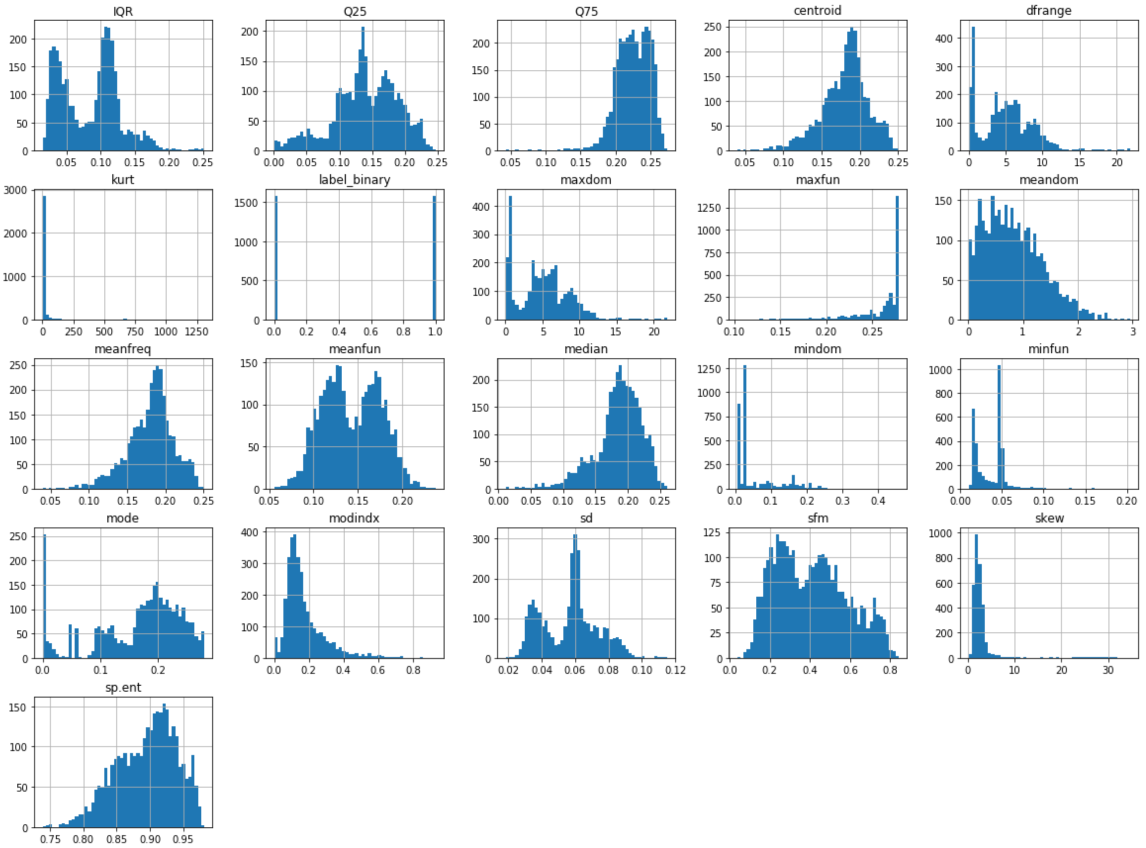
In [6]:

```
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 [7]:

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



In [8]:

```
gender = data['label_binary'].values
features = data.drop(['label_binary'], axis=1).values
print ('Data is ready.')
print ('x shape', features.shape)
print ('y shape', gender.shape)
```

```
Data is ready.
('x shape', (3168, 20))
('y shape', (3168,))
```

2. model building

In [9]:

```
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 [10]:

```
# 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

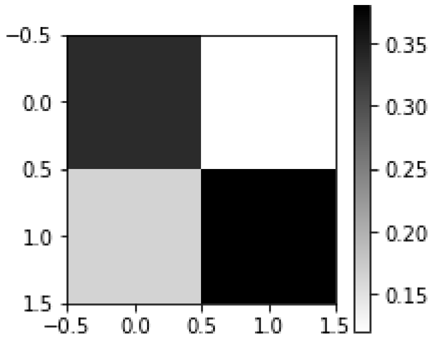
Note that when finetuning parameters, I use 5-fold average testing auc (area under roc curve) score as main metric.

Conclusion: according to the results below, we set optimal k to be 10.

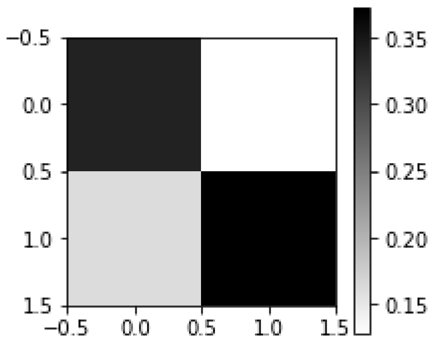
In [11]:

```
# 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, gender, stratify = gender)
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()
```

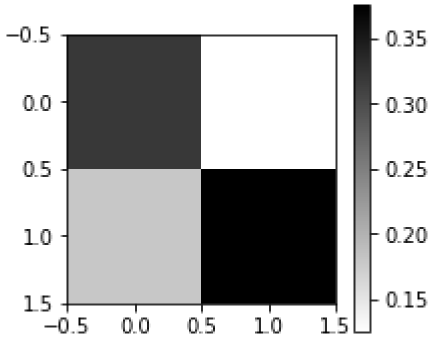
```
kNN k = 1
Building classifier: KNeighborsClassifier....
 train set: acc=1.0000; f1=1.0000; auc=1.0000
 test set: acc=0.7159; f1=0.7279; auc=0.7159
 test confusion martix:
[[266 130]
 [ 95 301]]
```



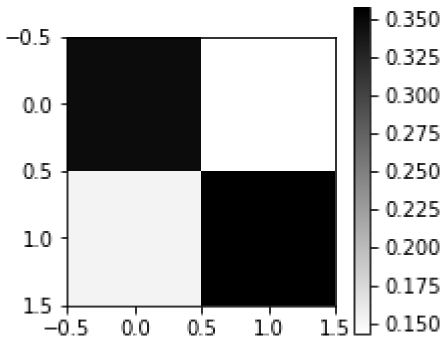

```
kNN k = 3
Building classifier: KNeighborsClassifier....
 train set: acc=0.8476; f1=0.8488; auc=0.9270
 test set: acc=0.7109; f1=0.7204; auc=0.7701
 test confusion martix:
[[268 128]
 [101 295]]
```



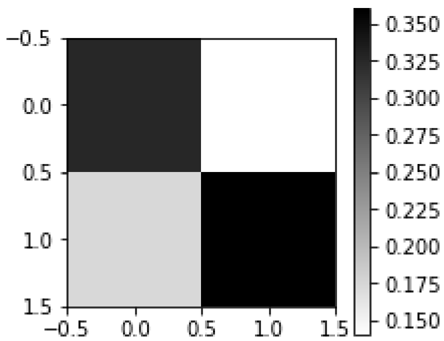

```
kNN k = 5
Building classifier: KNeighborsClassifier....
 train set: acc=0.8098; f1=0.8124; auc=0.8952
 test set: acc=0.6944; f1=0.7105; auc=0.7673
 test confusion martix:
[[253 143]
 [ 99 297]]
```



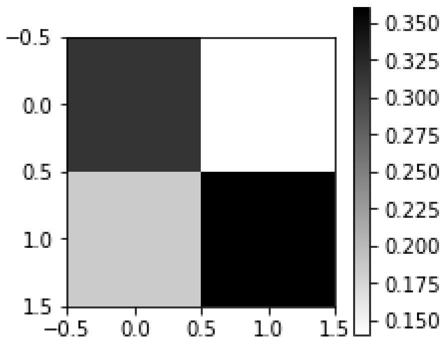

```
kNN k = 10
Building classifier: KNeighborsClassifier....
 train set: acc=0.7652; f1=0.7632; auc=0.8494
 test set: acc=0.7045; f1=0.7075; auc=0.7707
 test confusion martix:
[[275 121]
 [113 283]]
```



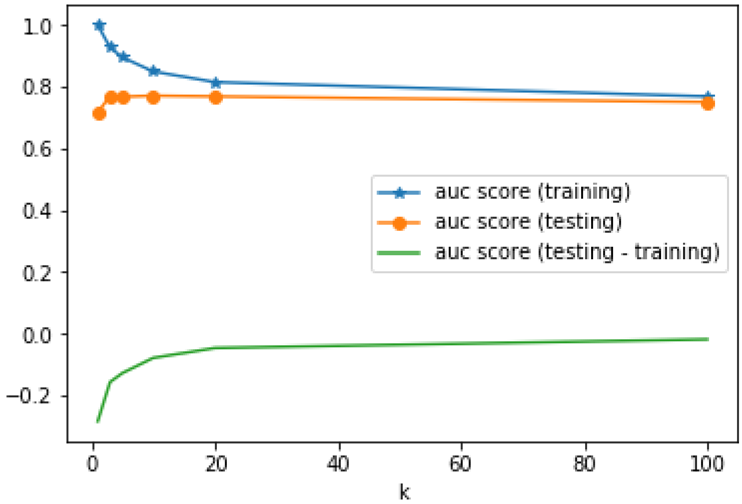

```
kNN k = 20
Building classifier: KNeighborsClassifier....
 train set: acc=0.7302; f1=0.7310; auc=0.8147
 test set: acc=0.6856; f1=0.6960; auc=0.7683
 test confusion martix:
[[258 138]
 [111 285]]
```

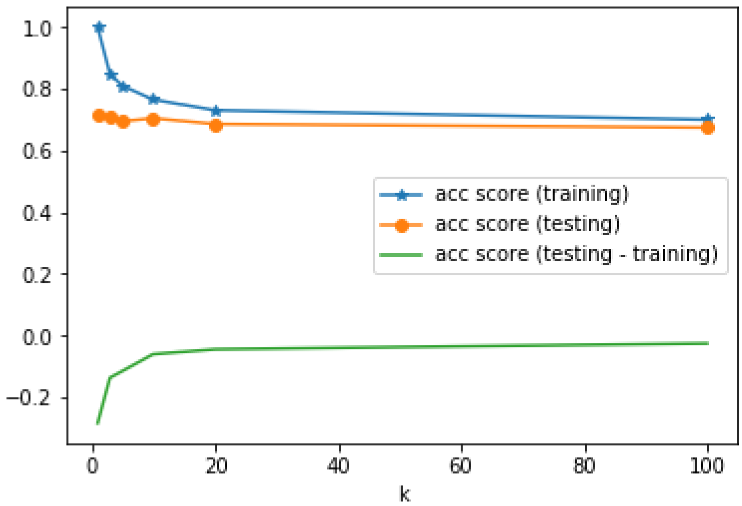


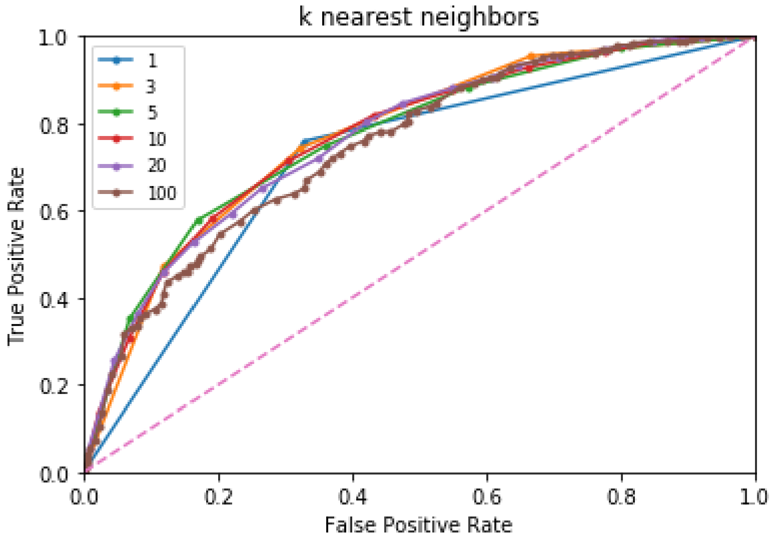

```
kNN k = 100
Building classifier: KNeighborsClassifier....
 train set: acc=0.6999; f1=0.7067; auc=0.7684
 test set: acc=0.6742; f1=0.6884; auc=0.7501
 test confusion martix:
[[249 147]
 [111 285]]
```



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






k nearest neighbors
1
3
5
10
20
100
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

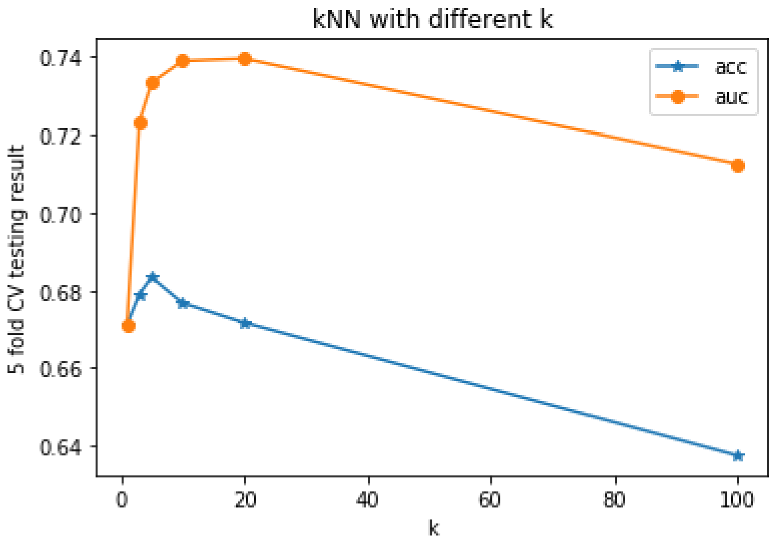
In [12]:

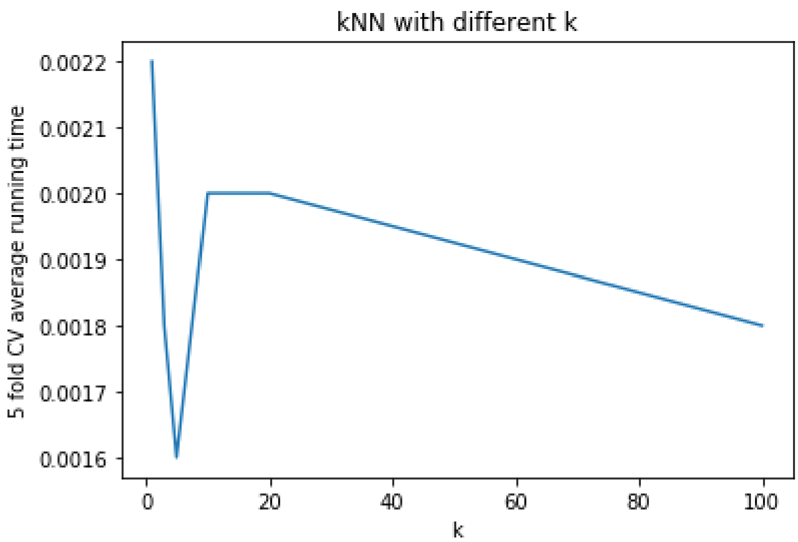
```
fold, n_fold = 0, 5
acc, f1, auc = np.zeros((len(klist), n_fold)), np.zeros((len(klist), n_fold)), np.zeros((len(kli
st), n_fold))
running_time = np.zeros((len(klist), n_fold))
for train_idx, test_idx in StratifiedKFold(n_splits=n_fold).split(features, gender):
    print('*** fold %d/%d'%(fold+1, n_fold))
    X_train, X_test = features[train_idx, :], features[test_idx, :]
    y_train, y_test = gender[train_idx], gender[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')
```

```
*** fold 1/5
*** fold 2/5
*** fold 3/5
*** fold 4/5
*** fold 5/5
5 fold CV result of kNN
 k=1, acc=0.6711; f1=0.6843; auc=0.6711
 k=3, acc=0.6790; f1=0.6925; auc=0.7231
 k=5, acc=0.6834; f1=0.6987; auc=0.7333
 k=10, acc=0.6768; f1=0.6732; auc=0.7389
 k=20, acc=0.6717; f1=0.6728; auc=0.7395
 k=100, acc=0.6376; f1=0.6359; auc=0.7125
*******kNN learning curve!!!*********
```

Out[12]:

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





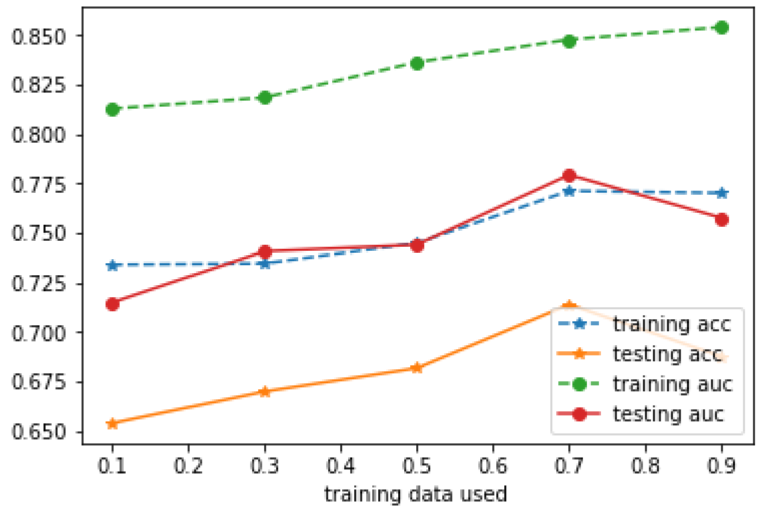
more training data used, the performance goes up, however, too many (90%) training data may cause overfitting, i.e., increse on training perfoamnce but decease on testing performance

In [13]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
nfold = 5
clf = KNeighborsClassifier(n_neighbors=10) # 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, gender, test_size = 1-size, st
ratify = gender)
    _, 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'])
```

Out[13]:

<matplotlib.legend.Legend at 0x15ab30d0>



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

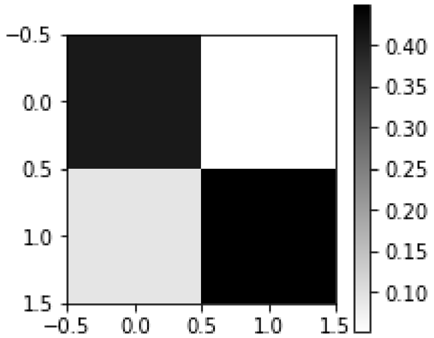
Conclusion: according to the results below, we set optimal n to be 100.

In [14]:

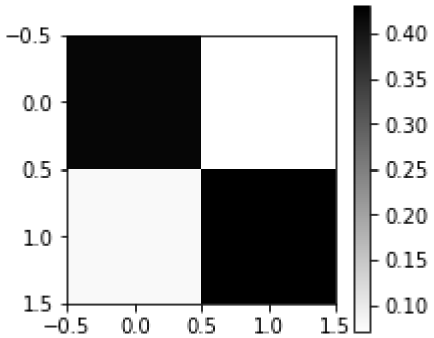
```
nlist = [10, 20, 50, 100]
X_train, X_test, y_train, y_test = train_test_split(features, gender, stratify = gender)
acc_train, f1_train, auc_train = np.zeros(len(nlist)), np.zeros(len(nlist)), np.zeros(len(nlist
))
acc_test, f1_test, auc_test = np.zeros(len(nlist)), np.zeros(len(nlist)), np.zeros(len(nlist))
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=0.8569; f1=0.8635; auc=0.8975
 test set: acc=0.8548; f1=0.8606; auc=0.9003
 test confusion martix:
[[322  74]
 [ 41 355]]
```

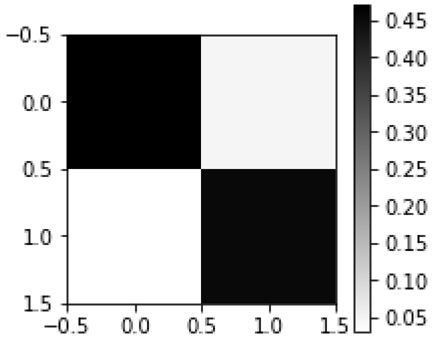
```
d:\python27\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:562:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```



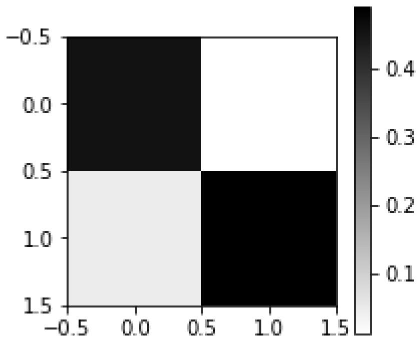

```
 neural networks (single hidden layer) n = 20
Building classifier: MLPClassifier....
 train set: acc=0.8872; f1=0.8892; auc=0.9581
 test set: acc=0.8523; f1=0.8536; auc=0.9532
 test confusion martix:
[[334  62]
 [ 55 341]]
```



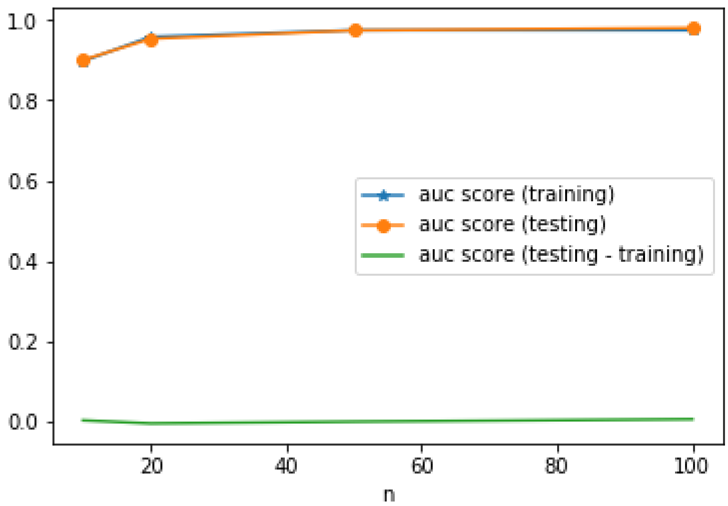

```
 neural networks (single hidden layer) n = 50
Building classifier: MLPClassifier....
 train set: acc=0.9230; f1=0.9225; auc=0.9746
 test set: acc=0.9242; f1=0.9229; auc=0.9742
 test confusion martix:
[[373  23]
 [ 37 359]]
```

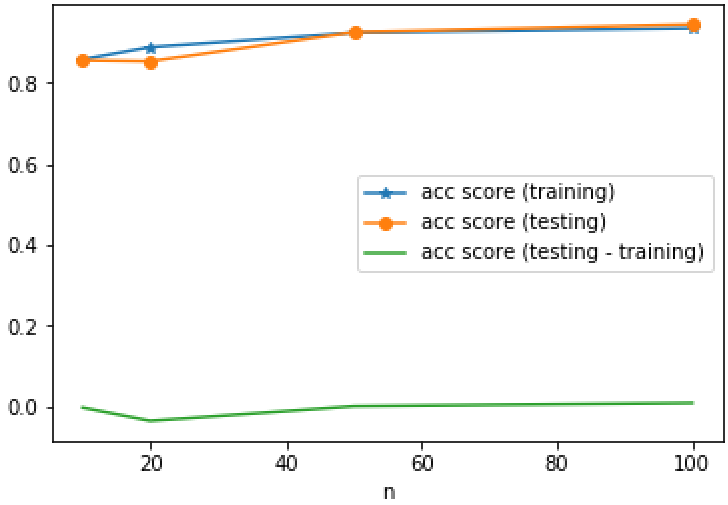


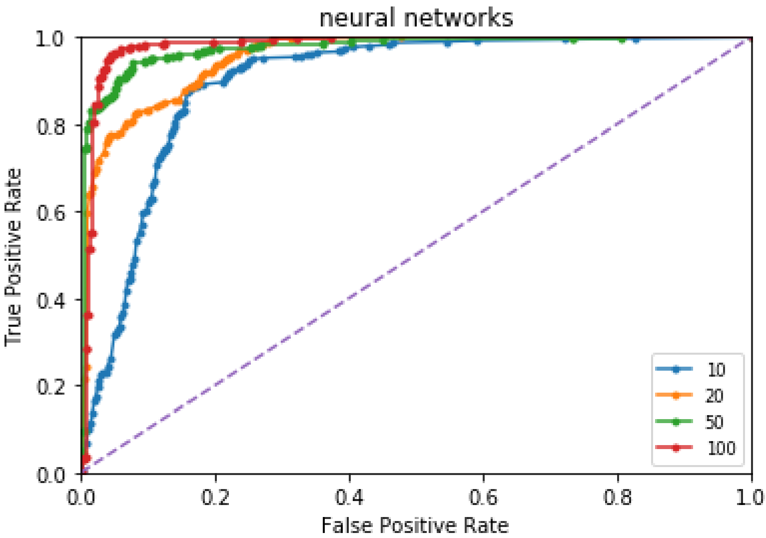

```
 neural networks (single hidden layer) n = 100
Building classifier: MLPClassifier....
 train set: acc=0.9339; f1=0.9366; auc=0.9745
 test set: acc=0.9432; f1=0.9452; auc=0.9801
 test confusion martix:
[[359  37]
 [  8 388]]
```




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




0.8
0.6
0.4
0.2
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

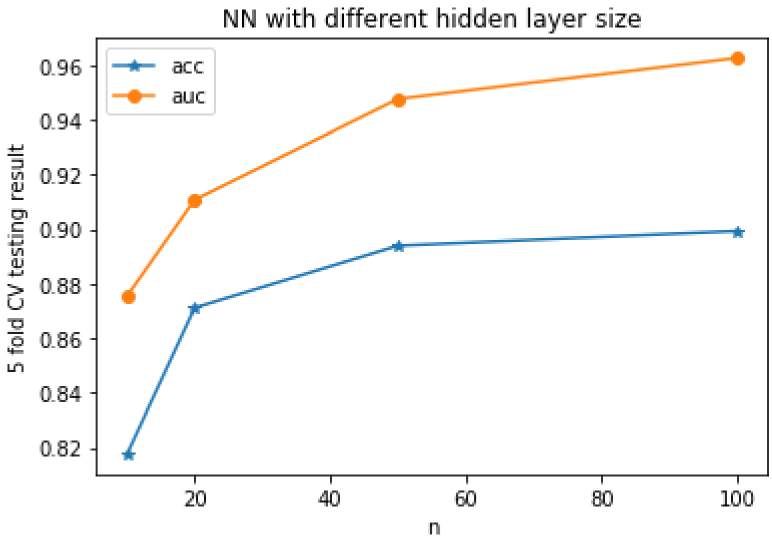
In [16]:

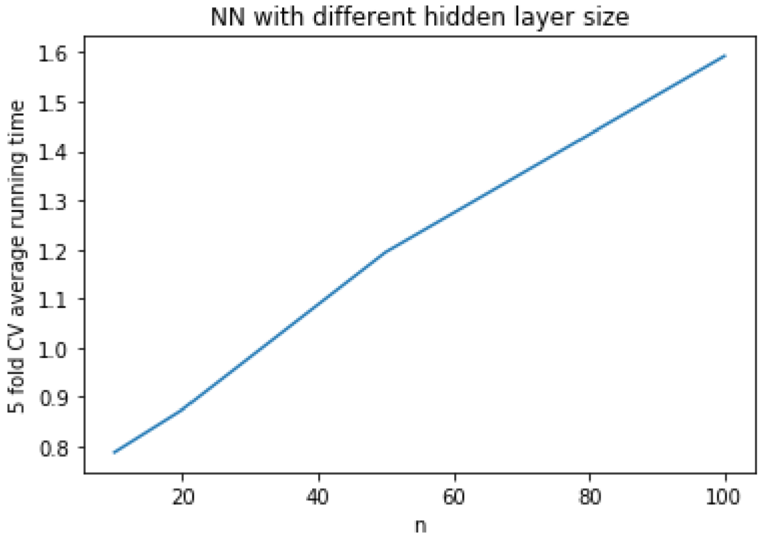
```
fold, n_fold = 0, 5
acc, f1, auc = np.zeros((len(nlist), n_fold)), np.zeros((len(nlist), n_fold)), np.zeros((len(nli
st), n_fold))
running_time = np.zeros((len(nlist), n_fold))
for train_idx, test_idx in StratifiedKFold(n_splits=n_fold).split(features, gender):
    print('*** fold %d/%d'%(fold+1, n_fold))
    X_train, X_test = features[train_idx, :], features[test_idx, :]
    y_train, y_test = gender[train_idx], gender[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=0.8176; f1=0.8232; auc=0.8754
 n=20, acc=0.8712; f1=0.8820; auc=0.9108
 n=50, acc=0.8939; f1=0.9001; auc=0.9476
 n=100, acc=0.8993; f1=0.8965; auc=0.9626
*******neural network learning curve!!!*********
```

Out[16]:

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



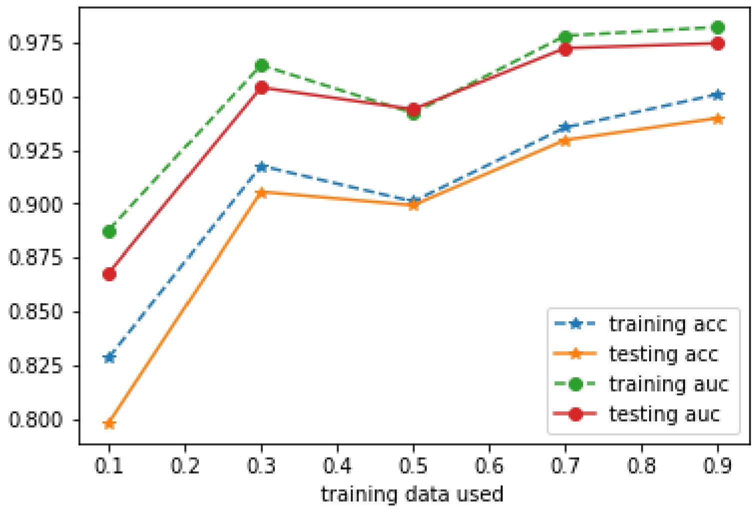

NN with different hidden layer size
5 fold CV average running time
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
20
40
60
80
100
n

In [17]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
nfold = 5
clf = MLPClassifier(hidden_layer_sizes=100, 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, gender, test_size = 1-size
, stratify = gender)
        _, 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[17]:

<matplotlib.legend.Legend at 0x1607b750>



finetune parameters for decision tree

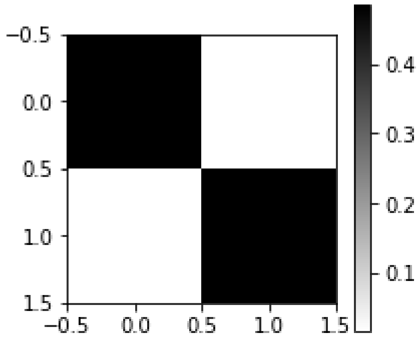
max_depth to be 3, 5, 10, 20, 30

Conclusion: according to the results below, we set the optimal depth to be 3

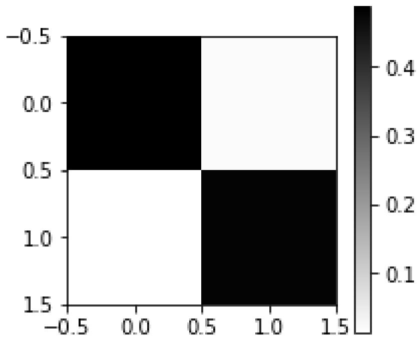
In [18]:

```
depth = [3, 5, 10, 20, 30]
X_train, X_test, y_train, y_test = train_test_split(features, gender, stratify = gender)
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()
```

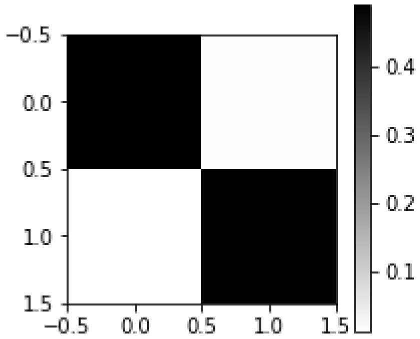
```
 tree with depth = 3
Building classifier: DecisionTreeClassifier....
 train set: acc=0.9705; f1=0.9705; auc=0.9856
 test set: acc=0.9722; f1=0.9722; auc=0.9823
 test confusion martix:
[[385  11]
 [ 11 385]]
```



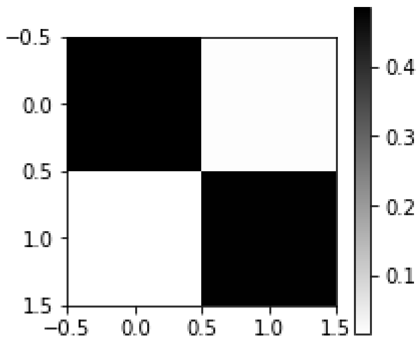

```
 tree with depth = 5
Building classifier: DecisionTreeClassifier....
 train set: acc=0.9827; f1=0.9827; auc=0.9975
 test set: acc=0.9672; f1=0.9669; auc=0.9773
 test confusion martix:
[[386  10]
 [ 16 380]]
```



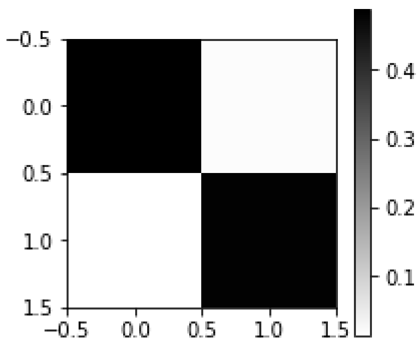

```
 tree with depth = 10
Building classifier: DecisionTreeClassifier....
 train set: acc=1.0000; f1=1.0000; auc=1.0000
 test set: acc=0.9747; f1=0.9746; auc=0.9747
 test confusion martix:
[[388   8]
 [ 12 384]]
```



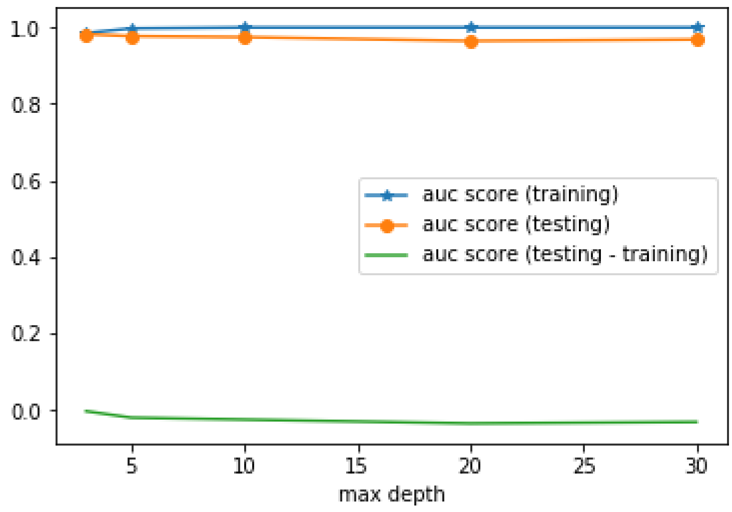

```
 tree with depth = 20
Building classifier: DecisionTreeClassifier....
 train set: acc=1.0000; f1=1.0000; auc=1.0000
 test set: acc=0.9646; f1=0.9645; auc=0.9646
 test confusion martix:
[[384  12]
 [ 16 380]]
```

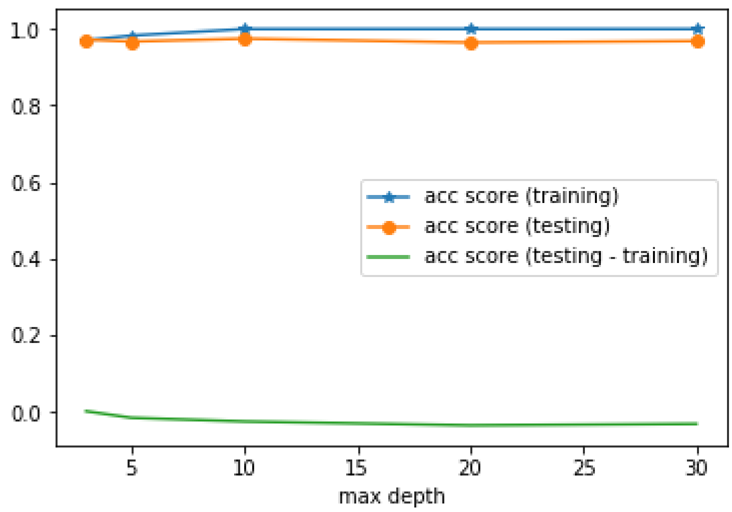


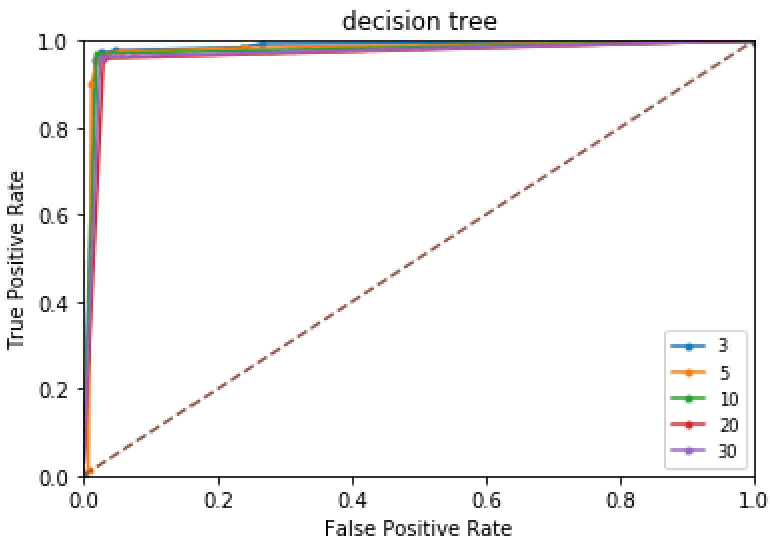

```
 tree with depth = 30
Building classifier: DecisionTreeClassifier....
 train set: acc=1.0000; f1=1.0000; auc=1.0000
 test set: acc=0.9684; f1=0.9682; auc=0.9684
 test confusion martix:
[[386  10]
 [ 15 381]]
```




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


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


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


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

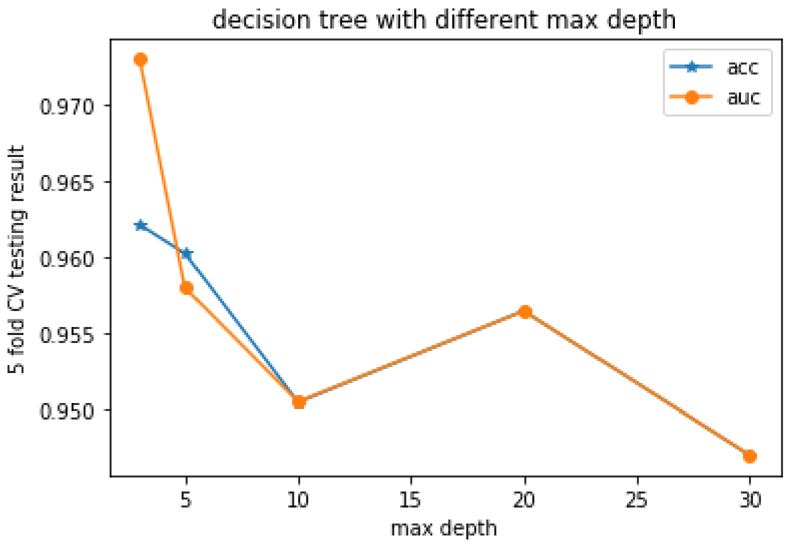
In [19]:

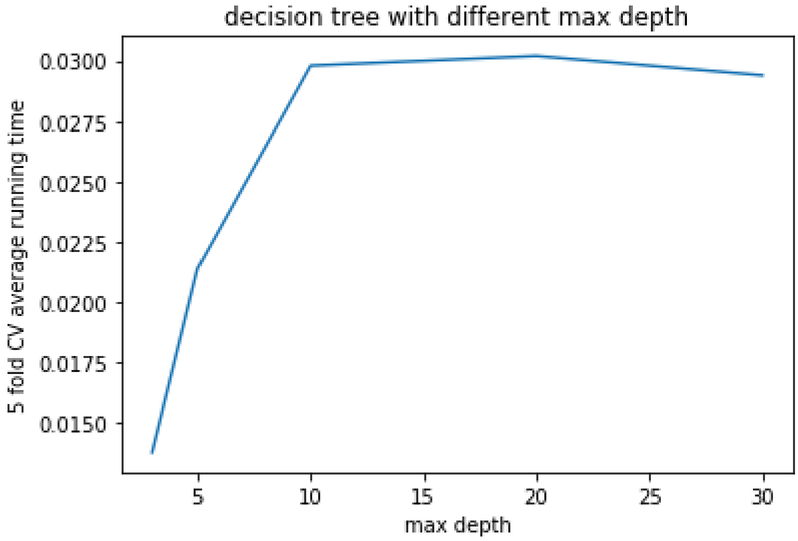
```
fold, n_fold = 0, 5
acc, f1, auc = np.zeros((len(depth), n_fold)), np.zeros((len(depth), n_fold)), np.zeros((len(dep
th), n_fold))
running_time = np.zeros((len(depth), n_fold))
for train_idx, test_idx in StratifiedKFold(n_splits=n_fold).split(features, gender):
    print('*** fold %d/%d'%(fold+1, n_fold))
    X_train, X_test = features[train_idx, :], features[test_idx, :]
    y_train, y_test = gender[train_idx], gender[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=0.9621; f1=0.9623; auc=0.9730
 max_depth=5, acc=0.9602; f1=0.9598; auc=0.9580
 max_depth=10, acc=0.9504; f1=0.9506; auc=0.9505
 max_depth=20, acc=0.9564; f1=0.9558; auc=0.9564
 max_depth=30, acc=0.9470; f1=0.9469; auc=0.9470
*******decision tree learning curve!!!*********
```

Out[19]:

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





for the tree with max-depth=3, there is no need to post prune it. So here I try to prune a tree with max_depth=50.

As we prune more nodes, the performance will first go up (we obtain a good model for generalization), then go down (because the tree is too simple now)

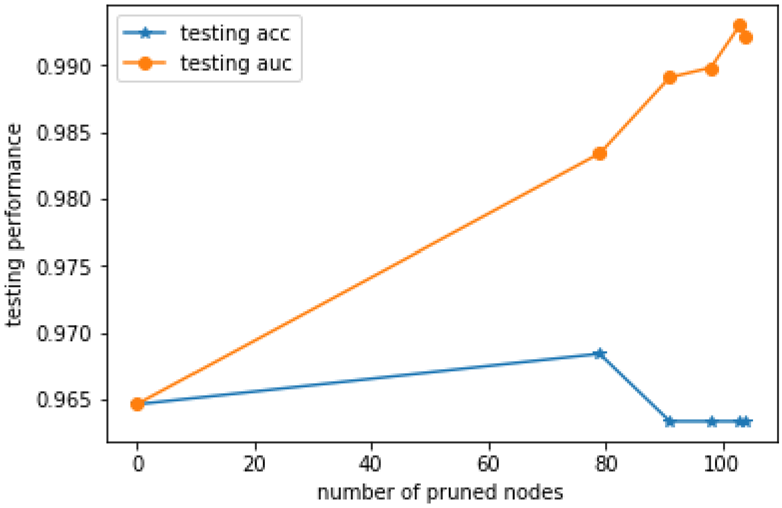
In [20]:

```
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, gender, stratify = gender)
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=0.9646; f1=0.9650; auc=0.9646
prune 79 nodes
prune 91 nodes
prune 98 nodes
prune 103 nodes
prune 104 nodes
Tree already more pruned
```

Out[20]:

```
<matplotlib.legend.Legend at 0x160c87f0>
```

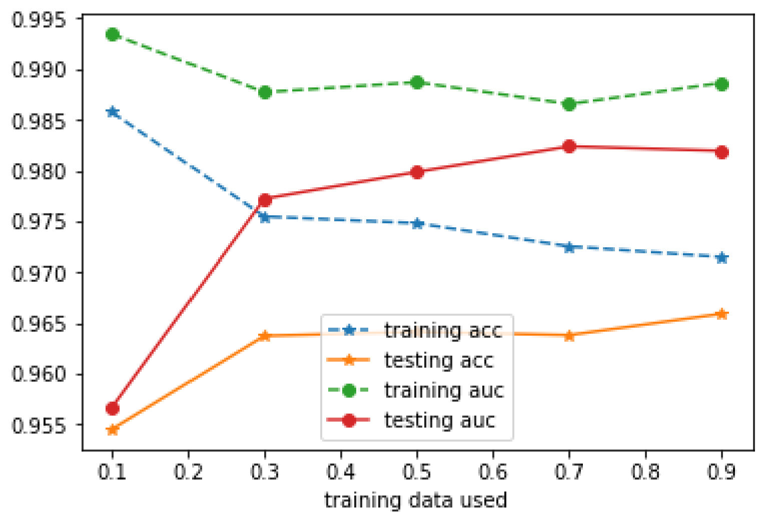


In [21]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
nfold = 5
clf = DecisionTreeClassifier(max_depth=3) # 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, gender, test_size = 1-size
, stratify = gender)
        _, 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[21]:

```
<matplotlib.legend.Legend at 0x15af7290>
```



finetune parameters for boosting

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

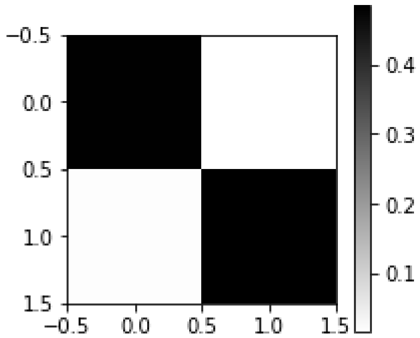
Conclusion: according to the results below, we set the optimal n to be 100

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

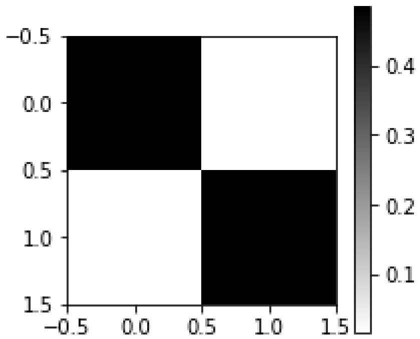
In [22]:

```
nlist = [10, 20, 50, 100, 200]
X_train, X_test, y_train, y_test = train_test_split(features, gender, stratify = gender)
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()
```

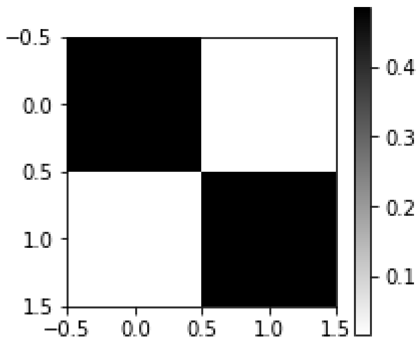
```
 number of base models = 10
Building classifier: GradientBoostingClassifier....
 train set: acc=0.9798; f1=0.9798; auc=0.9959
 test set: acc=0.9684; f1=0.9686; auc=0.9900
 test confusion martix:
[[382  14]
 [ 11 385]]
```



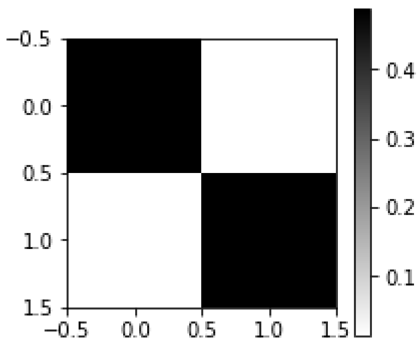

```
 number of base models = 20
Building classifier: GradientBoostingClassifier....
 train set: acc=0.9832; f1=0.9831; auc=0.9983
 test set: acc=0.9710; f1=0.9709; auc=0.9912
 test confusion martix:
[[385  11]
 [ 12 384]]
```



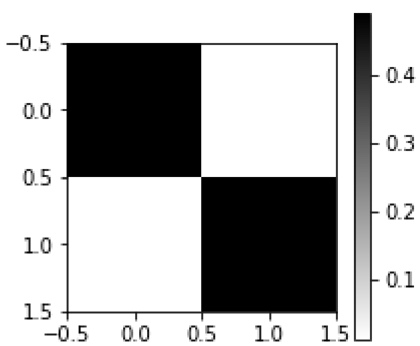

```
 number of base models = 50
Building classifier: GradientBoostingClassifier....
 train set: acc=0.9907; f1=0.9907; auc=0.9996
 test set: acc=0.9722; f1=0.9722; auc=0.9918
 test confusion martix:
[[385  11]
 [ 11 385]]
```



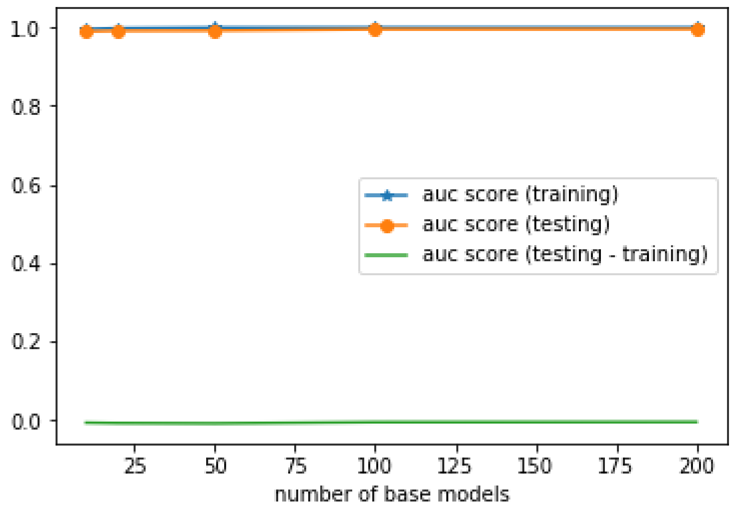

```
 number of base models = 100
Building classifier：GradientBoostingClassifier....
 train set: acc=0.9987; f1=0.9987; auc=1.0000
 test set: acc=0.9735; f1=0.9735; auc=0.9954
 test confusion martix:
[[386  10]
 [ 11 385]]
```

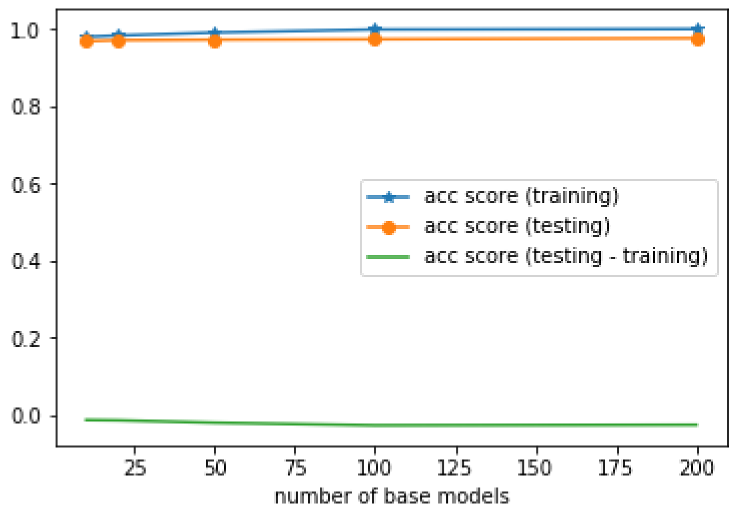


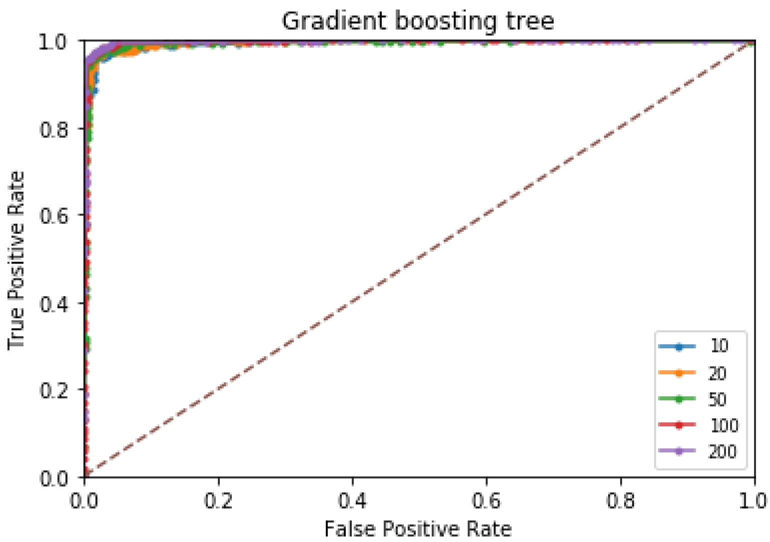

```
 number of base models = 200
Building classifier：GradientBoostingClassifier....
 train set: acc=1.0000; f1=1.0000; auc=1.0000
 test set: acc=0.9760; f1=0.9760; auc=0.9960
 test confusion martix:
[[387   9]
 [ 10 386]]
```




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


auc score (training)
auc score (testing)
auc score (testing - training)
number of base models
1.0
0.8
0.6
0.4
0.2
0.0
25
50
75
100
125
150
175
200


acc score (training)
acc score (testing)
acc score (testing - training)
number of base models
1.0
0.8
0.6
0.4
0.2
0.0
25
50
75
100
125
150
175
200


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

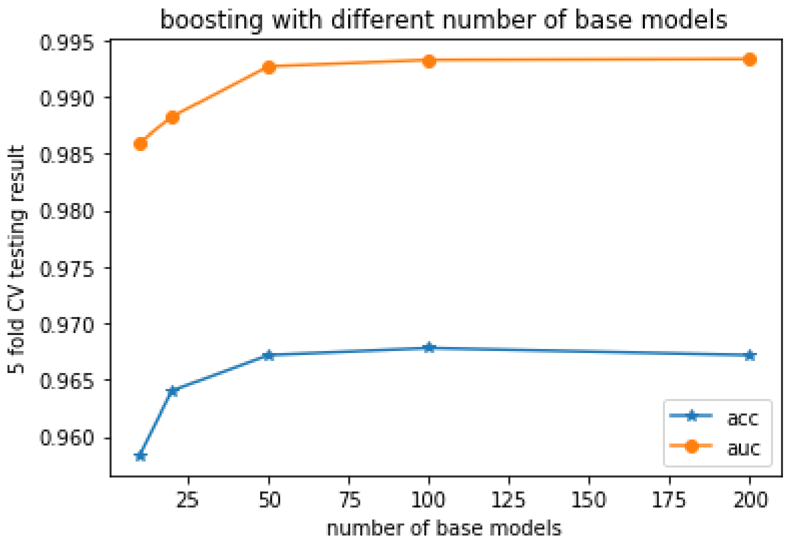
In [23]:

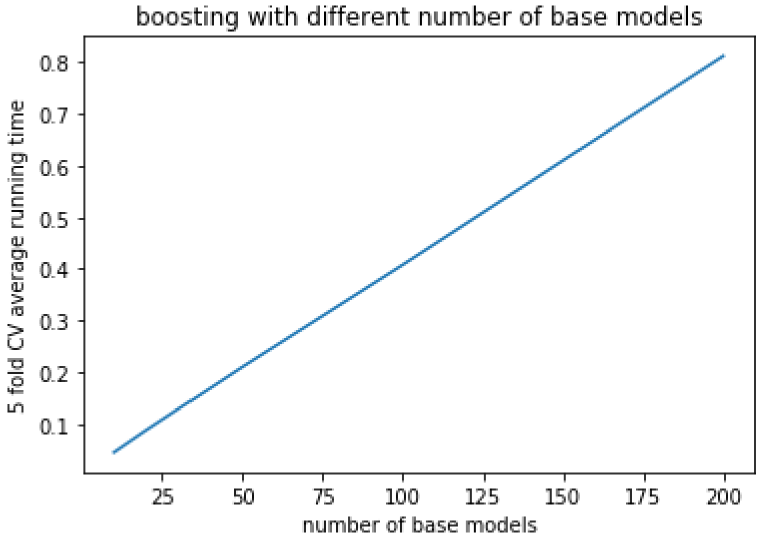
```
fold, n_fold = 0, 5
acc, f1, auc = np.zeros((len(nlist), n_fold)), np.zeros((len(nlist), n_fold)), np.zeros((len(nli
st), n_fold))
running_time = np.zeros((len(nlist), n_fold))
for train_idx, test_idx in StratifiedKFold(n_splits=n_fold).split(features, gender):
    print('*** fold %d/%d'%(fold+1, n_fold))
    X_train, X_test = features[train_idx, :], features[test_idx, :]
    y_train, y_test = gender[train_idx], gender[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=0.9583; f1=0.9584; auc=0.9860
 number of base models=20, acc=0.9640; f1=0.9638; auc=0.9882
 number of base models=50, acc=0.9672; f1=0.9671; auc=0.9927
 number of base models=100, acc=0.9678; f1=0.9678; auc=0.9933
 number of base models=200, acc=0.9672; f1=0.9672; auc=0.9934
*******boosting learning curve!!!*********
```

Out[23]:

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



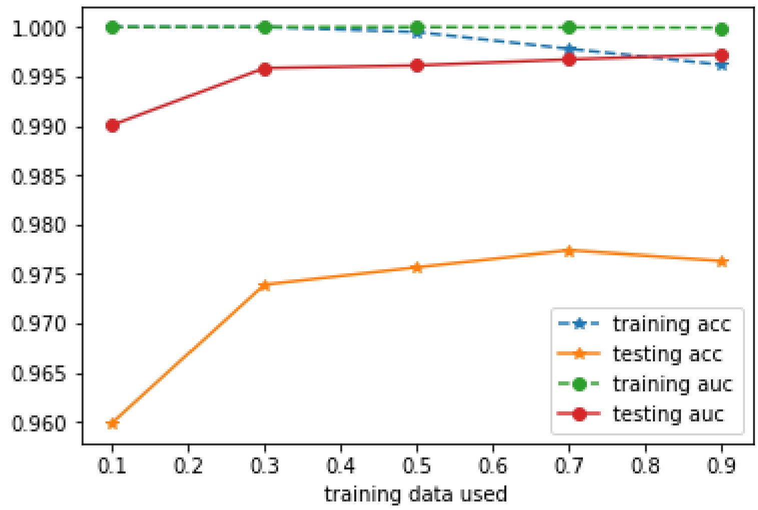

boosting with different number of base models
5 fold CV average running time
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
25
50
75
100
125
150
175
200
number of base models

In [24]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
nfold = 5
clf = GradientBoostingClassifier(n_estimators=100) # 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, gender, test_size = 1-size
, stratify = gender)
        _, 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[24]:

<matplotlib.legend.Legend at 0x160b1590>



finetune parameters for support vector machines

kernel = linear, rbf or polynomial

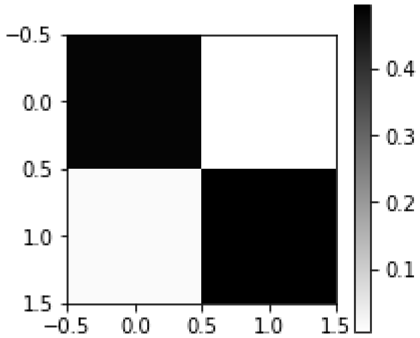
Conclusion: according to the results below, we set the kernel to be linear (mainly depends on acc here)

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 0.1

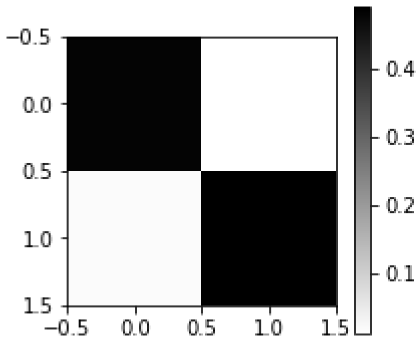
In [25]:

```
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, gender, stratify = gender)
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()
```

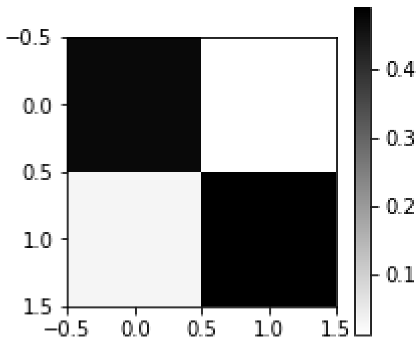
```
 svm kerel = linear
Building classifier：SVC....
 train set: acc=0.9739; f1=0.9739; auc=0.9941
 test set: acc=0.9760; f1=0.9763; auc=0.9923
 test confusion martix:
[[382  14]
 [  5 391]]
```

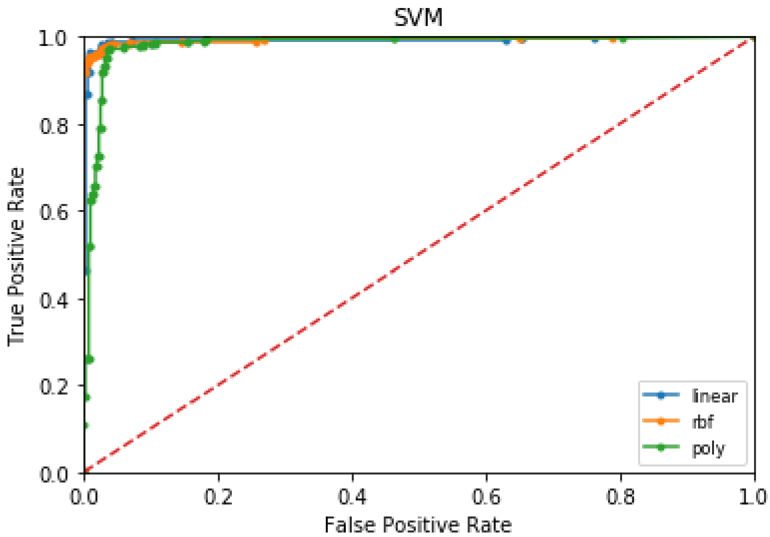



```
 svm kerel = rbf
Building classifier：SVC....

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=0.9705; f1=0.9707; auc=0.9964
 test set: acc=0.9722; f1=0.9724; auc=0.9935
 test confusion martix:
[[382  14]
 [  8 388]]
```




```
 svm kerel = poly
Building classifier：SVC....
 train set: acc=0.9676; f1=0.9681; auc=0.9906
 test set: acc=0.9583; f1=0.9591; auc=0.9824
 test confusion martix:
[[372  24]
 [  9 387]]
```

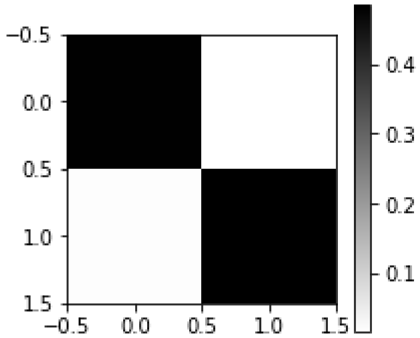

−0.5
0.0
0.5
1.0
1.5
−0.5
0.0
0.5
1.0
1.5
0.4
0.3
0.2
0.1


SVM
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
linear
rbf
poly

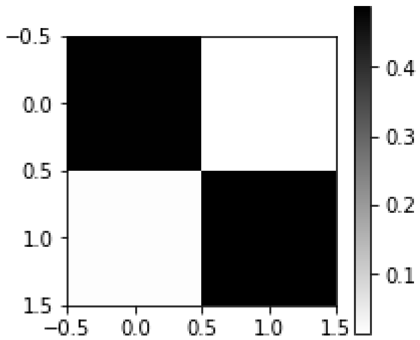
In [26]:

```
Clist = [.01, .1, .5, 1., 5., 10.]
X_train, X_test, y_train, y_test = train_test_split(features_svm, gender, stratify = gender)
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('linear-svm  C= %f'%C)
    clf = SVC(kernel='linear', 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 linear kernel')
plt.show()
```

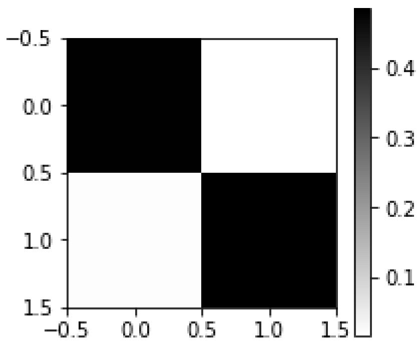
```
linear-svm  C= 0.010000
Building classifier: SVC....
 train set: acc=0.9752; f1=0.9752; auc=0.9932
 test set: acc=0.9684; f1=0.9686; auc=0.9943
 test confusion martix:
[[382  14]
 [ 11 385]]
```



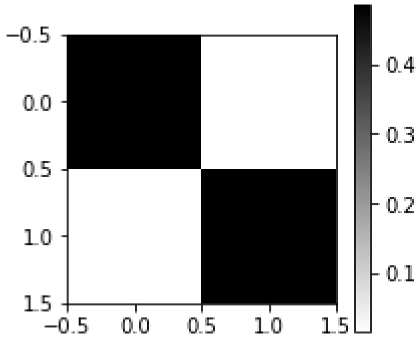

```
linear-svm  C= 0.100000
Building classifier: SVC....
 train set: acc=0.9764; f1=0.9765; auc=0.9932
 test set: acc=0.9710; f1=0.9711; auc=0.9946
 test confusion martix:
[[383  13]
 [ 10 386]]
```



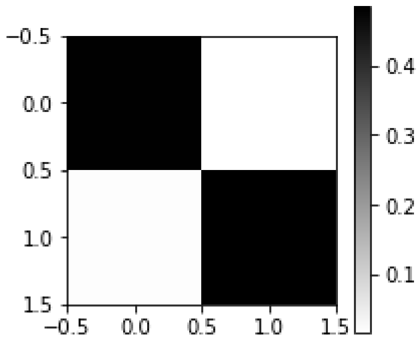

```
linear-svm  C= 0.500000
Building classifier: SVC....
 train set: acc=0.9764; f1=0.9765; auc=0.9932
 test set: acc=0.9672; f1=0.9673; auc=0.9947
 test confusion martix:
[[381  15]
 [ 11 385]]
```



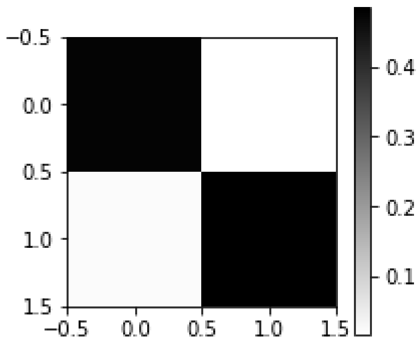
```
linear-svm  C= 1.000000
Building classifier: SVC....
 train set: acc=0.9773; f1=0.9773; auc=0.9931
 test set: acc=0.9697; f1=0.9698; auc=0.9946
 test confusion martix:
[[383  13]
 [ 11 385]]

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)
```



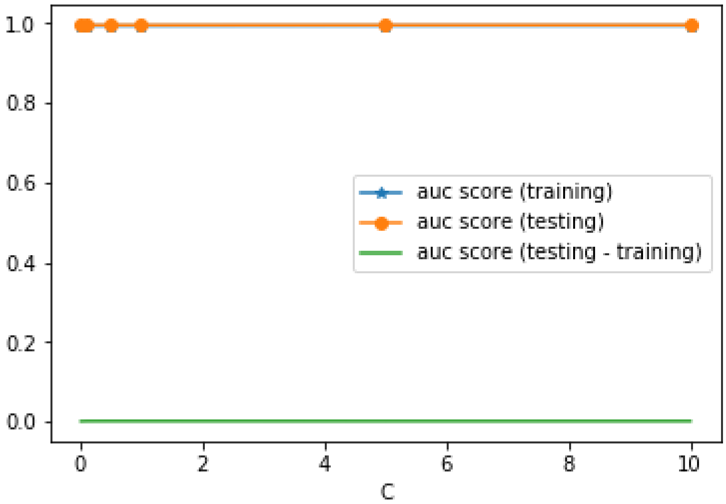

```
linear-svm  C= 5.000000
Building classifier: SVC....
 train set: acc=0.9752; f1=0.9753; auc=0.9933
 test set: acc=0.9672; f1=0.9673; auc=0.9947
 test confusion martix:
[[381  15]
 [ 11 385]]
```

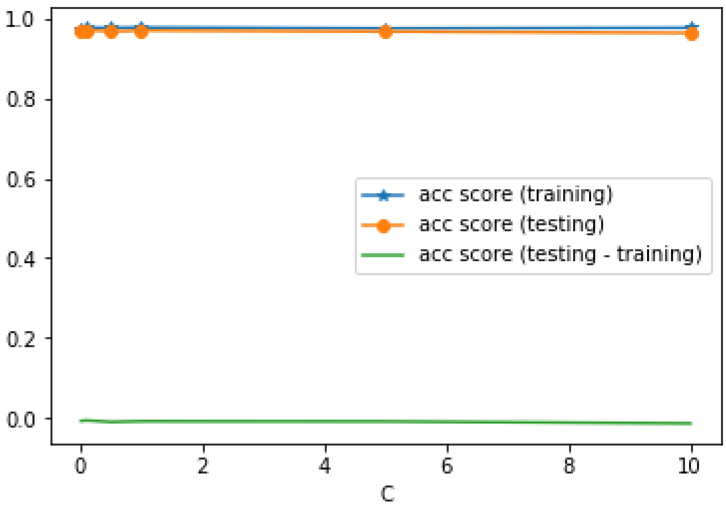


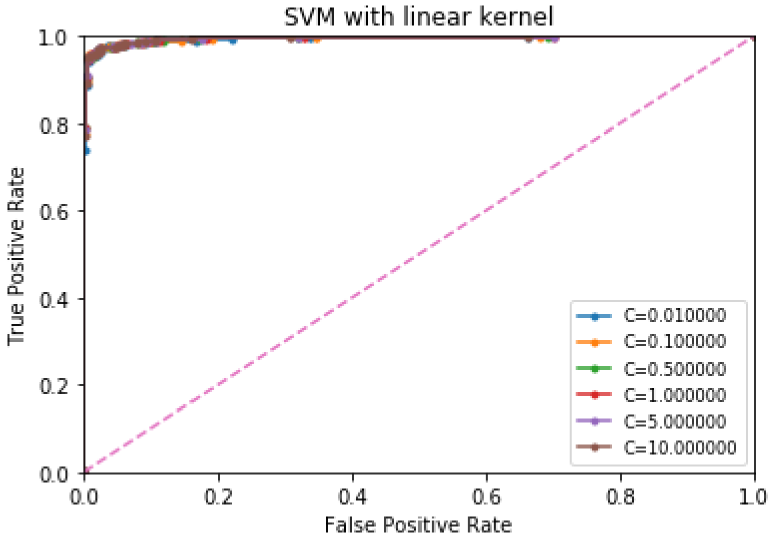

```
linear-svm  C= 10.000000
Building classifier: SVC....
 train set: acc=0.9764; f1=0.9765; auc=0.9934
 test set: acc=0.9634; f1=0.9637; auc=0.9948
 test confusion martix:
[[378  18]
 [ 11 385]]
```




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






SVM with linear kernel
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
C=0.010000
C=0.100000
C=0.500000
C=1.000000
C=5.000000
C=10.000000

In [32]:

```
train_size = [0.1*i for i in range(1, 10, 2)]
nfold = 5
clf = SVC(kernel='linear', C=0.1, 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, gender, test_size = 1-size, stratify = gender)
        _, train_metric, test_metric, t = train_predict(clf, X_train, y_train, X_test, y_test, silent=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 0x15c801f0>