In [1]:

```
#import sys
#from importlib import reload
import matplotlib.pyplot as plt
#reload(sys)
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
#sys.setdefaultencoding('utf-8')
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
from sklearn.datasets import fetch_20newsgroups
twenty_news = fetch_20newsgroups()
y = twenty_news.target
X = twenty_news.data
from sklearn.feature_extraction.text import TfidfVectorizer
# 初始化 TFIV 对象，去停用词，加 2 元语言模型
tfv = TfidfVectorizer(min_df=3, max_features=None, strip_accents='unicode', analyzer='word',token_p
# 提取特征，会有点慢
X = tfv.fit_transform(X)
#将数据分割训练数据与测试数据
from sklearn.model_selection import train_test_split
# 随机采样 20%的数据构建测试样本，其余作为训练样本
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33, test_size=0.2)
X_train.shape
print(np.max(y_train))
X_test.shape
```

19

Out[2]:

(2263, 155784)

In [3]:

```
# 多项朴素贝叶斯
from sklearn.naive_bayes import MultinomialNB
MNB = MultinomialNB()
MNB.fit(X_train, y_train) #特征数据直接灌进来
#输出每类的概率
#y_test_pred = MNB.predict_proba(X_test)
y_test_pred = MNB.predict(X_test)
```

In [4]:

```
print(metrics.classification_report(y_test, y_test_pred, target_names=twenty_news.target_names))
#print(metrics.confusion_matrix(y_test, y_test_pred))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
                          precision    recall  f1-score   support

             alt.atheism       0.91      0.94      0.93        89
           comp.graphics       0.75      0.87      0.81        99
 comp.os.ms-windows.misc       0.91      0.82      0.87       130
comp.sys.ibm.pc.hardware       0.73      0.84      0.78       109
   comp.sys.mac.hardware       0.98      0.92      0.95       117
          comp.windows.x       0.87      0.89      0.88       118
            misc.forsale       0.82      0.90      0.86       117
               rec.autos       0.91      0.94      0.93       121
         rec.motorcycles       0.97      0.92      0.94       119
      rec.sport.baseball       0.95      0.94      0.94       113
        rec.sport.hockey       0.93      0.97      0.95       129
               sci.crypt       0.88      0.99      0.93       109
         sci.electronics       0.96      0.79      0.87       120
                 sci.med       0.99      0.87      0.93       123
               sci.space       0.90      0.97      0.94       119
  soc.religion.christian       0.78      0.98      0.87       128
      talk.politics.guns       0.88      0.91      0.89       120
   talk.politics.mideast       0.97      1.00      0.99       100
      talk.politics.misc       0.97      0.84      0.90       106
      talk.religion.misc       1.00      0.43      0.60        77

                accuracy                           0.89      2263
               macro avg       0.90      0.89      0.89      2263
            weighted avg       0.90      0.89      0.89      2263

0.894387980556783
```

In [5]:

```
# 和逻辑回归比较
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
#设置超参数搜索范围
penaltys = ['l1','l2']
Cs = [ 0.1, 1, 10, 100, 1000]
tuned_parameters = dict(penalty = penaltys, C = Cs)
# LR 学习器实例
lr_penalty= LogisticRegression(tol=0.0001,solver='liblinear')
#GridSearchCV 实例
grid= GridSearchCV(lr_penalty, tuned_parameters, cv=5, scoring='neg_log_loss',n_jobs = 4,verbose=5)
# 模型训练
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[5]:

```
GridSearchCV(cv=5, estimator=LogisticRegression(solver='liblinear'), n_jobs=4,
             param_grid={'C': [0.1, 1, 10, 100, 1000], 'penalty': ['l1', 'l2']},
             scoring='neg_log_loss', verbose=5)
```

In [6]:

```
#输出结果
grid.cv_results_
```

Out[6]:


```
{'mean_fit_time': array([ 7.39196825,  4.13813906, 12.81704421,  7.14956408, 16.79
744592,
         9.20578589, 17.49883623, 11.52133584, 18.90248942, 11.52883091]),
 'std_fit_time': array([0.47701609, 0.23398088, 1.04536938, 0.22728291, 0.7888992
9,
        0.31959048, 1.24146655, 0.76398376, 0.64861013, 0.94209656]),
 'mean_score_time': array([0.02200632, 0.02980437, 0.0176054 , 0.02720714, 0.02580
667,
        0.02400441, 0.01920624, 0.02340622, 0.01759777, 0.01920452]),
 'std_score_time': array([0.00438074, 0.00574123, 0.00135735, 0.00549338, 0.007836
69,
        0.0085137 , 0.0033108 , 0.00531522, 0.00431342, 0.00676599]),
 'param_C': masked_array(data=[0.1, 0.1, 1, 1, 10, 10, 100, 100, 1000, 1000],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_penalty': masked_array(data=['l1', 'l2', 'l1', 'l2', 'l1', 'l2', 'l1', 'l
2', 'l1',
                    'l2'],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.1, 'penalty': 'l1'},
  {'C': 0.1, 'penalty': 'l2'},
  {'C': 1, 'penalty': 'l1'},
  {'C': 1, 'penalty': 'l2'},
  {'C': 10, 'penalty': 'l1'},
  {'C': 10, 'penalty': 'l2'},
  {'C': 100, 'penalty': 'l1'},
  {'C': 100, 'penalty': 'l2'},
  {'C': 1000, 'penalty': 'l1'},
  {'C': 1000, 'penalty': 'l2'}],
 'split0_test_score': array([-2.68464413, -2.59804463, -1.20554516, -1.35249755, -
0.49404397,
        -0.54840253, -0.45354637, -0.32876172, -0.50802064, -0.30339728]),
 'split1_test_score': array([-2.68184852, -2.59383524, -1.20859633, -1.33950593, -
0.47833202,
        -0.53062987, -0.41640139, -0.297626  , -0.45883863, -0.25860239]),
 'split2_test_score': array([-2.68326419, -2.59380639, -1.20772861, -1.34895369, -
0.46682724,
        -0.54316612, -0.39316634, -0.31159045, -0.43072041, -0.27705888]),
 'split3_test_score': array([-2.68692361, -2.59756655, -1.23480395, -1.36139768, -
0.49004467,
        -0.55607315, -0.44501474, -0.33266075, -0.47414506, -0.30758812]),
 'split4_test_score': array([-2.7006154 , -2.6034643 , -1.25301447, -1.37458984, -
0.50049238,
        -0.55914097, -0.45510682, -0.32609854, -0.52415495, -0.29194738]),
 'mean_test_score': array([-2.68745917, -2.59734342, -1.2219377 , -1.35538894, -0.
48594806,
        -0.54748253, -0.43264713, -0.31934749, -0.47917594, -0.28771881]),
 'std_test_score': array([0.00678785, 0.0035447 , 0.01886749, 0.01188968, 0.011978
```

```
92,
        0.01013122, 0.02415297, 0.01298801, 0.03359162, 0.01799936]),
 'rank_test_score': array([10  9  7  8  5  6  3  2  4  1])}
```

In [7]:

```
grid.best_estimator_
```

Out[7]:

```
LogisticRegression(C=1000, solver='liblinear')
```

In [8]:

```
#设置超参数搜索范围
penaltys = ['l1','l2']
Cs = [10000,100000]
tuned_parameters = dict(penalty = penaltys, C = Cs)
# LR 学习器实例
lr_penalty= LogisticRegression(tol=0.0001,solver='liblinear')
#GridSearchCV 实例
grid= GridSearchCV(lr_penalty, tuned_parameters, cv=5, scoring='neg_log_loss',n_jobs = 4,verbose=5)
# 模型训练
grid.fit(X_train,y_train)
#输出结果
grid.cv_results_
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

Out[8]:

```
{'mean_fit_time': array([19.52955432, 15.01517234, 22.9392303 , 14.24248667]),
 'std_fit_time': array([2.59542371, 0.72952275, 1.23048041, 2.07108293]),
 'mean_score_time': array([0.02020464, 0.02343531, 0.02638512, 0.02620716]),
 'std_score_time': array([0.00515505, 0.00652827, 0.00478723, 0.00932488]),
 'param_C': masked_array(data=[10000, 10000, 100000, 100000],
              mask=[False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_penalty': masked_array(data=['l1', 'l2', 'l1', 'l2'],
              mask=[False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 10000, 'penalty': 'l1'},
  {'C': 10000, 'penalty': 'l2'},
  {'C': 100000, 'penalty': 'l1'},
  {'C': 100000, 'penalty': 'l2'}],
 'split0_test_score': array([-0.42349949, -0.33380673, -0.3953851 , -0.37466618]),
 'split1_test_score': array([-0.34068551, -0.27912095, -0.3180441 , -0.30968601]),
 'split2_test_score': array([-0.34677532, -0.30215746, -0.34312415, -0.33673667]),
 'split3_test_score': array([-0.40022814, -0.34264629, -0.39090289, -0.38818853]),
 'split4_test_score': array([-0.41385386, -0.3173014 , -0.39867542, -0.35430679]),
 'mean_test_score': array([-0.38500846, -0.31500657, -0.36922633, -0.35271684]),
 'std_test_score': array([0.0345588 , 0.02269601, 0.03262621, 0.02774729]),
 'rank_test_score': array([4, 1, 3, 2])}
```

In [9]:

```
y_test_pred = grid.best_estimator_.predict(X_test)
print(metrics.classification_report(y_test, y_test_pred, target_names=twenty_news.target_names))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
                          precision    recall  f1-score   support

             alt.atheism       0.97      0.94      0.95        89
           comp.graphics       0.80      0.84      0.82        99
 comp.os.ms-windows.misc       0.89      0.91      0.90       130
comp.sys.ibm.pc.hardware       0.86      0.80      0.83       109
   comp.sys.mac.hardware       0.92      0.93      0.92       117
          comp.windows.x       0.92      0.89      0.91       118
            misc.forsale       0.87      0.94      0.90       117
               rec.autos       0.95      0.94      0.95       121
         rec.motorcycles       0.96      0.97      0.97       119
      rec.sport.baseball       0.98      0.99      0.99       113
        rec.sport.hockey       1.00      0.97      0.98       129
               sci.crypt       0.96      0.98      0.97       109
         sci.electronics       0.91      0.93      0.92       120
                 sci.med       0.99      0.91      0.95       123
               sci.space       0.96      0.97      0.97       119
  soc.religion.christian       0.93      0.98      0.95       128
      talk.politics.guns       0.97      0.96      0.96       120
   talk.politics.mideast       1.00      1.00      1.00       100
      talk.politics.misc       0.95      0.94      0.95       106
      talk.religion.misc       0.94      0.86      0.90        77

                accuracy                           0.94      2263
               macro avg       0.94      0.93      0.93      2263
            weighted avg       0.94      0.94      0.94      2263

0.9354838709677419
```