In [1]:

```
from os import listdir
from os.path import isfile, join
from bs4 import BeautifulSoup
import xml.etree.ElementTree as ET
import codecs
mypath ="/Anaconda/blogs"
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cross_validation import train_test_split
from time import time
import sys
import scipy.sparse as sp
import pylab as pl
import cPickle
```

In [2]:

```
from sklearn.datasets import load_mlcomp
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
```

In [3]:

```
def benchmark(clf_class, params, name):
    print("parameters:", params)
    t0 = time()
    clf = clf_class(**params).fit(Xtrain, y_train)
    print("done in %fs" % (time() - t0))

    if hasattr(clf, 'coef_'):
        print("Percentage of non zeros coef: %f"
              % (np.mean(clf.coef_ != 0) * 100))
    print("Predicting the outcomes of the testing set")
    t0 = time()
    pred = clf.predict(Xtest)
    print("done in %fs" % (time() - t0))

    print("Classification report on test set for classifier:")
    print(clf)
    print()
    print(classification_report(y_test, pred))

    with open(name+'gender.pkl', 'wb') as fid:
        cPickle.dump(clf, fid)

    cm = confusion_matrix(y_test, pred)
    print("Confusion matrix:")
    print(cm)

    # Show confusion matrix
    pl.matshow(cm)
    pl.title('Confusion matrix of the %s classifier' % name)
    pl.colorbar()
```

In [5]:

```
onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]
```

In [6]:

```
def make_documentIndex(listOfDocuments):
    dict_of_words = {}
    doc_corpus = []
    for doc in listOfDocuments:
        file = open(mypath+"/"+doc)
        str = file.read()
        doc_corpus.append(str)



    return doc_corpus
```

In [7]:

```
corpus = make_documentIndex(onlyfiles)
```

In [7]:

```
len(corpus)
```

Out[7]:

19320

In [8]:

```
def HandleDataAndLabels(coprus,onlyfiles):
    Y = []
    X = []
    numFiles = len(onlyfiles)
    for f in range(0,numFiles):
        gender = onlyfiles[f].split('.')[1]

        soup = BeautifulSoup(corpus[f],"lxml")
        result= soup.findAll('post')
        for a in result:
            X.append(a.get_text())
            Y.append(gender)

    return X,Y
```

In [9]:

```
X,Y = HandleDataAndLabels(corpus,onlyfiles)
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.7)
```

In [10]:

```
print "Original Data X,Y",len(X),len(Y)
print "Xtrain,Ytrain ",len(X_train),len(y_train)
print "XTest, YTest ",len(X_test),len(y_test)
```

```
Original Data X,Y 681253 681253
Xtrain,Ytrain  476877 476877
XTest, YTest  204376 204376
```

In [ ]:

```

```

In [10]:

```
t0 = time()
vectorizer = TfidfVectorizer(encoding='utf-8')
Xtrain = vectorizer.fit_transform(X_train)
print("done in %fs" % (time() - t0))
print("n_samples: %d, n_features: %d" % Xtrain.shape)
assert sp.issparse(Xtrain)
```

```
done in 113.136000s
n_samples: 476877, n_features: 586906
```

In [11]:

```
with open('TFIDF_Age_Vectorizer.pkl', 'wb') as fid:
    cPickle.dump(vectorizer, fid)
```

In [12]:

```
t0 = time()
Xtest = vectorizer.transform(X_test)

print("done in %fs" % (time() - t0))
print("n_samples: %d, n_features: %d" % Xtest.shape)
```

```
done in 44.345000s
n_samples: 204376, n_features: 586906
```

In [14]:

```
t0 = time()
clf = RandomForestClassifier(n_estimators=100,max_depth=32,n_jobs=4,verbose=True)
clf.fit(Xtrain, y_train)
print("done in %fs" % (time() - t0))

if hasattr(clf, 'coef_'):
    print("Percentage of non zeros coef: %f"
          % (np.mean(clf.coef_ != 0) * 100))
print("Predicting the outcomes of the testing set")
t0 = time()
pred = clf.predict(Xtest)
print("done in %fs" % (time() - t0))

print("Classification report on test set for classifier:")
print(clf)
print()
print(classification_report(y_test, pred))
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:  2.9min
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:  6.5min finished
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    2.5s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    5.7s finished

done in 395.579000s
Predicting the outcomes of the testing set
done in 6.061000s
Classification report on test set for classifier:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
            max_depth=32, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=4,
            oob_score=False, random_state=None, verbose=True,
            warm_start=False)
()
             precision    recall  f1-score   support

          1       0.92      0.23      0.36     70783
          2       0.51      0.99      0.67     96334
          3       0.97      0.01      0.01     37259

avg / total       0.74      0.54      0.44    204376
```

In [15]:

```
"""""
importances = clf.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf.estimators_],axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(Xtrain.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(Xtrain.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
  File "<ipython-input-15-00f4c2db25c8>", line 19
    plt.show()

^
SyntaxError: EOF while scanning triple-quoted string literal
```

In [15]:

```
with open('RF_classifier2.pkl', 'wb') as fid:
    cPickle.dump(clf, fid)

# load it again
#with open('my_dumped_classifier.pkl', 'rb') as fid:
 #   gnb_loaded = cPickle.load(fid)
```

In [13]:

```
print("Testbenching a MultinomialNB classifier...")
parameters = {'alpha': 0.01}

benchmark(MultinomialNB, parameters, 'MultinomialNB')
pl.show()
```

```
Testbenching a MultinomialNB classifier...
('parameters:', {'alpha': 0.01})
done in 1.312000s
Percentage of non zeros coef: 100.000000
Predicting the outcomes of the testing set
done in 0.218000s
Classification report on test set for classifier:
MultinomialNB(alpha=0.01, class_prior=None, fit_prior=True)
()
             precision    recall  f1-score   support

     female       0.68      0.74      0.71    100842
       male       0.73      0.66      0.69    103534

avg / total       0.71      0.70      0.70    204376

Confusion matrix:
[[75026 25816]
 [34786 68748]]
```

In [17]:

```
print("Testbenching a linear classifier...")
parameters = {
    'loss': 'hinge',
    'penalty': 'l2',
    'n_iter': 100,
    'alpha': 0.00001,
    'fit_intercept': True,
}

benchmark(SGDClassifier, parameters, 'SGD3')
pl.show()
```

```
Testbenching a linear classifier...
('parameters:', {'penalty': 'l2', 'loss': 'hinge', 'alpha': 0.0001, 'fi
t_intercept': True, 'n_iter': 100})
done in 38.035000s
Percentage of non zeros coef: 89.554375
Predicting the outcomes of the testing set
done in 0.080000s
Classification report on test set for classifier:
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=
0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', n_iter=100, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=None, shuffle=True,
       verbose=0, warm_start=False)
()
             precision    recall  f1-score   support

     female       0.70      0.58      0.64    100842
       male       0.65      0.75      0.70    103534

avg / total       0.67      0.67      0.67    204376

Confusion matrix:
[[58906 41936]
 [25370 78164]]
```

In [16]:

```
print("Testbenching a logistic Regression...")
parameters = {
    'loss': 'log',
    'penalty': 'l2',
    'n_iter': 250,
    'alpha': 0.000001,
    'fit_intercept': True,
    'verbose':True,
}

benchmark(SGDClassifier, parameters, 'SGD')
pl.show()
```

```
Total training time: 17.82 seconds.
-- Epoch 41
Norm: 241.44, NNZs: 589993, Bias: -1.590145, T: 19551957, Avg. loss:
0.307190
Total training time: 18.23 seconds.
-- Epoch 42
Norm: 241.42, NNZs: 589993, Bias: -1.620475, T: 20028834, Avg. loss:
0.306931
Total training time: 18.68 seconds.
-- Epoch 43
Norm: 241.38, NNZs: 589993, Bias: -1.606789, T: 20505711, Avg. loss:
0.306683
Total training time: 19.10 seconds.
-- Epoch 44
Norm: 241.36, NNZs: 589993, Bias: -1.608439, T: 20982588, Avg. loss:
0.306446
Total training time: 19.55 seconds.
-- Epoch 45
Norm: 241.34, NNZs: 589993, Bias: -1.604055, T: 21459465, Avg. loss:
0.306219
```

In [18]:

```
print x.shape
```

```
(681253, 724796)
```

In [ ]:

```

```