In [16]:
```python
import pandas as pd

df = pd.read_csv('train_set.csv', sep='\t')

from os import path
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS

sw = STOPWORDS
# insert my_stopwords
sw.update(['one','say','first','second','new','two','will','us','also', \
        'said','U','.', ',', '"', "'", '?', '!', ':', ';', '(', ')', '[', ']', '{', '}'])
```

Technology WordCloud

In [17]:
```python
technology = df[df["Category"] == "Technology"]
wordcloud_technology = WordCloud(width=600,height=400,stopwords=sw).generate(' '.join(technology["Content"]+technology["Title"])
 )
plt.figure()
plt.imshow(wordcloud_technology, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Business WordCloud

In [18]:
```python
business = df[df["Category"] == "Business"]
wordcloud_business = WordCloud(width=600,height=400,stopwords=sw).generate(' '.join(business["Content"]+business["Title"]))
plt.figure()
plt.imshow(wordcloud_business, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Film WordCloud

In [19]:
```python
film = df[df["Category"] == "Film"]
wordcloud_film = WordCloud(width=600,height=400,stopwords=sw).generate(' '.join(film["Content"]+film["Title"]))
plt.figure()
plt.imshow(wordcloud_film, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Football WordCloud

In [20]:
```python
football = df[df["Category"] == "Football"]
wordcloud_football = WordCloud(width=600,height=400,stopwords=sw).generate(' '.join(football["Content"]+football["Title"]))
plt.figure()
plt.imshow(wordcloud_football, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Politics WordCloud

In [21]:
```python
politics = df[df["Category"] == "Politics"]
wordcloud_politics = WordCloud(width=600,height=400,stopwords=sw).generate(' '.join(politics["Content"]+politics["Title"]))
plt.figure()
plt.imshow(wordcloud_politics, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Clustering

In [7]:
```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.decomposition import TruncatedSVD

#Ignore Category column
df1 = df.drop(['Category'], axis=1)

count_vect = CountVectorizer(stop_words=sw)
X_train_counts = count_vect.fit_transform(df1.Content)
transformer = TfidfTransformer(smooth_idf=False)
X_train_counts=transformer.fit_transform(X_train_counts)
svd = TruncatedSVD(n_components=100)
X_train_counts=svd.fit_transform(X_train_counts)

import numpy as np
from sklearn.cluster import KMeans
import nltk
from nltk.cluster.kmeans import KMeansClusterer
from nltk.cluster import cosine_distance

# avoid dividing by 0
from numpy.linalg import norm
for v in X_train_counts:
    if norm(v,2) == 0:
        v += 1
        v /= norm(v,2)

NUM_CLUSTERS = 5
kclusterer = KMeansClusterer(NUM_CLUSTERS, distance=cosine_distance, repeats=10)
assigned_clusters = kclusterer.cluster(X_train_counts, assign_clusters=True)
df["Category"] = df["Category"].map({"Politics": 0, "Business": 1,"Football": 2,"Film": 3, "Technology": 4})

df.head(5)
```

Out[7]:

	RowNum	Id	Title	Content	Category
0	9560	9561	Sam Adams founder: Beer is more than just 'col...	The craft beer boom, which and been attributed...	1
1	10801	10802	Slump in oil prices could mean fall in investm...	The International Energy Agency has warned tha...	1
2	6726	6727	British Gas owner Centrica warns of higher gas...	Senior executives at British have been accused...	1
3	12365	12366	Ole Gunnar Solskjaer appointed manager of Card...	is confident he will have complete control of...	2
4	11782	11783	Sunderland target loan signings of Kurt Zouma ...	Kurt Zouma and Jack Rodwell are on Sunderland'...	2

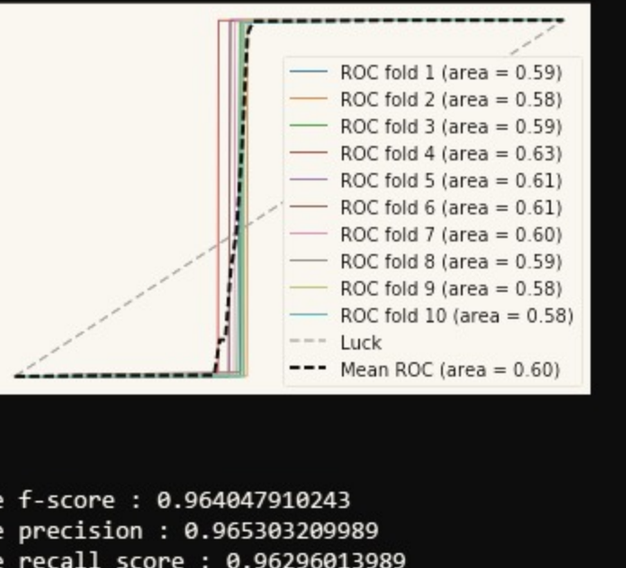
In [8]:
```python
A = np.array(df)
myarray = np.zeros((5,5))

from collections import Counter

#Cluster calculation by category
cnt=Counter()
for i, val in enumerate(assigned_clusters):
    myarray[val,A[i,4]] += 1
    cnt[val] += 1
for i in range(myarray.shape[0]):
    for j in range(myarray.shape[1]):
        myarray[i,j] /= cnt[i]
```

Save clusters @ .csv file

In [9]:
```python
titles= ["Politics","Business","Football","Film","Technology"]
clusterpoints= ["Cluster1","Cluster2","Cluster3","Cluster4","Cluster5"]
mycsv = pd.DataFrame(data=myarray,index=clusterpoints,columns=titles)
mycsv.to_csv("clustering_KMeans.csv")
```

Best classification method : SVC_linear

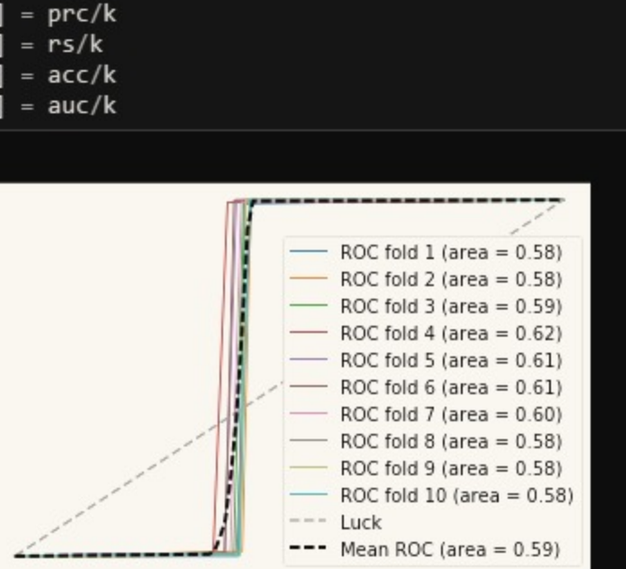
In [10]:
```python
Y = df["Category"]

from scipy import interp

from sklearn import svm
from sklearn.model_selection import GridSearchCV

count_vect = CountVectorizer(stop_words=sw)
vectorizer = TfidfTransformer()
X = df["Title"] + df["Content"]

from sklearn.model_selection import KFold
from sklearn.model_selection import ShuffleSplit
k = 10
kf = ShuffleSplit(n_splits=k, test_size=0.1, random_state=0)
fold = 0

from sklearn import metrics
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_curve, auc
#metrics initialization
base_fpr = np.linspace(0, 1, 101)
fpr = tpr = thresholds = f1 = prc = rs = acc = auc =  0.0
mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)

for train_index, test_index in kf.split(X):

    X_train_counts = count_vect.fit_transform(X[train_index])
    X_train_counts = vectorizer.fit_transform(X_train_counts)
    svd = TruncatedSVD(n_components=200)
    X_lsi = svd.fit_transform(X_train_counts)

    parameters = {'C':[1, 10]}
    svr = svm.LinearSVC()
    clf = GridSearchCV(svr, parameters)

    clf = clf.fit(X_lsi, Y[train_index])

    X_test_counts = count_vect.transform(X[test_index])
    X_test_counts = vectorizer.transform(X_test_counts)
    X_test_counts = svd.transform(X_test_counts)
    yPred = clf.predict(X_test_counts)

    fold += 1

    # Calculating metrics
    f1 += f1_score(yPred,Y[test_index],average="macro")
    prc += precision_score(yPred,Y[test_index],average="macro")
    rs += recall_score(yPred,Y[test_index],average="macro")
    acc += accuracy_score(yPred, Y[test_index])

    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(yPred,Y[test_index], pos_label = 2)
    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = metrics.auc(fpr,tpr)
    auc += roc_auc
    plt.plot(fpr, tpr, lw=1, label='ROC fold %d (area = %0.2f)' % (fold, roc_auc))

plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')

mean_tpr /= kf.get_n_splits(X, Y)
mean_tpr[-1] = 1.0
mean_auc = metrics.auc(mean_fpr, mean_tpr)

plt.plot(mean_fpr, mean_tpr, 'k--',
         label='Mean ROC (area = %0.2f)' % mean_auc, lw=2)

#roc_10fold
roc10 = np.zeros((4,3),dtype=object) #saving values for roc_10fold plot
roc10[0,0] = mean_fpr
roc10[0,1] = mean_tpr
roc10[0,2] = mean_auc

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('SVC_linear classifier\'s roc plot')
plt.legend(loc="lower right")
plt.show()

print "Average f-score : " + str(f1/k)
print "Average precision : " + str(prc/k)
print "Average recall_score : " + str(rs/k)
print "Average Accuracy : " + str(acc/k)
print "Average AUC : " + str(auc/k)

em = np.zeros((4,5)) #evaluation metric matrix
em[0,0] = f1/k
em[0,1] = prc/k
em[0,2] = rs/k
em[0,3] = acc/k
em[0,4] = auc/k
```




```
Average f-score : 0.964047910243
Average precision : 0.965303209989
Average recall_score : 0.96296013989
Average Accuracy : 0.966422167889
Average AUC : 0.595803571553
```

GaussianNB

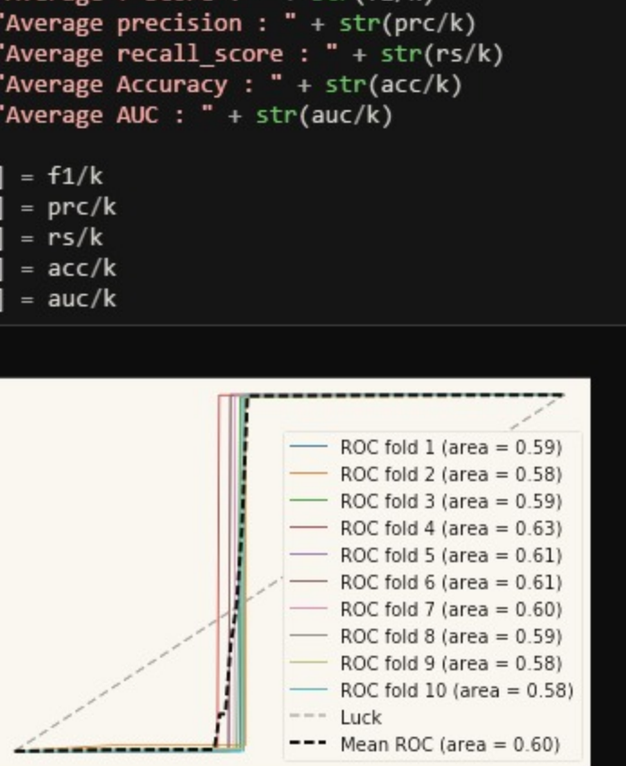
In [11]:
```python
from sklearn.naive_bayes import GaussianNB

kf = ShuffleSplit(n_splits=k, test_size=0.1, random_state=0)
fold = 0
fpr = tpr = thresholds = f1 = prc = rs = acc = auc =  0
mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)

for train_index, test_index in kf.split(X):

    X_train_counts = count_vect.fit_transform(X[train_index])
    X_train_counts = vectorizer.fit_transform(X_train_counts)
    svd = TruncatedSVD(n_components=200)
    X_lsi = svd.fit_transform(X_train_counts)

    clf = GaussianNB()

    clf = clf.fit(X_lsi, Y[train_index])

    X_test_counts = count_vect.transform(X[test_index])
    X_test_counts = vectorizer.transform(X_test_counts)
    X_test_counts = svd.transform(X_test_counts)
    yPred = clf.predict(X_test_counts)

    fold += 1

    # Calculating metrics
    f1 += f1_score(yPred,Y[test_index],average="macro")
    prc += precision_score(yPred,Y[test_index],average="macro")
    rs += recall_score(yPred,Y[test_index],average="macro")
    acc += accuracy_score(yPred, Y[test_index])

    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(yPred,Y[test_index], pos_label = 2)
    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = metrics.auc(fpr,tpr)
    auc += roc_auc
    plt.plot(fpr, tpr, lw=1, label='ROC fold %d (area = %0.2f)' % (fold, roc_auc))

plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')
mean_tpr /= kf.get_n_splits(X, Y)
mean_tpr[-1] = 1.0
mean_auc = metrics.auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, 'k--',
         label='Mean ROC (area = %0.2f)' % mean_auc, lw=2)

roc10[1,0] = mean_fpr
roc10[1,1] = mean_tpr
roc10[1,2] = mean_auc

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('GaussianNB classifier\'s roc plot')
plt.legend(loc="lower right")
plt.show()

print "Average f-score : " + str(f1/k)
print "Average precision : " + str(prc/k)
print "Average recall_score : " + str(rs/k)
print "Average Accuracy : " + str(acc/k)
print "Average AUC : " + str(auc/k)
em[1,0] = f1/k
em[1,1] = prc/k
em[1,2] = rs/k
em[1,3] = acc/k
em[1,4] = auc/k
```




```
Average f-score : 0.879356532218
Average precision : 0.881800629534
Average recall_score : 0.883028066722
Average Accuracy : 0.887367563162
Average AUC : 0.594693052123
```

RandomForest Classifier

In [12]:
```python
from sklearn.ensemble import RandomForestClassifier

kf = ShuffleSplit(n_splits=k, test_size=0.1, random_state=0)
fold = 0
fpr = tpr = thresholds = f1 = prc = rs = acc = auc =  0

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)

for train_index, test_index in kf.split(X):

    X_train_counts = count_vect.fit_transform(X[train_index])
    X_train_counts = vectorizer.fit_transform(X_train_counts)
    svd = TruncatedSVD(n_components=200)
    X_lsi = svd.fit_transform(X_train_counts)

    clf = RandomForestClassifier(n_estimators = 20 ,n_jobs = -1)

    clf = clf.fit(X_lsi, Y[train_index])

    X_test_counts = count_vect.transform(X[test_index])
    X_test_counts = vectorizer.transform(X_test_counts)
    X_test_counts = svd.transform(X_test_counts)
    yPred = clf.predict(X_test_counts)

    fold += 1

    # Calculating metrics
    f1 += f1_score(yPred,Y[test_index],average="macro")
    prc += precision_score(yPred,Y[test_index],average="macro")
    rs += recall_score(yPred,Y[test_index],average="macro")
    acc += accuracy_score(yPred, Y[test_index])

    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(yPred,Y[test_index], pos_label = 2)
    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = metrics.auc(fpr,tpr)
    auc += roc_auc
    plt.plot(fpr, tpr, lw=1, label='ROC fold %d (area = %0.2f)' % (fold, roc_auc))

plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')

mean_tpr /= kf.get_n_splits(X, Y)
mean_tpr[-1] = 1.0
mean_auc = metrics.auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, 'k--',
         label='Mean ROC (area = %0.2f)' % mean_auc, lw=2)

roc10[2,0] = mean_fpr
roc10[2,1] = mean_tpr
roc10[2,2] = mean_auc
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('RandomForest classifier\'s roc plot')
plt.legend(loc="lower right")
plt.show()

print "Average f-score : " + str(f1/k)
print "Average precision : " + str(prc/k)
print "Average recall_score : " + str(rs/k)
print "Average Accuracy : " + str(acc/k)
print "Average AUC : " + str(auc/k)

em[2,0] = f1/k
em[2,1] = prc/k
em[2,2] = rs/k
em[2,3] = acc/k
em[2,4] = auc/k
```




```
Average f-score : 0.955523931946
Average precision : 0.955949945872
Average recall_score : 0.955269586273
Average Accuracy : 0.958761206194
Average AUC : 0.596414097114
```

KNN

In [ ]:
```python
import math
def euclideanDistance(instance1, instance2):
    mylist1=[]
    mylist2=[]
    for i in instance1:
        mylist1.append(i)
    for j in instance2:
        mylist2.append(j)
    points = zip(mylist1,mylist2)
    diffs_squared_distance = [pow(a - b, 2) for (a, b) in points]
    return math.sqrt(sum(diffs_squared_distance))

import operator

def tuple_distance(training_instance, test_instance,val):
    return (val, euclideanDistance(test_instance, training_instance))

def getKneighbors(trainingSet, testInstance, k):
    distances = [tuple_distance(training_instance, testInstance,val) for val,training_instance in enumerate(trainingSet)]
    # index 1 is the calculated distance between training_instance and test_instance
    distances.sort(key=operator.itemgetter(1))
    # extract only training instances
    sorted_training_instances = [tuple[0] for tuple in distances]
    # select first k elements
    return sorted_training_instances[:k]

from collections import Counter

def getResponse(neighbors):
    classes = [neighbour[4] for neighbour in neighbors]
    cnt=Counter()
    for x in classes:
        if x in cnt:
            cnt[x] +=1
        else:
            cnt[x] = 1
    majority_sorted = sorted(cnt.items(),key=operator.itemgetter(0),reverse=True)
    return majority_sorted[0][0]

kf = ShuffleSplit(n_splits=k, test_size=0.1, random_state=0)
fold = 0
fpr = tpr = thresholds = f1 = prc = rs = acc = auc =  0

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)

for train_index, test_index in kf.split(X):

    X_train_counts = count_vect.fit_transform(X[train_index])
    X_train_counts = vectorizer.fit_transform(X_train_counts)
    svd = TruncatedSVD(n_components=200)
    X_lsi = svd.fit_transform(X_train_counts)

    X_test_counts = count_vect.transform(X[test_index])
    X_test_counts = vectorizer.transform(X_test_counts)
    X_test_counts = svd.transform(X_test_counts)
    ypred=[]
    neighlist=[]
    A = np.array(df)
    mycount=0
    for i in X_test_counts:
        neighbors=getKneighbors(X_lsi,i,10)
        neighlist=[A[j,:] for j in neighbors]
        myres = getResponse(neighlist)
        ypred.append(myres)
        neighlist=[]
        print ypred
        mycount += 1

    fold += 1
    print str(fold)
    # Calculating metrics
    f1 += f1_score(yPred,Y[test_index],average="macro")
    prc += precision_score(yPred,Y[test_index],average="macro")
    rs += recall_score(yPred,Y[test_index],average="macro")
    acc += accuracy_score(yPred, Y[test_index])

    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(yPred,Y[test_index], pos_label = 2)
    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = metrics.auc(fpr,tpr)
    auc += roc_auc
    plt.plot(fpr, tpr, lw=1, label='ROC fold %d (area = %0.2f)' % (fold, roc_auc))

plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')
mean_tpr /= kf.get_n_splits(X, Y)
mean_tpr[-1] = 1.0
mean_auc = metrics.auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, 'k--',
         label='Mean ROC (area = %0.2f)' % mean_auc, lw=2)

roc10[3,0] = mean_fpr
roc10[3,1] = mean_tpr
roc10[3,2] = mean_auc

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('GaussianNB classifier\'s roc plot')
plt.legend(loc="lower right")
plt.show()

print( "Average f-score : " + str(f1/k))
print( "Average precision : " + str(prc/k))
print( "Average recall_score : " + str(rs/k))
print( "Average Accuracy : " + str(acc/k) )
print( "Average AUC : " + str(auc/k))
em[3,0] = f1/k
em[3,1] = prc/k
em[3,2] = rs/k
em[3,3] = acc/k
em[3,4] = auc/k
```

Output Files

In [14]:
```python
#Evaluation metric
rows = ["SVC_linear","GaussianNB","RandomForest","KNN"]
columns = ["F-measure","Precision","Recall_score","Accuracy","AUC"]
mycsv = pd.DataFrame(data=em,index=rows,columns=columns)
mycsv.to_csv("EvaluationMetric_10fold.csv")
print "Created EvaluationMetric_10fold.csv file!"

#roc_10fold
plt.plot(roc10[0,0], roc10[0,1], 'k--',
         label='SVClinear Mean_ROC (area = %0.2f)' % roc10[0,2], lw=2)
plt.plot(roc10[1,0], roc10[1,1], 'k--',
         label='GNB Mean_ROC (area = %0.2f)' %  roc10[1,2], lw=2)
plt.plot(roc10[2,0], roc10[2,1], 'k--',
         label='RF Mean_ROC (area = %0.2f)' %  roc10[2,2], lw=2)
plt.plot(roc10[3,0], roc10[3,1], 'k--',
         label='KNN Mean_ROC (area = %0.2f)' %  roc10[3,2], lw=2)

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Classifiers\'s mean_roc plot')
plt.legend(loc="lower right")
plt.savefig("roc_10fold.png")
plt.show()
print "Created roc_10fold.png file!"
```

```
Created EvaluationMetric_10fold.csv file!
```


```
Created roc_10fold.png file!
```

In [15]:
```python
test = pd.read_csv('test_set.csv', sep='\t')
testX = test["Title"] + test["Content"]

X_train_counts = count_vect.fit_transform(X)
X_train_counts = vectorizer.fit_transform(X_train_counts)
svd = TruncatedSVD(n_components=200)
X_lsi = svd.fit_transform(X_train_counts)

parameters = {'C':[1, 10]}
svr = svm.LinearSVC()
clf = GridSearchCV(svr, parameters)

clf = clf.fit(X_lsi, Y)

X_test_counts = count_vect.transform(testX)
X_test_counts = vectorizer.transform(X_test_counts)
X_test_counts = svd.transform(X_test_counts)
predicted = clf.predict(X_test_counts)

output = np.zeros((len(predicted),2),dtype=object)

for i,j in zip(predicted,range(len(predicted))):
    if i == 0:
        output[j][0] = test.iloc[j]["Id"]
        output[j][1] = "Politics"
    elif i == 1:
        output[j][0] = test.iloc[j]["Id"]
        output[j][1] = "Business"
    elif i == 2:
        output[j][0] = test.iloc[j]["Id"]
        output[j][1] = "Football"
    elif i == 3:
        output[j][0] = test.iloc[j]["Id"]
        output[j][1] = "Film"
    elif i == 4:
        output[j][0] = test.iloc[j]["Id"]
        output[j][1] = "Technology"

mycsv = pd.DataFrame(data=output,columns=["ID","Predicted_Category"])
mycsv.to_csv("testSet_categories.csv", sep='\t',index=False, header=False)
print "Created testSet_categories.csv file!"
```

```
Created testSet_categories.csv file!
```

In [ ]: