Sentiment Analysis on movie reviews

Load the Data from Source

```
import os
import tarfile
from contextlib import closing
try:
    from urllib import urlopen
except ImportError:
    from urllib.request import urlopen


URL = ("http://www.cs.cornell.edu/people/pabo/"
       "movie-review-data/review_polarity.tar.gz")

ARCHIVE_NAME = URL.rsplit('/', 1)[1]
DATA_FOLDER = "txt_sentoken"


if not os.path.exists(DATA_FOLDER):

    if not os.path.exists(ARCHIVE_NAME):
        print("Downloading dataset from %s (3 MB)" % URL)
        opener = urlopen(URL)
        with open(ARCHIVE_NAME, 'wb') as archive:
            archive.write(opener.read())

    print("Decompressing %s" % ARCHIVE_NAME)
    with closing(tarfile.open(ARCHIVE_NAME, "r:gz")) as archive:
        archive.extractall(path='.')
    os.remove(ARCHIVE_NAME)
else:
    print("Dataset already exists")
```

```
Dataset already exists
```

Global Imports

```
import numpy as np
import pandas as pa
import matplotlib.pylab as py
import matplotlib.pyplot as plt
import scipy
from time import time

%matplotlib inline
```

Load data

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline
from sklearn.grid_search import GridSearchCV
from sklearn.datasets import load_files
from sklearn.cross_validation import train_test_split
from sklearn import metrics

dataset = load_files('txt_sentoken', shuffle=False)
print("n_samples: %d" % len(dataset.data))
```

```
n_samples: 2000
```

Split data into training (75%) and testing (25%) sets

```
docs_train, docs_test, y_train, y_test = train_test_split(
    dataset.data, dataset.target, test_size=0.25, random_state=None)
```

Build pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

# Vectorizer / classifier pipeline that filters out tokens that are too rare or too frequent
pipeline = Pipeline([
    ('vect', TfidfVectorizer(min_df=3, max_df=0.95)),
    ('clf', LinearSVC(C=1000)),
```

```
])
```

Grid search

```
# Find out whether unigrams or bigrams are more useful.
parameters = {
    'vect__ngram_range': [(1, 1), (1, 2)],
}
grid_search = GridSearchCV(pipeline, parameters, n_jobs=1)
# Fit pipeline on training set using grid search for the parameters
grid_search.fit(docs_train, y_train)

# Print cross-validated scores for each parameter set explored by the grid search
print(grid_search.grid_scores_)

# Predict outcome on testing set and store it in a variable named y_predicted
y_predicted = grid_search.predict(docs_test)

# Print classification report
print(metrics.classification_report(y_test, y_predicted,
                                    target_names=dataset.target_names))
```

```
[mean: 0.82200, std: 0.00748, params: {'vect__ngram_range': (1, 1)}, mean: 0.83533, std: 0.00984, params: {'vect__ngram_range': (1, 2)}]
             precision    recall  f1-score   support

        neg       0.90      0.88      0.89       247
        pos       0.88      0.91      0.89       253

avg / total       0.89      0.89      0.89       500
```

Print and plot confusion matrix

```
cm = metrics.confusion_matrix(y_test, y_predicted)
print(cm)
plt.matshow(cm)
plt.colorbar()
plt.title('Confusion matrix')
plt.ylabel('True')
plt.xlabel('Predicted')
plt.show()
```

```
[[217  30]
 [ 24 229]]
```