Project 4 - IMDb Prediction

By Yuheng Lin

Mount google drive to the colab

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989

Enter your authorization code:
··········
Mounted at /content/drive
```

1. Directly unpack the Gzip-compressed tarball archive directly in Python after download the data zip file

```
import tarfile

with tarfile.open('/content/drive/My Drive/aclImdb_v1.tar.gz', 'r:gz') as tar:
  tar.extractall()
```

2. Preprocessing the movie dataset into more convenient format

```
!pip install pyprind

import pyprind
import pandas as pd
import os



basepath = 'aclImdb'

labels = {'pos': 1, 'neg': 0}
pbar = pyprind.ProgBar(50000)
df = pd.DataFrame()

for s in ('test', 'train'):
  for l in ('pos', 'neg'):
    path = os.path.join(basepath, s, l)
    for file in os.listdir(path):
      with open(os.path.join(path, file),
                'r', encoding='utf-8') as infile:
        txt = infile.read()
        df = df.append([[txt, labels[l]]],
                       ignore_index=True)
        pbar.update()

df.columns = ['review', 'sentiment']
```

```
0% [##############################] 100% | ETA: 00:00:00
Total time elapsed: 00:01:26
```

3. Read sentences and reviews from the dataset above

```
import numpy as np

np.random.seed(0)
df = df.reindex(np.random.permutation(df.index))
df.to_csv('movie_data.csv', index=False, encoding='utf-8')
df = pd.read_csv('movie_data.csv', encoding='utf-8')
df.head(3)
```

	review	sentiment
0	Starring: James Belushi; Peter Dinklage; Alex ...	0
1	Fulci... Does this man brings one of the gorie...	1
2	As a low budget enterprise in which the filmma...	1

4. Cleaning our text data

```
import re

def preprocessor(text):
  text = re.sub('<[^>]*>', '', text)
  emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text)
  text = (re.sub('[\W]+', ' ', text.lower()) + ' '.join(emoticons).replace('-', ''))
  return text

df['review'].apply(preprocessor)
```

5. Processing documents into tokens

```
from nltk.stem.porter import PorterStemmer
import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

porter = PorterStemmer()
def tokenizer_porter(text):
  return [porter.stem(word) for word in text.split()]

def tokenizer(text):
  return text.split()

stop = stopwords.words('english')
[w for w in df['review'].apply(tokenizer_porter)[-10:] if w not in stop]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[['veri',
  'straight',
  '_',
  'not',
  'happi',
  'with',
  'the',
  'movie.<br',
  '/><br',
  '/>the',
  'main',
  'center',
  'of',
  'the',
  'movi',
  'is',
  'the',
  'stori',
  'where',
  'the',
  'ladi',
  'is',
  'the',
  'mother',
  'of',
  'all',
  'the',
  'snack',
  'and',
  'all',
  'the',
  'things.<br',
  '/><br',
  '/>if',
  'they',
  'can',
  'more',
  'explain',
  'that',
  'how',
  'thi',
  'is',
  'happen',
  'and',
  'all',
```

By looking at the head of our break down sentences. They seem good.

Next: **6. Transforming words into feature vectors**

```
        it ,
```

```
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer()
bag = count.fit_transform(df['review'])
```

```
print(count.vocabulary_)
```

```
{'starring': 85514, 'james': 47250, 'belushi': 9389, 'peter': 67587, 'dinklage': 25259, 'a
```

7. Access word relevancy

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf = TfidfTransformer(use_idf=True,
                         norm='l2',
                         smooth_idf=True)

np.set_printoptions(precision=2)
print(tfidf.fit_transform(count.fit_transform(df['review'])).toarray())
```

8. Build the logistic model

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer

X_train = sample.loc[:25000, 'review'].values
y_train = sample.loc[:25000, 'sentiment'].values
X_test = sample.loc[25000:, 'review'].values
y_test = sample.loc[25000:, 'sentiment'].values

tfidf = TfidfVectorizer(strip_accents=None, lowercase=False, preprocessor=None)
param_grid = [{'vect__ngram_range': [(1,1)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer,
                                   tokenizer_porter],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': [1.0, 10.0, 100.0]},
              {'vect__ngram_range': [(1,1)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer,
                                   tokenizer_porter],
               'vect__use_idf':[False],
               'vect__norm':[None],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': [1.0, 10.0, 100.0]}
             ]
lr_tfidf = Pipeline([('vect', tfidf), ('clf', LogisticRegression(random_state=0))])
gs_lr_tfidf = GridSearchCV(lr_tfidf, param_grid, scoring='accuracy', cv=5, verbose=1, n_jobs=1)
gs_lr_tfidf.fit(X_train, y_train)

print('CV Accuracy: %.3f' % gs_lr_tfidf.best_score_)
clf = gs_lr_tfidf.best_estimator_
print('Test Accuracy: %.3f' % clf.score(X_test, y_test))
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/feature_extraction/text.py:301: UserWarning
  'stop_words.' % sorted(inconsistent))
[Parallel(n_jobs=1)]: Done 240 out of 240 | elapsed: 10.1min finished
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:841: Deprecation
  DeprecationWarning)
CV Accuracy: 0.778
Test Accuracy: 0.778
```