Project 5

April 25, 2020

1 Unzip dataset and read into Dataframe

[1]:

```
import pyprind
import pandas as pd
import os
# change the `basepath` to the directory of the
# unzipped movie dataset
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: 00:03:04
```

2 Put into csv for data manipulation and shuffle

[2]:

```
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')
```

[3]:

```
df = pd.read_csv('movie_data.csv', encoding='utf-8')
df.head(3)
```

[3]:

```
                                              review  sentiment
0  My family and I normally do not watch local mo…          1
1  Believe it or not, this was at one time the wo…          0
2  After some internet surfing, I found the "Home…          0
```

[4]:

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
count = CountVectorizer()
docs = np.array([
'The sun is shining',
'The weather is sweet',
'The sun is shining and the weather is sweet'])
bag = count.fit_transform(docs)
```

[5]:

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

```
{'the': 5, 'sun': 3, 'is': 1, 'shining': 2, 'weather': 6, 'sweet': 4, 'and': 0}
```

3 Feature vectors that are mapped

[6]:

```
print(bag.toarray())
```

```
[[0 1 1 1 0 1 0]
 [0 1 0 0 1 1 1]
 [1 2 1 1 1 2 1]]
```

4 Transformation to tf-idfs

[7]:

```
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(docs)).toarray())
```

```
[[0.   0.43 0.56 0.56 0.   0.43 0.  ]
 [0.   0.43 0.   0.   0.56 0.43 0.56]
 [0.4  0.48 0.31 0.31 0.31 0.48 0.31]]
```

5 Cleanup our data from unwanted characters

[8]:

```
#df.loc[0, 'review'][-50:]
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
```

[9]:
```
preprocessor(df.loc[0, 'review'][-50:])
```

[9]:
```
'to star cinema way to go jericho and claudine '
```

[10]:
```
preprocessor("</a>This :) is :( a test :-)!")
df['review'] = df['review'].apply(preprocessor)
```

6 Splitting text into individual elements

[11]:
```
def tokenizer(text):
    return text.split()
tokenizer('runners like running and thus they run')
```

[11]:
```
['runners', 'like', 'running', 'and', 'thus', 'they', 'run']
```

7 Reduce words to root form using Porter stemming algorithm

[12]:
```
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()
def tokenizer_porter(text):
    return [porter.stem(word) for word in text.split()]
tokenizer_porter('runners like running and thus they run')
```

[12]:
```
['runner', 'like', 'run', 'and', 'thu', 'they', 'run']
```

8 Remove stopwords so we avoid commonality

[13]:
```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')
[w for w in tokenizer_porter('a runner likes running and runs a lot')[-10:] if␣
 ↪w not in stop]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/nickfrasco/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

```
[13]: ['runner', 'like', 'run', 'run', 'lot']
```

9 Make our test and train sets

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

10 Make and train model

```
[15]: from sklearn.model_selection import GridSearchCV
      from sklearn.pipeline import Pipeline
      from sklearn.linear_model import LogisticRegression
      from sklearn.feature_extraction.text import TfidfVectorizer
      tfidf = TfidfVectorizer(strip_accents=None, lowercase=False, preprocessor=None)
      param_grid = [{'vect__ngram_range': [(1,1)],
                         'vect__stop_words': [stop, None],
                         'vect__tokenizer': [str.split],
                         'clf__penalty': ['l1', 'l2'],
                         'clf__C': [1.0, 10.0, 100.0]},
                        {'vect__ngram_range': [(1,1)],
                         'vect__stop_words': [stop, None],
                         'vect__tokenizer': [str.split],
                         '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)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   41.6s
```

```
/opt/anaconda3/lib/python3.7/site-
packages/joblib/externals/loky/process_executor.py:706: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:  3.4min finished
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

```
[15]: GridSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=Pipeline(memory=None,
                                      steps=[('vect',
                                              TfidfVectorizer(analyzer='word',
                                                              binary=False,
                                                              decode_error='strict',
                                                              dtype=<class
      'numpy.float64'>,
                                                              encoding='utf-8',
                                                              input='content',
                                                              lowercase=False,
                                                              max_df=1.0,
                                                              max_features=None,
                                                              min_df=1,
                                                              ngram_range=(1, 1),
                                                              norm='l2',
                                                              preprocessor=None,
                                                              smooth_idf=True,
                                                              stop_word…
                                                      'our', 'ours', 'ourselves',
                                                      'you', "you're", "you've",
                                                      "you'll", "you'd", 'your',
                                                      'yours', 'yourself',
                                                      'yourselves', 'he', 'him',
                                                      'his', 'himself', 'she',
                                                      "she's", 'her', 'hers',
                                                      'herself', 'it', "it's", 'its',
                                                      'itself', …],
                                                     None],
                                'vect__tokenizer': [<method 'split' of 'str'
      objects>],
                                'vect__use_idf': [False]}],
                   pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                   scoring='accuracy', verbose=1)
```

11 Find best hyperparamters for model

```
[16]: print('Best parameter set: %s ' % gs_lr_tfidf.best_params_)
```

```
Best parameter set: {'clf__C': 10.0, 'clf__penalty': 'l2', 'vect__ngram_range':
(1, 1), 'vect__stop_words': None, 'vect__tokenizer': <method 'split' of 'str'
objects>}
```

12 Test accuracy

```
[17]: 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))
```

```
CV Accuracy: 0.893
Test Accuracy: 0.900
```

13 Conclusion

As we can see, the grid search comes out to be pretty accurate in terms of recognition. Our model can predict whether a movie review is positive or negative with about 90 perfect accuracy. I had to change a few of the hyperparameters so it would run efficiently enough to get through the whole program. If I hadn't, It would have taken an eternity.