資料探勘研究與實務 Data Mining

0853426 陳紹雲

一、 資料前處理-讀入資料

利用迴圈以行依序讀入串列中，分別對 Train, Test Data 做處理，最後轉成 DataFrame

In [1]:
```python
import numpy as np
import pandas as pd

res = [];count = 0
with open('data/training_label.txt', 'r' , encoding='utf-8') as fn:
    for line in fn:
        line=line.strip('\n')
        if line != "":
            line_list = str(line).split("+++$+++")
            line_list[1] = line_list[1].strip()
            res.append(line_list)
            count += 1
            if(count>=10000):
                break
```

二、 資料前處理- Data Cleaning

主要對資料做三種保留方式，

1. 用 bs4 套件去除 html tags。
2. 用 re 抓 emoji，保留在 text
3. 最後再把所有單詞轉成小寫把 emoji 附加在最後面

```python
def preprocessor(text):
    # remove HTML tags
    text = BeautifulSoup(text, 'html.parser').get_text()
    |
    # regex for matching emoticons, keep emoticons, ex: :), :-P, :-D
    r = '(?::|;|=|X)(?:-)?(?:\)|\(|D|P)'
    emoticons = re.findall(r, text)
    text = re.sub(r, '', text)

    # convert to lowercase and append all emoticons behind (with space in between)
    # replace('-','') removes nose of emoticons
    text = re.sub('[\W]+', ' ', text.lower()) + ' ' + ' '.join(emoticons).replace('-','')
    return text
```

三、 資料前處理- Word Stemming

製作一個可以拆解句子成 token 的函式，且跟 StopWord 比對

```
def tokenizer_stem(text):
    porter = PorterStemmer()
    return [porter.stem(word) for word in re.split('\s+', text.strip()) \
            if word not in stop and re.match('[a-zA-Z]+', text)] # re.match只比對字串頭
```

四、 CountVectorizer

利用字頻建立 Token of Bags.

```
doc = train['review']
count = CountVectorizer(preprocessor=preprocessor, tokenizer=tokenizer_stem)
doc_bag = count.fit_transform(doc).toarray() # every data features
```

五、 TfidfVectorizer

建立 idf scores vector ，其中考慮 term-frequency (TF) as BoW 也考慮到 the document-frequency (DF)

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(preprocessor=preprocessor, tokenizer=tokenizer_stem)
tfidf.fit(doc)
```

六、 Feature Hashing

將每個詞彙散列到具有固定數量的 bucket 的 table 中來減少維度詞彙空間。(降低內存所需，但是犧牲掉 IDF 加權)

```
In [9]:
from sklearn.feature_extraction.text import HashingVectorizer

hashvec = HashingVectorizer(n_features=2**10,
                            preprocessor=preprocessor,
                            tokenizer=tokenizer_stem)

doc_hash = hashvec.transform(doc)
print(doc_hash.shape)
```

```
(10000, 1024)
```

七、利用 Pipeline 建模

```python
from sklearn.pipeline import Pipeline
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier

names = ['AdaBoostClassifier+preprocess+hash', 'AdaBoostClassifier+preprocess', 'XGBClassifier+preprocess+hash', 'XGBClassif
pipe1 = Pipeline([('vect', HashingVectorizer(n_features=2**10,
                                             preprocessor=preprocessor,
                                             tokenizer=tokenizer_stem)),
                  ('clf', AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
        learning_rate=1.0, n_estimators=50, random_state=0))])

pipe2 = Pipeline([('vect', TfidfVectorizer(preprocessor=preprocessor,
                                           tokenizer=tokenizer_stem)),
                  ('clf', AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
        learning_rate=1.0, n_estimators=50, random_state=0))])

pipe3 = Pipeline([('vect', HashingVectorizer(n_features=2**10,
                                             preprocessor=preprocessor,
                                             tokenizer=tokenizer_stem)),
                  ('clf', XGBClassifier(n_estimators=50, random_state=0))])

pipe4 = Pipeline([('vect', TfidfVectorizer(preprocessor=preprocessor,
                                           tokenizer=tokenizer_stem)),
                  ('clf', XGBClassifier(n_estimators=50, random_state=0))])
```

七、 交叉驗證(10-fold)

```python
print('[auc (10-fold cv)]')
for name, clf in zip(names, [pipe1, pipe2, pipe3, pipe4]):
    scores = cross_val_score(estimator=clf, X=train['review'], y=train['sentiment'], \
                             cv=10, scoring='roc_auc')
    print('%s: %.3f (+/-%.3f)' % (name, scores.mean(), scores.std()))
```

```
[auc (10-fold cv)]
AdaBoostClassifier+preprocess+hash: 0.710 (+/-0.014)
AdaBoostClassifier+preprocess: 0.724 (+/-0.014)
XGBClassifier+preprocess+hash: 0.704 (+/-0.014)
XGBClassifier+preprocess: 0.710 (+/-0.018)
```

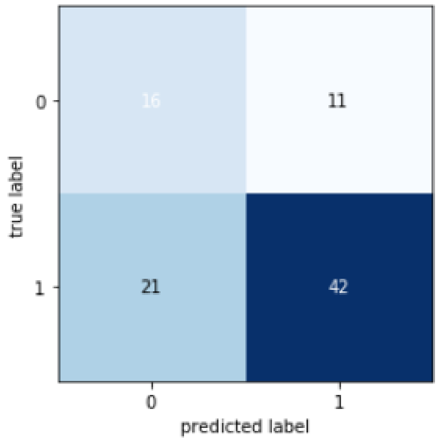
這邊發現 Adaboost 加上有做過預處理的效果最好。

八、 結果(抓兩個模型+預處理的做比較)

AdaBoostClassifier+preprocess

```
pipe2.fit(X=train['review'], y=train['sentiment'])
y_pred = pipe2.predict(test['review']).astype('int64')
y_test = test['sentiment'].values.astype('int64')
shen_bing_how_shy(y_pred, y_test)
```

```
Accuracy : 0.644
F1_Score : 0.724
Precision : 0.432
Recall : 0.593
```



XGBClassifier+preprocess

```
pipe4.fit(X=train['review'], y=train['sentiment'])
y_pred = pipe3.predict(test['review']).astype('int64')
y_test = test['sentiment'].values.astype('int64')
shen_bing_how_shy(y_pred, y_test)
```

```
[XGBClassifier+preprocess]
Accuracy : 0.622
F1_Score : 0.691
Precision : 0.486
Recall : 0.545
```