NLP文本分类 新闻分类

一娃的成长 一娃的成长 4月16日

记一次中文新闻文本分类的实验记录，本实验主要是对以下几个网页内容的整合及结果复现：
https://www.sohu.com/a/165903757_176628
https://baijiahao.baidu.com/s?id=1641172818761365604&wfr=spider&for=pc

0. 实验环境
代码环境：Python 3.6
分词工具：jieba
框架工具：sklearn gensim

1. 数据准备
1.1 训练及测试数据
采用开源的清华新闻数据集THUCNews，数据下载链接
http://thuctc.thunlp.org/message
本次实验只处理'财经', '股票', '科技', '社会', '游戏'五个类别的部分数据，其中每个类别前1000条作为训练数据，后200条作为测试数据。

1.2 停用词文件
https://github.com/foowaa/Chinese_from_dongxiexidian

1.3 数据提取与清洗
prepare_data.py

```python
# encoding: utf-8
import os
import jieba
from multiprocessing import Process, Queue
import multiprocessing

def strQ2B(ustring):
    """全角转半角"""
    rstring = ""
    for uchar in ustring:
        inside_code=ord(uchar)
        if inside_code == 12288:                              #全角空格直接转
            inside_code = 32
        elif (inside_code >= 65281 and inside_code <= 65374): #全角字符（除空
            inside_code -= 65248
        rstring += chr(inside_code)
    return rstring
# datadir include category subdir
# return tuple list [(filepath, category)]
def get_data_tuples(datadir,categories,operation,cnt_per_type):
    tuples = []
    for _,dirs,_ in os.walk(datadir):
        for d in dirs:
            if d in categories:
                files = os.listdir(datadir+'/'+d)
                if operation == 'train':
                    for f in files[:cnt_per_type]:
                        f = datadir+'/'+d+'/'+f
                        tuples.append((f,d))
                if operation == 'test':
                    for f in files[-cnt_per_type-1:-1]:
                        f = datadir + '/' + d + '/'+f
                        tuples.append((f,d))

    return tuples

def get_stop_words(fpath):
    ignore_words= []
    with open(fpath,'r',encoding='utf-8') as fstop:
        for l in fstop.readlines():
            l = l.replace('\n','')
            ignore_words.append(l)

    return ignore_words

def cleanup_sentence(q,ftuples,stop_words):
    for f,k in ftuples:
        with open(f,'r',encoding='utf-8') as fr:
            text = fr.read().lower()
            text = text.replace("\t"," ").replace("\n"," ")
            seg_text = jieba.cut(text)
            segout = " ".join(seg_text)
            words = segout.split()
            outline = []  # " ".join(outline.split())
            for i in words:
                if i not in stop_words:
                    outline.append(i)
            outline = " ".join(outline) + "\t__label__" + k + "\n"
            q.put(outline)
            # print("segment file: %s" % f)
            # print(outline)
            # break

if __name__ == '__main__':
    stop_words = get_stop_words('./stopwords.dat')

    train_categories = ['财经', '股票', '科技', '社会', '游戏']
    test_categories = train_categories

    train_tuples = get_data_tuples('../data/THUCNews', train_categories, 'tr
    test_tuples = get_data_tuples('../data/THUCNews', test_categories, 'test

    q = Queue()
    procs = []
    proc_cnt = multiprocessing.cpu_count()

    if len(train_tuples) <= proc_cnt:
        proc_cnt = len(train_tuples)
    task_piece = len(train_tuples) / proc_cnt
    task_reserve = len(train_tuples) % proc_cnt

    e = 0
    for i in range(multiprocessing.cpu_count()):
        b = e
        e += task_piece
        if i < task_reserve:
            e += 1
        p = Process(target=cleanup_sentence, args=(q,train_tuples[int(b):int
        p.start()
        procs.append(p)

    with open("news_train.txt", "w", encoding='utf-8') as ftrain:
        i = 0
        while i < len(train_tuples):
            line = q.get()
            ftrain.write(line)
            #ftrain.flush()
            i += 1

    for t in procs:
        t.join()
    procs.clear()

    if len(test_tuples) <= proc_cnt:
        proc_cnt = len(test_tuples)
    task_piece = len(test_tuples) / proc_cnt
    task_reserve = len(test_tuples) % proc_cnt

    e = 0
    for i in range(multiprocessing.cpu_count()):
        b = e
        e += task_piece
        if i < task_reserve:
            e += 1
        p = Process(target=cleanup_sentence, args=(q,test_tuples[int(b):int
        p.start()
        procs.append(p)

    with open("news_test.txt", "w", encoding='utf-8') as f:
        i = 0
        while i < len(test_tuples):
            line = q.get()
            f.write(line)
            i += 1

    for t in procs:
        t.join()
    procs.clear()
```

2. 训练&测试

2.1 训练过程

此过程以两种向量空间模型Doc2Vec和Tf-Idf为例，分别使用 `LogisticRegression` `RandomForestClassifier` `XGBoost` 三种机器学习算法进行对比测试。

`transformers.py`

```python
# encoding: utf-8

from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from sklearn.base import BaseEstimator
from sklearn import utils as skl_utils
from tqdm import tqdm
import multiprocessing
import numpy as np

class Doc2VecTransformer(BaseEstimator):
    def __init__(self, vector_size=100, learning_rate=0.02, epochs=5):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self._model = None
        self.vector_size = vector_size
        self.workers = multiprocessing.cpu_count() - 1

    def fit(self, df_x, df_y=None):
        tagged_x = [TaggedDocument(row.split(), [index]) for index, row in er
        model = Doc2Vec(documents=tagged_x, vector_size=self.vector_size, wor
        for epoch in range(self.epochs):
            model.train(skl_utils.shuffle([x for x in tqdm(tagged_x)]), tota
            model.alpha -= self.learning_rate
            model.min_alpha = model.alpha
            self._model = model

        return self

    def transform(self, df_x):
        return np.asmatrix(np.array([self._model.infer_vector(x.split()) for


# tf-idf
from sklearn.feature_extraction.text import TfidfVectorizer

class Text2TfIdfTransformer(BaseEstimator):
    def __init__(self):
        self._model = TfidfVectorizer()
        pass

    def fit(self, df_x, df_y=None):
        # df_x = df_x.apply(lambda x : clean_text(x))
        self._model.fit(df_x)

        return self

    def transform(self, df_x):
        return self._model.transform(df_x)


from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score


if __name__ == "__main__":
    in_x = []
    in_y = []
    with open('news_train.txt','r',encoding='utf-8') as f:
        for l in f.readlines():
            l = l.replace('\n','')
            in_x.append(l.split('\t')[0])
            in_y.append(l.split('\t')[1].replace('__label__',''))

    # doc2vec_trf = Doc2VecTransformer()
    # doc2vec_features = doc2vec_trf.fit(in_x).transform(in_x)
    # print(doc2vec_features)
    #
    # pl_log_reg = Pipeline(steps=[('doc2vec',Doc2VecTransformer()),
    #     ('log_reg', LogisticRegression(multi_class='auto', solver='liblinea
    #
    # scores = cross_val_score(pl_log_reg, in_x, in_y, cv=5,scoring='accuracy
    # print('Accuracy for Logistic Regression: ', scores.mean())

    tfidf_transformer = Text2TfIdfTransformer()
    tfidf_vectors = tfidf_transformer.fit(in_x).transform(in_x)
    print("tf-idf vector shape: ", tfidf_vectors.shape)

    pl_log_reg_tf_idf = Pipeline(steps=[('tfidf', Text2TfIdfTransformer()),
        ('log_reg', LogisticRegression(multi_class='auto', solver='liblinear'
    scores = cross_val_score(pl_log_reg_tf_idf, in_x, in_y, cv=5, scoring='ac
    print('Accuracy for Tf-Idf & Logistic Regression: ', scores.mean())
```

classify_cmp.py

```
# encoding: utf-8

from transformers import *

from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import xgboost as xgb
import random


class EmptyTransformer():
    def __init__(self):
        pass

    def fit(self, df_x, df_y=None):
        # print("do fit ...")
        return self

    def transform(self, in_x):
        # print("do transform ...")
        return in_x
```

```


if __name__ == "__main__":
    in_x = []
    in_y = []
    with open('news_train.txt','r',encoding='utf-8') as f:
        for l in f.readlines():
            l = l.replace('\n','')
            in_x.append(l.split('\t')[0])
            in_y.append(l.split('\t')[1].replace('__label__',''))

    randnum = random.randint(0, 100)
    random.seed(randnum)
    random.shuffle(in_x)
    random.seed(randnum)
    random.shuffle(in_y)

    doc2vec_trf = Doc2VecTransformer()
    doc2vec_features = doc2vec_trf.fit(in_x).transform(in_x)
    # print(doc2vec_features)
    print("doc2vec vector shape: ", doc2vec_features.shape)

    # pl_log_reg = Pipeline(steps=[('doc2vec',Doc2VecTransformer()),
    #     ('log_reg', LogisticRegression(multi_class='auto', solver='liblin
    #
    # scores = cross_val_score(pl_log_reg, in_x, in_y, cv=5,scoring='accura
    # print('Accuracy for Logistic Regression: ', scores.mean())

    classes = list(set(in_y))
    label_y = []
    output_empty = [0] * len(classes)
    for y in in_y:
        output_row= list(output_empty)
        output_row[classes.index(y)]= 1
        label_y.append(output_row)

    test_x = []
    test_y = []
    with open('news_test.txt','r',encoding='utf-8') as f:
        for l in f.readlines():
```

```
        l = l.replace('\n','')
        test_x.append(l.split('\t')[0])
        test_y.append(l.split('\t')[1].replace('__label__',''))

randnum = random.randint(0, 100)
random.seed(randnum)
random.shuffle(test_x)
random.seed(randnum)
random.shuffle(test_y)

test_doc2vec_features = doc2vec_trf.transform(test_x)
print("test_doc2vec vector shape: ", test_doc2vec_features.shape)

test_label_y = []
for y in test_y:
    output_row= list(output_empty)
    output_row[classes.index(y)]= 1
    test_label_y.append(output_row)

pl_log_reg = Pipeline(steps=[('doc2vec',EmptyTransformer()),
    ('log_reg', LogisticRegression(multi_class='auto', solver='liblinear
scores = cross_val_score(pl_log_reg, doc2vec_features, in_y, cv=5,scorin
print('Accuracy for Logistic Regression Classifier : ', scores.mean())

pl_random_forest = Pipeline(steps=[('doc2vec',EmptyTransformer()),
    ('random_forest',RandomForestClassifier())])
scores = cross_val_score(pl_random_forest, doc2vec_features, in_y, cv=5,
print('Accuracy for RandomForest Classifier : ', scores.mean())

pl_xgb = Pipeline(steps=[('doc2vec',EmptyTransformer()),
    ('xgb_boost', xgb.XGBClassifier(objective='binary:logistic'))])
scores = cross_val_score(pl_xgb, doc2vec_features, in_y, cv=5)
print('Accuracy for XGBoost Classifier : ', scores.mean())

tfidf_transformer = Text2TfIdfTransformer()
tfidf_vectors = tfidf_transformer.fit(in_x).transform(in_x)
print("tf-idf vector shape: ", tfidf_vectors.shape)

test_tfidf_features = tfidf_transformer.transform(test_x)
print("test_tf-idf vector shape: ", test_tfidf_features.shape)
```

```

    pl_log_reg_tf_idf = Pipeline(steps=[('tfidf', EmptyTransformer()),
        ('log_reg', LogisticRegression(multi_class='auto', solver='liblinear
    scores = cross_val_score(pl_log_reg_tf_idf, tfidf_vectors, in_y, cv=5,
    print('Accuracy for Tf-Idf & Logistic Regression: ', scores.mean())

    pl_random_forest_tf_idf = Pipeline(steps=[('tfidf', EmptyTransformer())
                                ('random_forest', RandomForestClassifier())
    scores = cross_val_score(pl_random_forest_tf_idf, tfidf_vectors, in_y,
    print('Accuracy for Tf-Idf & RandomForest : ', scores.mean())

    pl_xgb_tf_idf = Pipeline(steps=[('tfidf', EmptyTransformer()),
                            ('xgboost', xgb.XGBClassifier(objective='binary
    scores = cross_val_score(pl_xgb_tf_idf, tfidf_vectors, in_y, cv=5)
    print('Accuracy for Tf-Idf & XGBoost Classifier : ', scores.mean())
```

2.2 测试结果

```
doc2vec vector shape:  (5000, 100)
test_doc2vec vector shape:  (1000, 100)
Accuracy for Logistic Regression Classifier :  0.6442
Accuracy for RandomForest Classifier :  0.6432
Accuracy for XGBoost Classifier :  0.6522
tf-idf vector shape:  (5000, 104198)
test_tf-idf vector shape:  (1000, 104198)
Accuracy for Tf-Idf & Logistic Regression:  0.9064
Accuracy for Tf-Idf & RandomForest :  0.8942
Accuracy for Tf-Idf & XGBoost Classifier :  0.9086
```

3. 结果分析

实验结束，引用原文的结果分析：

尽管在自然语言处理中，“DocVec”模型比“Tf-Idf”模型更高级，但我们的案例证明，后者效果更佳。我们分别使用了基于线性、袋状以及推进型的分类器。

原因可以这么理解。在我们的数据集中，每一个“文本”领域包含了一些决定其类别的高频单词/标记。因此，应用一个对语境/上下文敏感的模型会使问题更为复杂、（或者）混淆信息。某些文本类别包含一些高频出现的标记，这些标记提供了大量数值以定义“Tf-Idf”模型。

同时，“文本”是细分领域的。比如，“布莱尔 (blair) ”一词更可能出现在“政治”类别，而非“运动”类别。因此，像这样的词对“Tf-Idf”模型起了作用。

而且，“Doc2Vec”模型更适合应用于语法正确的文本中。大量案例和数据科学家的实验证明，虽然“Tf-Idf”模型次于”DocVec”模型，但前者对于细分领域的文本分类更为有效。

实验代码GitHub地址：
https://github.com/cddypang/nlp-learning