به نام خدا

گزارش تمرین سوم درس NLP

دانشجو: محمد قربانی – ۹۷۱۳۱۰۹۹

بهار ۹۷-۹۸

لینک ها:

برای قسمت Pos تگینگ یک مدل با استفاده از شبکه های عصبی ایجاد کردیم که نام آن NNModel.joblib است و در لینک زیر قرار دارد:

https://www.dropbox.com/s/cp5w19fqbndgm7q/NNModel.joblib?dl=0

برای قسمت Ner نیز یک مدل با استفاده از Stanford NER ایجاد کردیم که نام آن trained_model.ser می باشد و در لینک زیر قرار دارد:

https://www.dropbox.com/s/cuv9ohh1nvqtwtb/trained_model.ser.gz?dl=0

توجه: لطفا دقت کنید که این دو فایل باید در فولدر روت پروژه قرار داشته باشند.

گزارش از ابزارهای استفاده شده در این دو بخش:

بخش اول:

در این بخش همچون سایر بخش‌ها از ابزار nltk برای توکنایز کردن یک جمله استفاده کردیم.

برای آموزش دسته‌بند خود، نیز از Neural Netwokها استفاده کردیم که نتایج بهتری نسبت به سایر روش‌ها تولید می‌کرد.

برای انجام ارزیابی‌ها نیز از کتابخانه‌های Sklearn استفاده شد.

برای ایجاد جدول‌ها نیز از کتابخانه pandas استفاده شد.

بخش دوم:

در این بخش همچون بخش قبل از ابزار nltk برای توکنایز کردن یک جمله استفاده کردیم.

برای ایجاد مدل tagger نیز از ابزار Stanford NER استفاده شد. برای استفاده از این ابزار مجبور به نصب جاوا بودیم چرا که این ابزار با استفاده از زبان جاوا نوشته شده است. سپس با استفاده از فایل آموزش در اختیار گذاشته شده، یک مدل با این ابزار ایجاد کردیم. سپس در برنامه پایتون خود از مدل ایجاد شده استفاده کردیم.

برای انجام ارزیابی‌ها نیز از کتابخانه‌های Sklearn استفاده شد.

برای ایجاد جدول‌ها نیز از کتابخانه pandas استفاده شد.

بخش اول تمرین

افزودن کتابخانه های مورد نیاز

In [1]:

```
import nltk, re, collections
from nltk import word_tokenize

from sklearn.feature_extraction import DictVectorizer
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPClassifier
from joblib import dump, load
import pandas as pd
from sklearn.metrics import confusion_matrix
```

در اینجا فایل آموزش و آزمون را میخوانیم و تبدیل به جملات می کنیم

In [2]:

```
train_docs = [line.rstrip('\n') for line in open('Data/POStrutf.txt', encoding="utf8")]

start = 0
end = 0
sentences_train = []
sentence_train = []
for i, doc in enumerate(train_docs[:]):
    word = re.split(r'\t+', doc)
    sentence_train.append(tuple([word[0], word[1]]))
    if word[0] == "#":
        start = i
    elif word[0] ==".":
        if start > end:
            nothing_to_do = 0
        else:
            start = end
        end = i
        sentences_train.append(sentence_train[start+1:end+1])
##################
test_docs = [line.rstrip('\n') for line in open('Data/POSteutf.txt', encoding="utf8")]

start = 0
end = 0
sentences_test = []
sentence_test = []
true_labels = []
for i, doc in enumerate(test_docs):
    word = re.split(r'\t+', doc)
    true_labels.append(word[1])
    sentence_test.append(tuple([word[0], word[1]]))
    if word[0] == ".":
        sentences_test.append(sentence_test[start:i+1])
        start = i + 1
```

تابع استخراج ویژگی را به صورت زیر تعریف میکنیم

In [3]:

```
def features(sentence, index):
    return {
        'word': sentence[index],
        'is_first': index == 0,
        'is_last': index == len(sentence) - 1,
        'prev_word': '' if index == 0 else sentence[index - 1],
        'next_word': '' if index == len(sentence) - 1 else sentence[index + 1],
        'has_hyphen': '-' in sentence[index],
        'is_numeric': sentence[index].isdigit(),
    }
```

دو تابع که از آن استفاده ابزاری خواهیم کرد

تابع اول برای جدا کردن تگ ها از کلمات یک جمله است

تابع دوم برای ایجاد دیتاستی از فیچر، تگ استفاده میشود

In [4]:

```
def untag(tagged_sentence):
    return [w for w, t in tagged_sentence]

def transform_to_dataset(tagged_sentences):
    X, y = [], []

    for tagged in tagged_sentences:
        for index in range(len(tagged)):
            X.append(features(untag(tagged), index))
            y.append(tagged[index][1])

    return X, y
```

برای آموزش مدلمان، دیتاست مورد نظر خودمان را ایجاد می کنیم

In [5]:

```
X, y = transform_to_dataset(sentences_train)
```

در این قسمت مدل خود را ایجاد می کنیم، این کار قبلا در سیستم های قوی تر انجام شده است و به صورت فایل ذخیره شده است

In [6]:

```
# Load from disk
clf = load('NNModel.joblib')

if clf is None:
    print("clf is none")
    train_model()
else:
    print("Model is loaded")

# We’re now ready to train the classifier. We use Neural Network classifier
def train_model():
    clf = Pipeline([
        ('vectorizer', DictVectorizer(sparse=False)),
        ('classifier', MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(100), random_st
ate=1))
    ])

    clf.fit(X[:53000], y[:53000])    # Use only the first 53K samples
    X_test, y_test = transform_to_dataset(sentences_test)

    print("Accuracy:", clf.score(X_test, y_test))
    # We reach to this accuracy
    # Accuracy: 0.8753189657822732

    # Save model to disk
    dump(clf, 'NNModel.joblib')
```

```
Model is loaded
```

همانطور که در بالا مشاهده می شود دقت مدل بدست آمده برابر با 87 درصد می باشد

در این تابع با استفاده از مدلی که ایجاد کردیم، جملات آزمون را تگ گذاری می کنیم

In [7]:

```
def pos_tag(sentence):
    tags = clf.predict([features(sentence, index) for index in range(len(sentence))])
    return zip(sentence, tags)

pred_labels_counter = collections.defaultdict(lambda:0)
pred_labels = []
for sentence in sentences_test:
    x = pos_tag(untag(sentence))
    for word, tag in x:
        pred_labels.append(tag)
        pred_labels_counter[tag] += 1

all_true_labels = list(set(true_labels))
```

در این قسمت ماتریس سرگشتگی را ایجاد می کنیم

In [8]:

```
cm = confusion_matrix(true_labels, pred_labels, labels=all_true_labels)
np.set_printoptions(suppress=True)
p = np.zeros((len(all_true_labels), len(all_true_labels)))

rounding_parameter = 4

for i in range(len(cm)):
    for j in range(len(cm)):
        p[i][j] = round((cm[i][j]/sum(cm[i])), rounding_parameter)

df = pd.DataFrame(p, columns=all_true_labels, index=all_true_labels)
df
```

Out[8]:

	PRO	MS	ADV	MQUA	OH	AR	DET	DELM	PS	P	...	PP	V	OHH	NP	SPEC	CON	MORP
PRO	0.8964	0.0	0.0057	0.0000	0.0	0.0	0.0278	0.0000	0.0	0.0000	...	0.0	0.0095	0.0	0.0	0.0006	0.0051	0.0006
MS	0.0000	0.0	0.0303	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0000
ADV	0.0059	0.0	0.5456	0.0000	0.0	0.0	0.0020	0.0000	0.0	0.0010	...	0.0	0.0118	0.0	0.0	0.0029	0.0275	0.0000
MQUA	0.0000	0.0	0.3333	0.0667	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0000
OH	0.0000	0.0	0.0000	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0000
AR	0.0367	0.0	0.0061	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0044	...	0.0	0.0306	0.0	0.0	0.0000	0.0847	0.0000
DET	0.0749	0.0	0.0078	0.0000	0.0	0.0	0.8998	0.0000	0.0	0.0000	...	0.0	0.0000	0.0	0.0	0.0010	0.0010	0.0000
DELM	0.0000	0.0	0.0020	0.0000	0.0	0.0	0.0000	0.9137	0.0	0.0000	...	0.0	0.0030	0.0	0.0	0.0000	0.0003	0.0000
PS	0.0000	0.0	0.0000	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0000
P	0.0001	0.0	0.0023	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.9744	...	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0000
N	0.0002	0.0	0.0030	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0050	...	0.0	0.0062	0.0	0.0	0.0010	0.0013	0.0000
PP	0.0000	0.0	0.0000	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0500	...	0.9	0.0000	0.0	0.0	0.0000	0.0000	0.0000
V	0.0000	0.0	0.0016	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.8829	0.0	0.0	0.0000	0.0000	0.0000
OHH	0.0000	0.0	0.0000	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0000
NP	0.0000	0.0	0.0000	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0000
SPEC	0.1122	0.0	0.0306	0.0000	0.0	0.0	0.0408	0.0000	0.0	0.0000	...	0.0	0.0000	0.0	0.0	0.3571	0.0000	0.0000
CON	0.0003	0.0	0.0269	0.0000	0.0	0.0	0.0003	0.0000	0.0	0.0018	...	0.0	0.0010	0.0	0.0	0.0000	0.9291	0.0000
MORP	0.0000	0.0	0.0000	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.0227	0.0	0.0	0.0000	0.0000	0.0682
IF	0.0000	0.0	0.0174	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.0087	0.0	0.0	0.0000	0.0261	0.0000
ADJ	0.0013	0.0	0.0085	0.0000	0.0	0.0	0.0007	0.0000	0.0	0.0001	...	0.0	0.0243	0.0	0.0	0.0000	0.0004	0.0004
QUA	0.0266	0.0	0.0102	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000	...	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0000

21 rows × 21 columns

در اینجا عناصر غیر قطری را به صورت نزولی مرتب کنیم و سپس 10 عنصری که در آنها بیشترین خطا را داریم استخراج می کنیم

In [9]:

```
non_diagonal_values = []
non_diagonal_labels = []
for i in range(len(cm)):
    for j in range(len(cm)):
        if j == i:
            continue
        else:
            non_diagonal_labels.append(tuple([str(all_true_labels[i])+","+str(all_true_labels[j]),p[
i][j]]))
            non_diagonal_values.append(p[i][j])

sorted_list_of_non_diagonal = np.flipud(np.argsort(non_diagonal_values))

for i in range(len(sorted_list_of_non_diagonal[:10])):
    print(non_diagonal_labels[sorted_list_of_non_diagonal[i]])
```

```
('OHH,N', 1.0)
('NP,N', 1.0)
('OH,N', 1.0)
('MS,N', 0.8788)
('AR,N', 0.8157)
('PS,N', 0.8)
('MORP,N', 0.5)
('SPEC,N', 0.4184)
('MORP,ADJ', 0.4091)
('ADJ,N', 0.3429)
```

همانطور که در بالا میشود بیشترین خطاها به دلیل در نظر گرفتن مقدار

N

به جای لیبل هایی مانند

OHH, NP, OH, MS

می باشد

در اینجا یک فایل ورودی را می گیرد و در خروجی دیگر لیبل ها را تولید می کند

In [11]:

```
in_docs = [line.rstrip('\n') for line in open('Data/in.txt', encoding="utf8")]

start = 0
end = 0
sentences_in = []
for i, doc in enumerate(in_docs):
    word = re.split(r'\t+', doc)
    sentence_test.append(tuple([word[0], word[1]]))
    if word[0] == ".":
        sentences_in.append(sentence_test[start:i+1])
        start = i + 1

resultFile = open('Data/out.txt', 'w', encoding="utf8")
for sentence in sentences_in:
    x = pos_tag(untag(sentence))
    for word, tag in x:
        resultFile.write(word+"\t"+tag+"\n")
resultFile.close()
```

بخش دوم تمرین

کتابخانه های مورد نظر خود را اضافه می کنیم

In [1]:

```python
import os
import nltk
from nltk.tag.stanford import StanfordNERTagger

import re
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import confusion_matrix
import pandas as pd
```

ما مدل را از قبل بوسیله جاوا آماده کردیم و فقط در اینجا از آن استفاده می کنیم

In [2]:

```python
os.environ['JAVAHOME'] = "C:/Program Files/Java/jre1.8.0_211/bin/java.exe"

jar = 'stanford-ner.jar'
model = 'trained_model.ser.gz'

ner_tagger = StanfordNERTagger(model, jar, encoding='utf8')
```

فایل تست را میخوانیم و با استفاده از مدل ایجاد شده، لیبل های آنها را پیش بینی می کنیم

In [3]:

```python
test_docs = [line.rstrip('\n') for line in open('Data/NERte.txt', encoding="utf8")]

true_labels = []
words = []
for i, doc in enumerate(test_docs):
    word = re.split(r'\t+', doc)
    if(word[0] != ""):
        words.append(word[0])
        true_labels.append(word[1])


predicts = ner_tagger.tag(words)
pred_labels = []
for element1, element2 in predicts:
    pred_labels.append(element2)
```

In [4]:

```python
entity_counter = 0
all_entities_start_end_labels = []
for i, label in enumerate(true_labels[:]):
    if label != 'O':
        label_start_end = []
        label_start_end.append(true_labels[i])
        entity_counter += 1
        label_start_end.append(i)
        current_label = label
        j = i
        while current_label == label:
            j += 1
            current_label = true_labels[j]
        label_start_end.append(j)
        i = j

        all_entities_start_end_labels.append(label_start_end)

entity_counter_pred = 0
for i, label in enumerate(pred_labels[:]):
    if label != 'O':
        entity_counter_pred += 1
        current_label = label
        j = i
        while current_label == label:
            j += 1
            current_label = pred_labels[j]
        i = j
```

دقت و ریکال را به صورت زیر بدست آوردیم

In [5]:

```python
true_match_counter = 0
for i, element in enumerate(all_entities_start_end_labels):
    start = element[1]
    end = element[2]
    match = True
    for j in range(start, end):
        if element[0] != pred_labels[j]:
            match = False
    if match == True:
        true_match_counter += 1

print("Our Precision is: "+str(true_match_counter/entity_counter_pred))
print("Our Recall is: " + str(true_match_counter/entity_counter))
```

```
Our Precision is: 0.672972972972973
Our Recall is: 0.21899736147757257
```

ماتریس سرگشتگی را به صورت زیر به دست آوردیم

In [6]:

```python
all_true_labels = list(set(true_labels))

cm = confusion_matrix(true_labels, pred_labels, labels=all_true_labels)

#Normalization
np.set_printoptions(suppress=True)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

df = pd.DataFrame(cm, columns=all_true_labels, index=all_true_labels)
df
```

Out[6]:

	musicartist	company	product	tvshow	geo-loc	sportsteam	facility	other	O	person	movie
musicartist	0.054795	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.917808	0.027397	0.000000
company	0.000000	0.22093	0.000000	0.000000	0.000000	0.000000	0.034884	0.000000	0.720930	0.023256	0.000000
product	0.000000	0.00000	0.018868	0.000000	0.000000	0.000000	0.000000	0.075472	0.830189	0.075472	0.000000
tvshow	0.000000	0.00000	0.000000	0.086957	0.000000	0.000000	0.000000	0.000000	0.869565	0.043478	0.000000
geo-loc	0.000000	0.00000	0.000000	0.000000	0.254902	0.000000	0.065359	0.000000	0.627451	0.052288	0.000000
sportsteam	0.000000	0.00000	0.000000	0.000000	0.052632	0.157895	0.000000	0.000000	0.789474	0.000000	0.000000
facility	0.000000	0.00000	0.000000	0.000000	0.009709	0.000000	0.310680	0.029126	0.631068	0.019417	0.000000
other	0.000000	0.00000	0.000000	0.000000	0.011111	0.000000	0.000000	0.122222	0.848148	0.018519	0.000000
O	0.000000	0.00000	0.000052	0.000000	0.000261	0.000052	0.000209	0.001200	0.997131	0.000991	0.000104
person	0.000000	0.00000	0.000000	0.000000	0.007519	0.000000	0.000000	0.015038	0.556391	0.421053	0.000000
movie	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.842105	0.000000	0.157895

همانطور که در زیر مشاهده می شود بیشترین خطا به دلیل استفاده از لیبل

O

به جای لیبل

musicartist, tvshow, other

می باشد

In [7]:

```python
non_diagonal_values = []
non_diagonal_labels = []
for i in range(len(cm)):
    for j in range(len(cm)):
        if j == i:
            continue
        else:
            non_diagonal_labels.append(tuple([str(all_true_labels[i])+","+str(all_true_labels[j]),cm[i][j]]))
            non_diagonal_values.append(cm[i][j])

sorted_list_of_non_diagonal = np.flipud(np.argsort(non_diagonal_values))

for i in range(len(sorted_list_of_non_diagonal[:10])):
    print(non_diagonal_labels[sorted_list_of_non_diagonal[i]])
```

```
('musicartist,O', 0.9178082191780822)
('tvshow,O', 0.8695652173913043)
('other,O', 0.8481481481481481)
('movie,O', 0.8421052631578947)
('product,O', 0.8301886792452831)
('sportsteam,O', 0.7894736842105263)
('company,O', 0.7209302325581395)
('facility,O', 0.6310679611650486)
('geo-loc,O', 0.6274509803921569)
('person,O', 0.556390977443609)
```

In [ ]: