SVM을 이용한
Text Classification

2015. 11. 26

대략적인 프로세스

- 사용자의 텍스트를 벡터 변환 합니다.(train_X)
- 텍스트에 해당하는 클래스(dialog-act)를 지정합니다.(train_y)
- svm 학습 시 테스트 데이터는 훈련에 사용하지 않습니다. train_X, train_y에는 모든 데이터가 들어있으므로 train과 test로 나눕니다.
- train 데이터를 이용해 svm으로 훈련을 하고 test 데이터로는 훈련모델에서 결과를 추정하여 추정값과 실제 값이 맞는지를 비교합니다.

코드설명

- data-train.json 파일을 읽어오면서 사용자의 transcript와 dialog-acts를 읽어옵니다.

```
''' json 파일에서 필요한 정보 읽어오기 '''
with open(DATA_FILE) as f:
    jData = json.load(f)

    for jSession in jData:
        jTurns = jSession.get('turns')
        for jTurn in jTurns:
            text = jTurn.get('user').get('transcript')

            dialog_acts = jTurn.get('user').get('dialog-acts')
```

- 아래 그림은 data-train.json의 일부입니다.
- 추정해야하는 정보는 user의 정보이므로 이전페이지에서 user의 정보만을 추출하였습니다.
    - (성능을 높이기 위해서는 system의 정보도 복합적으로 사용하여야 할 것 같습니다. 이 설명에서는 svm을 돌리기 위한 부분에 대해서만 참고하시면 되겠습니다.)


```
            "turn-index": "2",
            "system": {
                "transcript": "thanh binh is a great restaurant",
                "dialog-acts": [
                    {
                        "act": "offer",
                        "slots": [
                            [
                                "name",
                                "thanh binh"
                            ]
                        ]
                    }
                ]
            },
            "user": {
                "transcript": "restaurant in the west part of town",
                "dialog-acts": [
                    {
                        "act": "inform",
                        "slots": [
                            [
                                "area",
                                "west"
                            ]
                        ]
                    }
                ],
                "method-label": "byconstraints"
            }
        },
        {
```

- dialog_act의 경우 한 턴의 대화에 여러개가 나오는 경우가 있습니다. (예 : act:reqalts, act:inform)

- 아래는 이 두개의 act를 조합하여 act처리하기 위한 부분입니다.

```
for dialog_act in dialog_acts:
    if dialog_act['act'] not in acts:
        acts.append(dialog_act['act'])

acts.sort()
act_str = '-'.join(acts) #act가 여러개 일 경우 - 으로 연결하여 하나의 조합의로 처리하기 위함

if act_str not in acts_map:
    acts_map.append(act_str)
```

- 위의 예의 결과로 inform-reqalts 라는 새로운 act가 생성됩니다.

  - 이러한 처리는 제가 임의로 처리한 방법이므로 반드시 똑같이 하실 필요는 없습니다.

- text를 전에 제공한 코드를 이용해 vector로 변환합니다.

```
vector_items = ta.txt2vectors(text)# 이 부분에서 text를 vector로 변환함
```

- print(vector_items)를 이용해 내용을 출력해 보면, dictionary 형태로 {단어:벡터} 로 되어 있는것을 확인 하실 수 있습니다. (vector 값은 numpy array 형태의 50-dimension)

```
OrderedDict([('expensive', array([  7.53870000e-01,  -2.86600000e-01,  -2.52520000e-02,
        -2.79810000e-01,  -6.21660000e-01,   9.52390000e-02,
        -5.19510000e-01,  -6.70060000e-01,   1.65080000e-01,
         1.04500000e+00,  -3.27010000e-01,   1.24680000e-01,
         5.81170000e-01,   8.43400000e-02,   6.27920000e-01,
         5.83310000e-01,  -3.05660000e-02,   7.44650000e-01,
         2.22060000e-01,  -1.23170000e+00,   6.51070000e-01,
        -5.25920000e-01,  -9.38790000e-01,  -3.07140000e-02,
        -4.61660000e-01,  -1.26860000e+00,  -6.33270000e-01,
         3.93320000e-01,   1.16480000e+00,   2.25340000e-01,
         2.69530000e+00,   4.09810000e-01,   4.33840000e-01,
         2.43470000e-01,   8.69960000e-01,   6.31770000e-01,
         1.12450000e-01,   1.59090000e+00,  -8.00760000e-01,
        -9.28250000e-01,  -1.31320000e-01,   5.68340000e-01,
         1.05100000e+00,   9.05860000e-01,   2.24490000e-01,
         1.03230000e-01,  -4.01460000e-01,  -2.93120000e-05,
         1.21710000e-01,   6.46710000e-01])), ('restaurant', array([ 0.69607  ,  0.42922  , -
       -0.8808   , -1.391    , -0.063854 ,  0.20051  ,  0.067331 ,
       -0.69957  , -0.44104  ,  0.41343  ,  0.98698  ,  0.48247  ,
       -0.20524  ,  0.015262 ,  0.81055  , -0.16189  , -0.57772  ,
        1.4103   ,  0.30844  , -0.84441  ,  1.4506   , -0.78245  ,
```

- 단어의 수 만큼 벡터값이 생기는데 그것을 하나로 만들기 위해서 average 를 구합니다.

```
''' average vector 를 계산 '''
vectors_sum = 0
for word in vector_items:
    vectors_sum = np.add(vectors_sum, vector_items[word])

if len(vector_items) > 0:
    avg_vector = np.nan_to_num(vectors_sum / len(vector_items))
else:
    avg_vector = np.zeros(50, dtype=float)
```

- 벡터를 다 더한 후 단어의 수 대로 나눕니다.
- 단어가 없는 문장의 경우 전부 0인 벡터를 생성합니다.
    - txt2vectors의 입력 중 훈련 맵에 없는 단어의 경우, 처리하지 않고 있는 단어에 대해서만 결과를 반환합니다. 그렇기 때문에 간혹 벡터값이 없는 경우도 발생합니다.
    - 예를들어 ‘I love sogang' 이라는 문장을 text2vectors의 입력으로 넣었는데 sogang이란 단어가 훈련 맵에 없을 경우, ‘I love’ 에 대해서만 벡터값을 반환합니다.

- train_X에는 벡터가, train_y에는 dialog-act가 저장되게 됩니다. 즉 train_X[0]의 답은 train_y[0]이 됩니다.

```
train_X.append(avg_vector)
train_y.append(acts_map.index(act_str))
```

- train_test_split 함수를 이용하면 훈련과 테스트에 사용할 데이터을 원하는 비율로 random하게 나눠줍니다. test_size=0.1 이면 90%는 train, 10%는 test로 배정합니다.

- fit 함수는 svm 훈련을, predict은 값을 추정하는 함수입니다.

```
'''svm 을 이용한 classfication '''
X_train, X_test, y_train, y_test = train_test_split(train_X, train_y, test_size=0.1)
clf = svm.SVC()
clf.fit(X_train, y_train)

predict = clf.predict(X_test)
```

- precit의 결과와 y_test의 값을 비교하면 맞는지 틀린지를 체크 할 수 있고, 전체 개수에서 맞은 비율을 구하면 성능을 표시할 수 있습니다.

```
total = len(X_test)
correct = 0
for i in range(len(X_test)):
    if predict[i] == y_test[i]:
        correct += 1

print('TEST : {0} / {1}, {2:.2f}%'.format(correct, total, correct / total * 100))
```

기타사항

- train_X와 train_y를 만드는 부분은 다른 알고리즘을 사용사더라도 거의 유사합니다. 이 예에서는 SVM이 1차원 벡터만 입력으로 받기 때문에 average vector로 만들어서 사용했지만 CNN의 입력으로 2차원 벡터를 그대로 사용할 수도 있습니다.

- text_analyzer.py와 text_classifier_svm.py를 같은 폴더에 놓고 다음명령을 이용하여 실행하시면 됩니다. (파일 경로는 서버에 업로드하여 별도로 바꾸시지 않아도 실행은 됩니다.)
    - $ python3 text_classifier_svm.py

- matrix 관련 처리는 numpy 라이브러리를 이용하면 편하게 계산하실 수 있습니다.
    - http://www.numpy.org/

- 같이 첨부해 드리는 ontology_dstc2.json 파일은 시스템이 처리가능한 slot의 사전지식 데이터가 담겨있습니다. 이 예에서는 dialog-act에 대해서만 처리했지만 앞으로 slot, value 추정에 활용하실 수 있습니다.