POS TAGGER

IF4072 PEMROSESAN TEXT DAN SUARA BAHASA ALAMI

Bervianto Leo P. - 13514047
M. Az-zahid Adhitya S. - 13514095

Persiapan dan Sumber Kode

1. Download kode dan data set pada https://github.com/Azzahid/NLP-Pos-Tagger
2. Install Python pada perangkat Anda, disarankan menggunakan Python 2 dikarenakan script akan berjalan lancar pada Python 2. (Perlu penyesuaian beberapa hal untuk dapat dijalankan pada Python 3.
3. Install library yang dibutuhkan dengan perintah berikut pip install sklearn numpy scipy dill nltk
4. Corpus yang digunakan pada Universal Dependencies, sudah disediakan pada repositori di atas.

Teknik yang Digunakan

Dalam percobaan ini digunakan 90% kalimat menjadi data training sedangkan 10% dari keseluruhan kalimat. Pembuatan model digunakan dua algoritma yaitu HMM dan DTL (Decision Tree Learning). Dalam mencoba pemasangan TAG pada satu kalimat, dapat memilih salah satu model yang sudah dibuat.

Algoritma

1. HMM

Kami menggunakan library NLTK untuk menyelesaikan permasalahan *pos tagger* dengan algoritma HMM. Algoritma HMM pada library NLTK hanya mengambil data POS tag, kata dari tag tersebut, dan urutannya. Maka pada HMM kami tidak menggunakan *feature* yang banyak. Berikut kode dari HMM yang kami buat :

```
import re
from nltk.tag import hmm
from sklearn.externals import joblib
from nltk.tag.hmm  import HiddenMarkovModelTagger, HiddenMarkovModelTrainer
from nltk.probability import LidstoneProbDist

def tokenize_conllu_file( file_dir ):
    file = open(file_dir, 'r')
    line = file.readline()
    sentences = []
    sentence = []
    for line in file:
        if line[0] not in ['#','\n']:
            conllu_array = re.split(r'\t+', line)
            if conllu_array[0] == '1':
```

```
                if sentence :
                    sentences.append(sentence)
                sentence = []
            word = [conllu_array[1], conllu_array[3]]
            sentence.append((conllu_array[1], conllu_array[3]))

    return sentences

sentences = tokenize_conllu_file('../en-ud-dev.conllu')
cutoff = int(.9 * len(sentences))
training_sentences = sentences[:cutoff]
test_sentences = sentences[cutoff:]

print('Training Sentences : %d ' % (len(training_sentences)))
print('Testing Sentences : %d ' % (len(test_sentences)))


print 'Training Start'
trainer = HiddenMarkovModelTrainer()
tagger = trainer.train_supervised(training_sentences, estimator=lambda fd, bins:
LidstoneProbDist(fd, 0.1, bins))
print 'Training Completed'

print 'Testing Start'
tagger.test(test_sentences, verbose=False)
print 'Testing Completed'
```

2. DTL (*learning using feature*)

Pada algoritma DTL kami menggunakan *feature* yang ada dibawah untuk melakukan training. Dari fitur tersebut akan didapatkan tree yang akan menjadi *classifier* dari POS Tagger

```
import re
def tokenize_conllu_file( file_dir ):
    file = open(file_dir, 'r')
    line = file.readline()
    sentences = []
    sentence = []
    for line in file:
        if line[0] not in ['#','\n']:
            conllu_array = re.split(r'\t+', line)
            if conllu_array[0] == '1':
                if sentence :
                    sentences.append(sentence)
                sentence = []
            word = [conllu_array[1], conllu_array[3]]
            sentence.append(word)

    return sentences

def features(sentence, index):
    """ sentence: [w1, w2, ...], index: the index of the word """
    return {
        'word': sentence[index][0],
        'is_first': index == 0,
        'is_last': index == len(sentence) - 1,
        'is_capitalized': sentence[index][0][0].upper() == sentence[index][0][0],
        'is_all_caps': sentence[index][0].upper() == sentence[index][0],
        'is_all_lower': sentence[index][0].lower() == sentence[index][0],
        'prefix-1': sentence[index][0][0],
        'prefix-2': sentence[index][0][:2],
```

```
        'prefix-3': sentence[index][0][:3],
        'suffix-1': sentence[index][0][-1],
        'suffix-2': sentence[index][0][-2:],
        'suffix-3': sentence[index][0][-3:],
        'prev_word': '' if index == 0 else sentence[index - 1][0],
        'prev_pos': '' if index == 0 else sentence[index - 1][1],
        'next_word': '' if index == len(sentence) - 1 else sentence[index + 1][0],
        'has_hyphen': '-' in sentence[index][0],
        'is_numeric': sentence[index][0].isdigit(),
        'capitals_inside': sentence[index][0][1:].lower() != sentence[index][0][1:]
    }

def transform_to_dataset(sentences):
    X, Y = [], []
    for sentence in sentences:
        for index in range(len(sentence)):
            X.append(features(sentence, index))
            Y.append(sentence[index][1])

    return X, Y

sentences = tokenize_conllu_file('../en-ud-dev.conllu')
cutoff = int(.9 * len(sentences))
training_sentences = sentences[:cutoff]
test_sentences = sentences[cutoff:]

print('Training Sentences : %d ' % (len(training_sentences)))
print('Testing Sentences : %d ' % (len(test_sentences)))

X, y = transform_to_dataset(training_sentences)

from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction import DictVectorizer
from sklearn.pipeline import Pipeline

clf = Pipeline([
    ('vectorizer', DictVectorizer(sparse=False)),
    ('classifier', DecisionTreeClassifier(criterion='entropy'))
])
print 'Training Start'
clf.fit(X, y)   # Use only the first 10K samples if you're running it multiple times. It
takes a fair bit :)
print 'Training completed'

print 'Test Start'
X_test, y_test = transform_to_dataset(test_sentences)
print 'Test Completed'
print "Accuracy:", clf.score(X_test, y_test)
```

Fitur yang digunakan yaitu sebagai berikut.

1. Apakah suatu kata ada di awal atau di akhir kalimat.
2. Apakah suatu kata diawali dengan huruf kapital.
3. Apakah suatu kata memiliki kapital di seluruh huruf.
4. Apakah suatu kata hanya ada huruf kecil di setiap huruf.
5. Prefix hingga huruf pertama, hingga huruf kedua serta huruf ketiga.
6. Suffix hingga huruf pertama, hingga huruf kedua serta huruf ketiga.
7. Kata sebelumnya.

8. Tag kata sebelumnya
9. Kata setelahnya.
10. Apakah sebuah angka.
11. Apakah ada tanda hubung “-”.
12. Apakah ada huruf kapital di antara huruf awal maupun akhir dalam suatu kata.

Hasil Eksperimen

Eksperimen dilakukan dengan membuat model terlebih dahulu, lalu menggunakan model tersebut untuk melakukan POS Tagging. Gambar 1 menunjukkan proses training, gambar 2 menunjukkan proses tagging.

1. HMM

```
azzahid@azzahid-X450LD:~/Documents/Semester 7/NLP/Tugas/POS Tagger/src$ python hmm.py
Training Sentences : 1800
Testing Sentences : 201
Training Start
Training Completed
Testing Start
accuracy over 2387 tokens: 82.20
Testing Completed
azzahid@azzahid-X450LD:~/Documents/Semester 7/NLP/Tugas/POS Tagger/src$
```

```
bervianto@bervianto-X455LJ ~/NLP-Pos-Tagger/src $ python hmm_pos_tagger.py
Kalimat yang ingin diambil pos taggernya:
We want to find another place
[('We', 'PRON'), ('want', 'VERB'), ('to', 'PART'), ('find', 'VERB'), ('another',
 'DET'), ('place', 'NOUN')]
bervianto@bervianto-X455LJ ~/NLP-Pos-Tagger/src $
```

2. Learning (DTL)

```
azzahid@azzahid-X450LD:~/Documents/Semester 7/NLP/Tugas/POS Tagger/src$ python tokenizer.py
Training Sentences : 1800
Testing Sentences : 201
22749
Training Start
Training completed
Test Start
Test Completed
Accuracy: 0.836614997905
azzahid@azzahid-X450LD:~/Documents/Semester 7/NLP/Tugas/POS Tagger/src$
```

```
bervianto@bervianto-X455LJ ~/NLP-Pos-Tagger/src $ python dt-model-tagger.py
> Masukan kalimat yang ingin dilakukan POS TAGGER: We want to find another place
['PRON' 'VERB' 'VERB' 'ADJ' 'DET' 'VERB']
bervianto@bervianto-X455LJ ~/NLP-Pos-Tagger/src $
```

Simpulan

Dari uji coba kami pada kedua cara tersebut kami menemukan beberapa kelebihan dan kekurangan masing masing cara. Berikut tabel nya :

No.	Metode	Kelebihan	Kekurangan
1.	HMM (NLTK)	- Bisa menggunakan data yang banyak untuk training. - Cepat. - Hemat memori saat training.	- State tidak bisa menggunakan fitur yang banyak.
2.	Learning (DTL SKLearn)	- Bisa menggunakan berbagai fitur. - Akurasi lebih tinggi untuk data tes saat percobaan training.	- Waktu training lama. - Saat training memakan banyak memori.

Jadi pada tugas ini kami menyimpulkan bahwa pos tagger dapat menggunakan 2 cara ini sesuai dengan kondisi. Saat data yang digunakan banyak dan ingin lebih cepat maka gunakan HMM. Dan saat ingin akurasi yang lebih tinggi maka gunakan DTL.

Jika dilihat pada satu contoh kalimat yang kami ambil secara acak, HMM melakukan kinerja yang sangat baik. Perbedaan terjadi pada empat kata terakhir. Pada HMM melakukan POS Tag dengan baik namun pada model yang dihasilkan dari percobaan DTL masih kurang tepat. Walaupun akurasi lebih baik pada data uji yang diberikan saat training, namun belum pasti akan lebih baik saat mendapat kalimat-kalimat lainnya.

Referensi

- https://explosion.ai/blog/part-of-speech-pos-tagger-in-python
- http://nlpforhackers.io/training-pos-tagger/
- https://github.com/kunbudiharta/nltk-hmm-pos-tagger-bahasa-indonesia/blob/master/Hidden%20Markov%20Models%20(HMMs)%20%26%20NLTK.ipynb