Chunking

October 21, 2020

1 Shallow Parsing Chunker using Maximum Entropy Markov Model

CS 626 *Natural Language Processing*

Maheer Maloth Vineet Bhat T Sanjev Vishnu *Indian Institute of Technology Bombay*

1.0.1 Opening the files

[1]:
```
training_file = open('assignment2dataset/train.txt',mode="r")
test_file = open('assignment2dataset/test.txt',mode="r")
training_raw_data = training_file.readlines()
test_raw_data = test_file.readlines()
training_data = []
test_data = []
```

1.0.2 Adding chunk tag (B/I/O) and creating a new training data

[ ]:
```
for new_line in training_raw_data:
    new_line = new_line.split()
    if len(new_line) is not 0 :
        chunk_tag = new_line[2][0]
        new_line.pop()
        new_line.append(chunk_tag)
        training_data.append(new_line)
        print(new_line)
```

[3]:
```
#print(training_data)
i = 0
for word in training_raw_data:
    i = i+1
print(i)
```

220663

1.0.3 Adding chunk tag (B/I/O) and creating a new training data

[4]:
```
for new_line in test_raw_data:
    new_line = new_line.split()
    if len(new_line) is not 0 :
        chunk_tag = new_line[2][0]
        new_line.pop()
        new_line.append(chunk_tag)
        test_data.append(new_line)
        #print(new_line)
```

[ ]:
```
print(test_data)
```

1.0.4 Counting number of B, I, O tags in the training data set

[6]:
```
count_O = 0
count_I = 0
count_B = 0
for line in training_data :
    if line[2] == 'O':
        count_O += 1
    if line[2] == 'I':
        count_I += 1
    if line[2] == 'B':
        count_B += 1
print(count_B, count_I, count_O)
print("Total Number of Words: ", count_B + count_I + count_O)
```

```
106978 76847 27902
Total Number of Words:  211727
```

1.0.5 Forming a list of sentences which are lists in itself containing words

[7]:
```
def getSentences(train_data,test_data):
    sentence_labels = []
    sentence = []
    train_sentences = []
    train_labels = []
    for x in train_data:
        if(x[1]!='.'):
            sentence.append(x)
            sentence_labels.append(x[2])
        else:
            sentence.append(x)
            sentence_labels.append(x[2])
            train_sentences.append(sentence)
            train_labels.append(sentence_labels)
```

```
            sentence_labels = []
            sentence = []

    sentence = []
    sentence_labels = []
    test_sentences = []
    test_labels = []
    for x in test_data:
        if(x[1]!='.'):
            sentence.append(x)
            sentence_labels.append(x[2])
        else:
            sentence.append(x)
            sentence_labels.append(x[2])
            test_sentences.append(sentence)
            test_labels.append(sentence_labels)
            sentence_labels = []
            sentence = []
    print("Train Sentences: "+str(len(train_sentences)))
    print("Test Sentences: "+str(len(test_sentences)))

    return train_sentences,train_labels,test_sentences,test_labels
```

[8]:
```
train_sentences,train_labels,test_sentences,test_labels =␣
 ↪getSentences(training_data,test_data)
```

```
Train Sentences: 8827
Test Sentences: 1975
```

1.0.6 To check if there are any errors in generation of training and test sentences

[9]:
```
for i in range(len(train_sentences)):
    #print(train_sentences[i])
    #print(train_labels[i],"\n")
    if len(train_sentences[i]) is len(train_labels[i]):
        continue
    else :
        print("Danger")

for i in range(len(test_sentences)):
    #print(test_sentences[i])
    #print(test_labels[i])
    if len(test_sentences[i]) is len(test_labels[i]):
        continue
    else :
        print("Danger")
```

[10]:
```
from gensim.models import Word2Vec
w2v_words = Word2Vec(training_data, size=30)
```

[11]:
```
print(w2v_words)
```

```
Word2Vec(vocab=4440, size=30, alpha=0.025)
```

1.0.7 Creating a feature set for a given word

[12]:
```
from nltk.stem import PorterStemmer
```

[13]:
```
ps = PorterStemmer()
def word2features(sent, i):
    word = sent[i][0]
    postag = sent[i][1]

    features = {
        'bias': 1.0,
        'word.lower()': word.lower(),
        'word_stem':ps.stem(word),
        'word.isupper()': word.isupper(),
        'word.istitle()': word.istitle(),
        'word.isdigit()': word.isdigit(),
        'postag': postag,
        'prefix1':word[0:1],
        'prefix2':word[0:2],
        'suffix1':word[-2:],
        'suffix2':word[-3:],
        'suffix3':word[-4:],

    }


    if i > 0:
        word1 = sent[i-1][0]
        postag1 = sent[i-1][1]
        chunktag1 = sent[i-1][2]
        features.update({
            '-1:word.lower()': word1.lower(),
            '-1:word_stem':ps.stem(word1),
            '-1:word.istitle()': word1.istitle(),
            '-1:word.isupper()': word1.isupper(),
            '-1:postag': postag1,
            '-1:prefix1':word1[0:1],
            '-1:prefix2':word1[0:2],
            '-1:suffix1':word1[-2:],
            '-1:suffix2':word1[-3:],
```

```
            '-1:suffix3':word1[-4:],
            '-1:chunktag':chunktag1

        })
    else:
        features['SOS'] = True

    if i > 1:
        word1 = sent[i-2][0]
        postag1 = sent[i-2][1]
        chunktag1 = sent[i-2][2]
        features.update({
            '-2:word.lower()': word1.lower(),
            '-2:word_stem':ps.stem(word1),
            '-2:word.istitle()': word1.istitle(),
            '-2:word.isupper()': word1.isupper(),
            '-2:postag': postag1,
            '-2:prefix1':word1[0:1],
            '-2:prefix2':word1[0:2],
            '-2:suffix1':word1[-2:],
            '-2:suffix2':word1[-3:],
            '-2:suffix3':word1[-4:],
            '-2:chunktag':chunktag1

        })
    else:
        if(i==1):
            features['SecondWord'] = True

    if i < len(sent)-1:
        word1 = sent[i+1][0]
        postag1 = sent[i+1][1]
        chunktag1 = sent[i+1][2]
        features.update({
            '+1:word.lower()': word1.lower(),
            '+1:word_stem':ps.stem(word1),
            '+1:word.istitle()': word1.istitle(),
            '+1:word.isupper()': word1.isupper(),
            '+1:postag': postag1,
            '+1:prefix1':word1[0:1],
            '+1:prefix2':word1[0:2],
            '+1:suffix1':word1[-2:],
            '+1:suffix2':word1[-3:],
            '+1:suffix3':word1[-4:],
            '+1:chunktag':chunktag1
        })
    else:
```

```
        features['EOS'] = True


    return features
```

[14]:
```
def sent2features(sent):
    return [word2features(sent, i) for i in range(len(sent))]
```

[15]:
```
def sent2labels(sent):
    return [label for token, postag, label in sent]
```

[16]:
```
def sent2tokens(sent):
    return [token for token, postag, label in sent]
```

[17]:
```
X_test = [sent2features(s) for s in test_sentences]
```

[18]:
```
y_test = [sent2labels(s) for s in test_sentences]
```

[19]:
```
y_train = [sent2labels(s) for s in train_sentences]
```

[20]:
```
X_train = [sent2features(s) for s in train_sentences]
```

[21]:
```
values = []
for sent in X_test:
    for word in sent:
        for var in word.values():
            values.append(var)
for sent in X_train:
    for word in sent:
        for var in word.values():
            values.append(var)
print(len(values))
#print(values)
```

```
11216808
```

[22]:
```
for feature, value in X_test[0][5].items():
    print ("{:<20} {:<15} ".format(feature, value))
```

```
bias                 1.0
word.lower()         unit
word_stem            unit
word.isupper()       0
word.istitle()       0
word.isdigit()       0
postag               NN
prefix1              u
```

```
prefix2                un
suffix1                it
suffix2                nit
suffix3                unit
-1:word.lower()        tulsa
-1:word_stem           tulsa
-1:word.istitle()      1
-1:word.isupper()      0
-1:postag              NNP
-1:prefix1             T
-1:prefix2             Tu
-1:suffix1             sa
-1:suffix2             lsa
-1:suffix3             ulsa
-1:chunktag            I
-2:word.lower()        's
-2:word_stem           's
-2:word.istitle()      0
-2:word.isupper()      0
-2:postag              POS
-2:prefix1             '
-2:prefix2             's
-2:suffix1             's
-2:suffix2             's
-2:suffix3             's
-2:chunktag            B
+1:word.lower()        said
+1:word_stem           said
+1:word.istitle()      0
+1:word.isupper()      0
+1:postag              VBD
+1:prefix1             s
+1:prefix2             sa
+1:suffix1             id
+1:suffix2             aid
+1:suffix3             said
+1:chunktag            B
```

1.0.8 Using Label Encoder to convert the feature set into numbers

[23]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(values)
```

[23]:
```
LabelEncoder()
```

[24]:
```
import matplotlib.pyplot as plt
```

1.0.9 Checking if the transformer gives the same output for the same features

[25]:
```
d = le.transform(["I","am","a","good","boy"])
e = le.transform(["I", "am", "in", "love"])
f = le.transform(["I","$","to","kiss", "you","in", "capital"])
c = le.classes_
print(d)
print(e)
print(f)
print(len(le.classes_))
```

```
[ 4293  6603  5736 15934  8640]
[ 4293  6603 17743 20216]
[ 4293     2 30921 19136 33649 17743  9229]
33777
```

[36]:
```
f = open("tr.txt", "w")
```

[ ]:
```
f.write(X_train)
```

2 Label Encoding

Not necessary Heavily consumes memory Time taken is unreasonably high ~ 25 hours

[26]:
```
#X_train = [sent2features(s) for s in train_sentences]
"""
count = 0

for sent in X_train:
    for word in sent:
        for key in word:

            e = []
            e.append('{}'.format(word[key]))
            temp = le.transform(e)
            for featvalue in temp:
                word[key] = featvalue

            if(count == 20000):
                break
            count += 1
            if(count%100 == 0):
                print(count)
            #print(word[key])

print(count)
""""
```

```
#print(X_train[0][5])
#print("label",y_train[0][5])
```

```
  File "<ipython-input-26-238ae3c3f8fc>", line 23
    """"

^
SyntaxError: EOL while scanning string literal
```

[26]:
```
tuple_entry = (X_train[0][5], y_train[0][5])
print(tuple_entry)
```

```
({'bias': 1.0, 'word.lower()': 'widely', 'word_stem': 'wide', 'word.isupper()':
False, 'word.istitle()': False, 'word.isdigit()': False, 'postag': 'RB',
'prefix1': 'w', 'prefix2': 'wi', 'suffix1': 'ly', 'suffix2': 'ely', 'suffix3':
'dely', '-1:word.lower()': 'is', '-1:word_stem': 'is', '-1:word.istitle()':
False, '-1:word.isupper()': False, '-1:postag': 'VBZ', '-1:prefix1': 'i',
'-1:prefix2': 'is', '-1:suffix1': 'is', '-1:suffix2': 'is', '-1:suffix3': 'is',
'-1:chunktag': 'B', '-2:word.lower()': 'pound', '-2:word_stem': 'pound',
'-2:word.istitle()': False, '-2:word.isupper()': False, '-2:postag': 'NN',
'-2:prefix1': 'p', '-2:prefix2': 'po', '-2:suffix1': 'nd', '-2:suffix2': 'und',
'-2:suffix3': 'ound', '-2:chunktag': 'I', '+1:word.lower()': 'expected',
'+1:word_stem': 'expect', '+1:word.istitle()': False, '+1:word.isupper()':
False, '+1:postag': 'VBN', '+1:prefix1': 'e', '+1:prefix2': 'ex', '+1:suffix1':
'ed', '+1:suffix2': 'ted', '+1:suffix3': 'cted', '+1:chunktag': 'I'}, 'I')
```

2.0.1 Creating Training Data

[ ]:
```
tr_data = []

for sent in range(len(X_train)):
    for word in range(len(X_train[sent])):
        print(sent, word)
        new_tuple = (X_train[sent][word], y_train[sent][word])
        tr_data.append(new_tuple)
```

[29]:
```
print(len(tr_data))
```

```
211727
```

[30]:
```
print(tr_data[26])
```

```
({'bias': 1.0, 'word.lower()': 'a', 'word_stem': 'a', 'word.isupper()': False,
'word.istitle()': False, 'word.isdigit()': False, 'postag': 'DT', 'prefix1':
'a', 'prefix2': 'a', 'suffix1': 'a', 'suffix2': 'a', 'suffix3': 'a',
'-1:word.lower()': 'show', '-1:word_stem': 'show', '-1:word.istitle()': False,
```

```
'-1:word.isupper()': False, '-1:postag': 'VB', '-1:prefix1': 's', '-1:prefix2':
'sh', '-1:suffix1': 'ow', '-1:suffix2': 'how', '-1:suffix3': 'show',
'-1:chunktag': 'I', '-2:word.lower()': 'to', '-2:word_stem': 'to',
'-2:word.istitle()': False, '-2:word.isupper()': False, '-2:postag': 'TO',
'-2:prefix1': 't', '-2:prefix2': 'to', '-2:suffix1': 'to', '-2:suffix2': 'to',
'-2:suffix3': 'to', '-2:chunktag': 'I', '+1:word.lower()': 'substantial',
'+1:word_stem': 'substanti', '+1:word.istitle()': False, '+1:word.isupper()':
False, '+1:postag': 'JJ', '+1:prefix1': 's', '+1:prefix2': 'su', '+1:suffix1':
'al', '+1:suffix2': 'ial', '+1:suffix3': 'tial', '+1:chunktag': 'I'}, 'B')
```

2.0.2 Finding the frequency of each features and it's joint feature

[ ]:
```
"""feat_count = []
n = 0
for feature in c:
    n = n + 1
    counter = 0
    for against in values:
        if feature == against:
            counter += 1
    feat_count.append(counter)
    print("Completed", n*100/len(c))
    """
```

[ ]:
```
#print(feat_count)
```

[ ]:
```
#plt.plot(c[7170:7180], feat_count[7170:7180])
```

2.0.3 Generating a cumulative distribution

[ ]:
```
'''for i in range(len(feat_count)):
    if i == 0:
        feat_count[0] = feat_count[0]
    else :
        feat_count[i] = feat_count[i] + feat_count[i-1]
print(feat_count)'''
```

[ ]:
```
#plt.figure(figsize=(20,10))
#plt.plot(c[:2000],feat_count[:2000])
#plt.savefig('feature_encoding.png', dpi = 300)
```

2.0.4 Importing Maxent Classifier NLTK

[31]:
```
import nltk
nltk.usage(nltk.classify.ClassifierI)
```

```
ClassifierI supports the following operations:
  - self.classify(featureset)
  - self.classify_many(featuresets)
  - self.labels()
  - self.prob_classify(featureset)
  - self.prob_classify_many(featuresets)
```

[32]:
```python
from nltk.classify import MaxentClassifier as mc
```

2.0.5 Creating Test Data

[33]:
```python
ts_data = []

for sent in range(len(X_test)):
    for word in range(len(X_test[sent])):
        #print(sent, word)
        new_tuple = (X_test[sent][word])
        ts_data.append(new_tuple)
```

[34]:
```python
type(ts_data[0].items())
```

[34]:
```
dict_items
```

[35]:
```python
ts_labels = []

for sent in y_test:
    for word in sent:
        ts_labels.append(word)
```

2.0.6 Training the dataset

[36]:
```python
import timeit
```

[37]:
```python
start = timeit.default_timer()
a = mc.train(train_toks= tr_data ,trace=0, max_iter=21)
stop = timeit.default_timer()

print("Time : ", stop-start)
```

```
Time :  2449.799099438
```

[38]:
```python
print(len(ts_labels), len(ts_data))
```

```
47377 47377
```

2.0.7 Predicting the labels

[39]:
```
pred_labels = []
for i in range(len(ts_data)):

    probability_distribution = a.prob_classify(ts_data[i])
    #print('%8.5f%8.5f%8.5f' % (probability_distribution.prob('B'),␣
↪probability_distribution.prob('I'), probability_distribution.prob('O')),␣
↪end=' ')
    max = 0
    if(probability_distribution.prob('B')>max):
        max = probability_distribution.prob('B')
        label = 'B'
    if(probability_distribution.prob('I')>max):
        max = probability_distribution.prob('I')
        label = 'I'
    if(probability_distribution.prob('O')>max):
        max = probability_distribution.prob('O')
        label = 'O'
    pred_labels.append(label)
    #print(label, ts_labels[i])
```

2.0.8 Metrics to measure effectiveness of the model

[40]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

[41]:
```
confusion_matrixs = confusion_matrix(ts_labels, pred_labels)
```

[42]:
```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    """
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    ---------
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from matplotlib.pyplot.cm
```

```
                 see http://matplotlib.org/examples/color/colormaps_reference.
↪html
                 plt.get_cmap('jet') or plt.cm.Blues

   normalize:    If False, plot the raw numbers
                 If True, plot the proportions

   Usage
   -----
   plot_confusion_matrix(cm           = cm,                  # confusion␣
↪matrix created by
                                                             # sklearn.metrics.
↪confusion_matrix
                         normalize    = True,                # show proportions
                         target_names = y_labels_vals,       # list of names␣
↪of the classes
                         title        = best_estimator_name) # title of graph

   Citiation
   ---------
   http://scikit-learn.org/stable/auto_examples/model_selection/
↪plot_confusion_matrix.html

   """
   import matplotlib.pyplot as plt
   import numpy as np
   import itertools

   accuracy = np.trace(cm) / float(np.sum(cm))
   misclass = 1 - accuracy

   if cmap is None:
       cmap = plt.get_cmap('Oranges')

   plt.figure(figsize=(8, 6))
   plt.imshow(cm, interpolation='nearest', cmap=cmap)
   plt.title(title)
   plt.colorbar()

   if target_names is not None:
       tick_marks = np.arange(len(target_names))
       plt.xticks(tick_marks, target_names, rotation=45)
       plt.yticks(tick_marks, target_names)

   if normalize:
       cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

```
    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

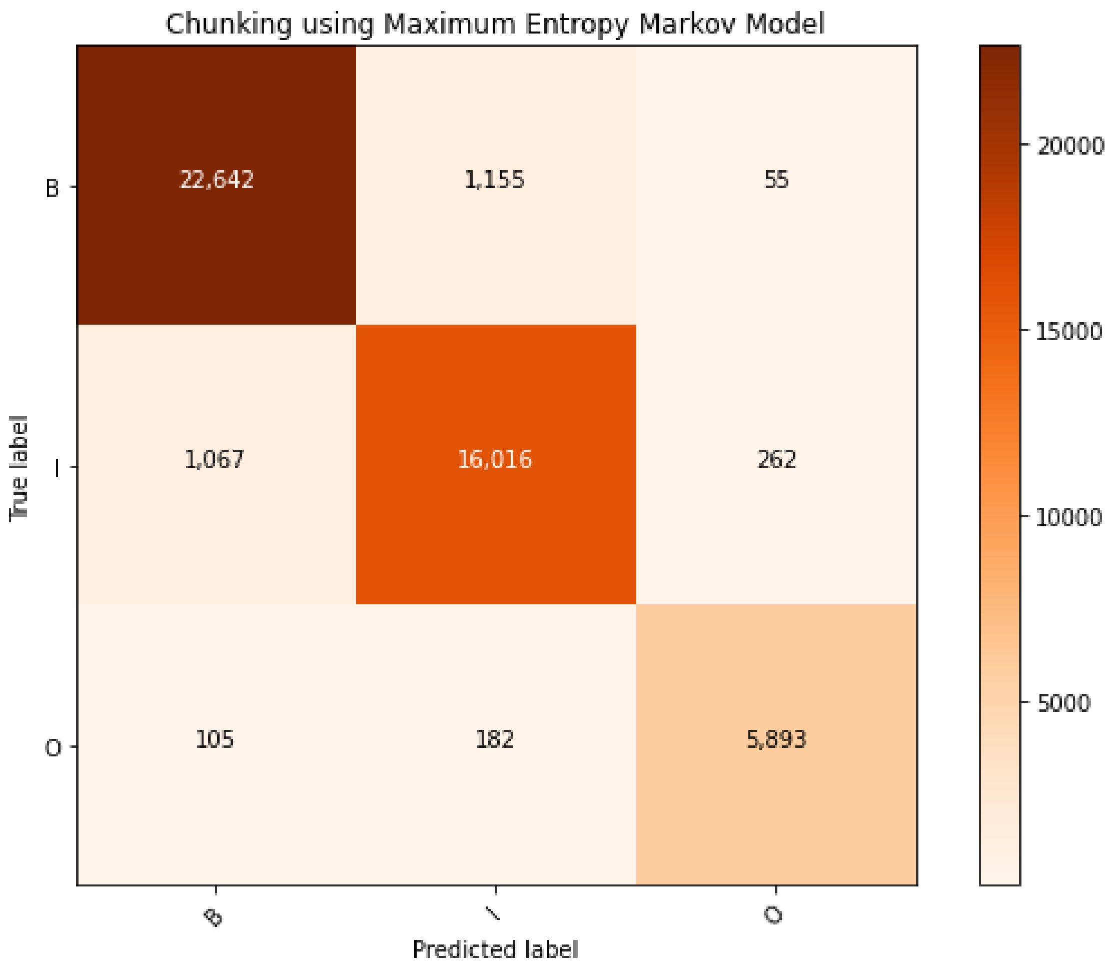
2.0.9 Accuracy

[43]:
```
B_identified = confusion_matrixs[0][0]/sum(confusion_matrixs[0])
I_identified = confusion_matrixs[1][1]/sum(confusion_matrixs[1])
O_identified = confusion_matrixs[2][2]/sum(confusion_matrixs[2])
print("Accuracy of B (correctly identified) : ", B_identified*100)
print("Accuracy of I (correctly identified) : ", I_identified*100)
print("Accuracy of O (correctly identified) : ", O_identified*100)
print("Overall Accuracy of the MEMM Chunker : ",accuracy_score(ts_labels,␣
↪pred_labels)*100)
```

```
Accuracy of B (correctly identified) :  94.9270501425457
Accuracy of I (correctly identified) :  92.3378495243586
Accuracy of O (correctly identified) :  95.35598705501617
Overall Accuracy of the MEMM Chunker :  94.03508031323216
```

2.0.10 Heat Map (Confusion Matrix)

[44]:
```
plot_confusion_matrix(confusion_matrixs, target_names=['B','I','O'],␣
↪title='Chunking using Maximum Entropy Markov Model', normalize=False)
```


Chunking using Maximum Entropy Markov Model
True label
Predicted label
B
I
O
22,642
1,155
55
1,067
16,016
262
105
182
5,893
20000
15000
10000
5000

```
[ ]:
```