WebContent_Conditional Random Field

December 6, 2018

1 Conditional Random Fields

In NLP, it is a common task to extract words/ phrases of particular types from a given sentence or paragraph. For example, when performing analysis of a corpus of travel articles, we may want to know which travelling destination are mentioned in the articles, and how many articles are related to each of these travelling document.

Conditional Random Fields are discriminative model used for predicting the sequence of labels. It uses information from the context through previous labels and thus helping the model in turn to make better prediction for unseen text/word etc.This technqiue can be very well used for Name Entity Recognition(NER), which is to give a accurate label to given word(incase of NLP).For Example: •*"Shahjahan went to see Taj Mahal"* - in this example the *Taj Mahal* can be location representing a monument in Agra or can correspond to Tea Bags.•*"This Apple product is one in its segment"* - now here should the word Apple is to be associated with the Apple(Tech company) or a food item. Hence, given a sequence of words identifying the correct sequence of labels is the problem to be solved and depending upon the problem finding the correct sequence of label, where label can be $person, location, organization, etc$ **for instance. So, the CRF can solve these kind of problem which is a challenging task for the old traditional graphical models like** $\underline{HMM}$ **or** $\underline{MaxEnt}$**.Let's try to understand through a bit more theory first and then I'll give a small implementation of CRF.I'll highlight few of the keywords discussed above first.**

1.0.1 Discriminative Model:

In Machine Learning we have two different types of modelling technqiue. • Discriminative - Logistic Regression, classifier based on Maximum Likelihood Estimation• Generative - Naive Bayes is popular and simple probablistic classifier

1.0.2 CRF Model: A special case of undirected graphical model.

So, the basic crux of the CRF is to generate labels given huge amount of data where the input data is sequential. Also we consider previous context while making a prediction for given data point. •Let W be the tokens in the document and w_i be the corresponding word observed. • y_i be the hidden label. The conditional distribtuion is modeled as:

$$\hat{y} = \underset{y}{\text{argmax}} P(y|x)$$

To model the appropriate behaviour of CRF we need a {feature function} which have the below parameters: • Set of Input Vectors - W • i - the word for which we want to predict label • Label for data points $w_{1:i-1}$ • label of point i in W The feature functions are the key components of CRF and in our special case of linear-chain CRF, and the general form is:

$$f_1(y_{i-1}, y_i, w_{1:N}, i)$$

which looks at a pair of adjacent labels y_{i1}, y_i, the whole input sequence $w_{1:N}$, and the current position i. For Ex: we can define a simple feature function which produces binary values: *1* for the current word is *TajMahal*, and if the current state y_n is *Monument*.Usage of such feature depends on the corresponding weight λ_1. If the $\lambda_1 > 0$ then label *Monument* is preferred for above example otherwise CRF will try to avoid the label *Monument* for the text *TajMahal*

1.0.3 CRF vs HMM:

Although both are used to model sequential data, they are different algorithms.Hidden Markov Models are generative, and give output by modeling the joint probability distribution whereeas Conditional Random Fields as mentioned above are discriminative, and model the conditional probability distribution. CRFs don't rely on the independence assumption (labels are independent of each other), and avoid label bias. One way to look at it is that Hidden Markov Models are a very specific case of Conditional Random Fields, with constant transition probabilities used instead. HMMs are based on Naive Bayes, which we say can be derived from Logistic Regression, from which CRFs are derived.

1.0.4 CRF Application:

• Part-of-Speech Tagging (POS). • Named Entity Recognition.• Image segmentation

1.0.5 Disadvantage:

Highly computational cost as well as complexity at the training stage of the algorithm. Retraining on new data is difficult.

1.0.6 Minor Implementation:

Sequence labelling

In [3]:
```
import nltk
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from bs4 import BeautifulSoup as bs
from bs4.element import Tag
import codecs
import pycrfsuite
#nltk.download('averaged_perceptron_tagger')
```

In [12]:
```
#Given the POS tags, generate more features for each
#of the tokens in the dataset.Below are some of the commonly used features
#for a word w in named entity recognition:
```

```python
def word2features(doc, i):
    #print(doc,i)
    word = doc[i][0]
    postag = doc[i][1]
    # Common features for all words
    features = [
        'bias',
        'word.lower=' + word.lower(),
        'word[-2:]=' + word[-2:],
        'word.isupper=%s' % word.isupper(),
        'word.isdigit=%s' % word.isdigit(),
        'postag=' + postag
                ]
    #print(features)
    # Features for words that are not at the beginning of a document
    if i > 0:
        word1 = doc[i-1][0]
        postag1 = doc[i-1][1]
        features.extend([
            '-1:word.lower=' + word1.lower(),
            '-1:word.isupper=%s' % word1.isupper(),
            '-1:word.isdigit=%s' % word1.isdigit(),
            '-1:postag=' + postag1
        ])
    else:
        # Indicate that it is the 'beginning of a document'
        features.append('<s>')

    # Features for words that are not at the end of a document
    if i < len(doc)-1:
        word1 = doc[i+1][0]
        postag1 = doc[i+1][1]
        features.extend([
            '+1:word.lower=' + word1.lower(),
            '+1:word.isupper=%s' % word1.isupper(),
            '+1:word.isdigit=%s' % word1.isdigit(),
            '+1:postag=' + postag1
        ])
    else:
        # Indicate that it is the 'end of a document'
        features.append('</s>')
    #print(features,"\n")
    return features
```

In [5]:
```python
# extracting features in documents
def extract_features(doc):
    return [word2features(doc, i) for i in range(len(doc))]
```

In [6]:
```python
# generating the list of labels for each document
```

```python
def get_labels(doc):
    return [label for (token, postag, label) in doc]
```

In [16]:
```python
# Read data file and parse the XML
with codecs.open("testFile.xml", "r", "utf-8") as infile:
    soup = bs(infile, "html5lib")
#print(soup.prettify(),"\n")
for elem in soup.find_all("document"):
    textContent = []
    # Loop through each child of the element under "textwithnamedentities"
    for c in elem.find("textwithnamedentities").children:
        if type(c) == Tag:
            if c.name == "namedentityintext":
                label = "N"  # part of a named entity
            else:
                label = "I"  # irrelevant word
            for w in c.text.split(" "):
                if len(w) > 0:
                    textContent.append((w, label))
    docs.append(textContent)
data = []
for i, doc in enumerate(docs):
    # fetching list of tokens in the document
    for _tuple in doc:
        tokens=_tuple[0]
    #POS tagging
    tagged = nltk.pos_tag(tokens)
    # creating a list of word-pos tag- label(N/I)
    data.append([(w, pos,label) for (w, label), (word, pos) in zip(doc, tagged)])
X = [extract_features(doc) for doc in data]
Y = [get_labels(doc) for doc in data]
#splitting data for training the data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

trainer = pycrfsuite.Trainer(verbose=True)

# Submit training data to the trainer
for xseq, yseq in zip(X_train, Y_train):
    trainer.append(xseq, yseq)

# Set the parameters of the model
trainer.set_params({
    # coefficient for L1 penalty
    'c1': 0.2,
    # coefficient for L2 penalty
    'c2': 0.01,
    # maximum number of iterations
    'max_iterations': 20,
```

```python
        # whether to include transitions that
        # are possible, but not observed
        'feature.possible_transitions': True
    })

    # Provide a file name as a parameter to the train function, to save the model when tr
    trainer.train('crf.model')
    #print(X_test)
    # Generate predictions
    tagger = pycrfsuite.Tagger()
    tagger.open('crf.model')
    Y_pred = [tagger.tag(xseq) for xseq in X_test]

    # Let's take a look at a random sample in the testing set
    i = 2
    for x, y in zip(Y_pred[i], [x[1].split("=")[1] for x in X_test[i]]):
        print("%s (%s)" % (y, x))

    # Create a mapping of labels to indices
    labels = {"N": 1, "I": 0}

    # Convert the sequences of tags into a 1-dimensional array
    predictions = np.array([labels[tag] for row in Y_pred for tag in row])
    truths = np.array([labels[tag] for row in Y_test for tag in row])

    # Print out the classification report
    print(classification_report(truths, predictions,target_names=["I", "N"]))
```

```
Feature generation
type: CRF1d
feature.minfreq: 0.000000
feature.possible_states: 0
feature.possible_transitions: 1
0...1...2...3...4...5...6...7...8...9...10
Number of features: 82
Seconds required: 0.003

L-BFGS optimization
c1: 0.200000
c2: 0.010000
num_memories: 6
max_iterations: 20
epsilon: 0.000010
stop: 10
delta: 0.000010
linesearch: MoreThuente
linesearch.max_iterations: 20
```

```
***** Iteration #1 *****
Loss: 24.519642
Feature norm: 1.000000
Error norm: 37.948507
Active features: 80
Line search trials: 1
Line search step: 0.040064
Seconds required for this iteration: 0.000

***** Iteration #2 *****
Loss: 16.058301
Feature norm: 1.317783
Error norm: 10.559112
Active features: 81
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.001

***** Iteration #3 *****
Loss: 14.637365
Feature norm: 1.545410
Error norm: 5.610032
Active features: 79
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #4 *****
Loss: 13.580063
Feature norm: 1.782253
Error norm: 5.336732
Active features: 82
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.001

***** Iteration #5 *****
Loss: 10.607076
Feature norm: 2.664108
Error norm: 3.458049
Active features: 76
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #6 *****
Loss: 8.321739
Feature norm: 3.715773
```

```
Error norm: 2.400939
Active features: 61
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.001

***** Iteration #7 *****
Loss: 7.637296
Feature norm: 4.391192
Error norm: 1.258366
Active features: 60
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.001

***** Iteration #8 *****
Loss: 7.482298
Feature norm: 4.629935
Error norm: 0.706088
Active features: 55
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #9 *****
Loss: 7.348097
Feature norm: 4.924919
Error norm: 0.678835
Active features: 49
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #10 *****
Loss: 7.182814
Feature norm: 5.248972
Error norm: 0.352148
Active features: 40
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #11 *****
Loss: 7.149011
Feature norm: 5.271214
Error norm: 0.484922
Active features: 40
Line search trials: 1
```

```
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #12 *****
Loss: 7.095619
Feature norm: 5.327155
Error norm: 0.192273
Active features: 37
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #13 *****
Loss: 7.038118
Feature norm: 5.518987
Error norm: 0.440120
Active features: 37
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #14 *****
Loss: 6.994441
Feature norm: 5.597406
Error norm: 0.300001
Active features: 37
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #15 *****
Loss: 6.941759
Feature norm: 5.724182
Error norm: 0.622532
Active features: 42
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #16 *****
Loss: 6.917742
Feature norm: 5.848332
Error norm: 0.425420
Active features: 39
Line search trials: 2
Line search step: 0.500000
Seconds required for this iteration: 0.000
```

```
***** Iteration #17 *****
Loss: 6.900773
Feature norm: 6.044487
Error norm: 0.628616
Active features: 39
Line search trials: 2
Line search step: 0.500000
Seconds required for this iteration: 0.000

***** Iteration #18 *****
Loss: 6.896096
Feature norm: 6.132641
Error norm: 0.443689
Active features: 31
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #19 *****
Loss: 6.889608
Feature norm: 6.157883
Error norm: 0.417977
Active features: 31
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

***** Iteration #20 *****
Loss: 6.885546
Feature norm: 6.116811
Error norm: 0.240603
Active features: 31
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.000

L-BFGS terminated with the maximum number of iterations
Total seconds required for training: 0.008

Storing the model
Number of active features: 31 (82)
Number of active attributes: 25 (57)
Number of active labels: 2 (2)
Writing labels
Writing attributes
Writing feature references for transitions
Writing feature references for attributes
Seconds required: 0.001
```

```
tajmahal (N)
is (I)
a (N)
tea (N)
product (I)
             precision    recall  f1-score   support

          I       1.00      1.00      1.00         7
          N       1.00      1.00      1.00         8

avg / total       1.00      1.00      1.00        15
```