Middle East Technical University Department of Computer Engineering

CENG 463

Introduction to Natural Language Processing

Spring 2017-2018

Homework 2

Due date: 19 April 2018, Thursday, 23:55

1 Objectives

In this assignment, you are expected to implement a series of sequence taggers. Your taggers will assign part-of-speech tags, phrase chunk labels and named-entity labels to the words in sentences. You will use 2 models, namely Logistic Regression and Conditional Random Fields for each of the 3 tasks.

You are also expected to write a report containing the analysis of your taggers and explain basic machine learning concepts. This is a research assignment and you are expected to experiment with the taggers and try different alternatives. Therefore, your reports are more important than the implementations of the taggers.

Keywords: *sequence tagging, part of speech tagging, chunking, shallow parsing, named entity recognition, maximum entropy models, logistic regression, conditional random fields*

2 Tagging Words

The task of labelling words in a sentence is at the beginning of almost every NLP task. Categorizing words is the most fundamental thing before moving onto more complex problems.

2.1 Part-of-Speech Tags

A part of speech of a word is the lexical category of the word (also word class, morphological class, lexical class, or lexical tag). A part of speech of a word provides information about a word and its neighbours, word's pronunciation, its morphological analysis or can be used in speech recognition, parsing, translation, information retrieval and question answering tasks. Many NLP applications may benefit from syntactically disambiguated text. Part of speech tagging is the process of assigning tags (class markers) to words. A word can be a member of multiple classes and main task of a tagger is to solve this ambiguity.

2.2 Chunks

Phrases in a sentence can be seen as blocks that move together. Finding and extracting them can help many NLP tasks. For example, a noun phrase (NP) can be an answer to a question and we can use chunking to label the words in that phrase instead of parsing the sentence to extract relations. Therefore, chunking task is also called shallow parsing. The categories include verb phrases (VP), prepositional phrases (PP), adjectives (ADJP), adverbs (ADVP) and many others.

2.3 Entities

Entity recognition tasks deal with names of persons, organizations, locations, expressions of times, quantities, monetary values etc. In this assignment, we focus on named entities which are person (PER), location (LOC) and organization (ORG) names and label some other miscellaneous names (MISC). The task again can be seen as a sequence labelling problem and solved similarly as a chunking or noun phrase chunking problem.

2.4 BIO Encoding

IOB or BIO encoding is used to assign labels in chunking or NER tasks. It is used to denote whether a word is at the beginning of a chunk/phrase/entity (B), inside it or continuation of the entity (I) or outside of any category (O). For each category, prefixes of 'B-' or 'I-' is used to label tokens. CoNLL (Conference on Natural Language Learning) datesets that are used in this assignment have BIO encoding.

3 Models

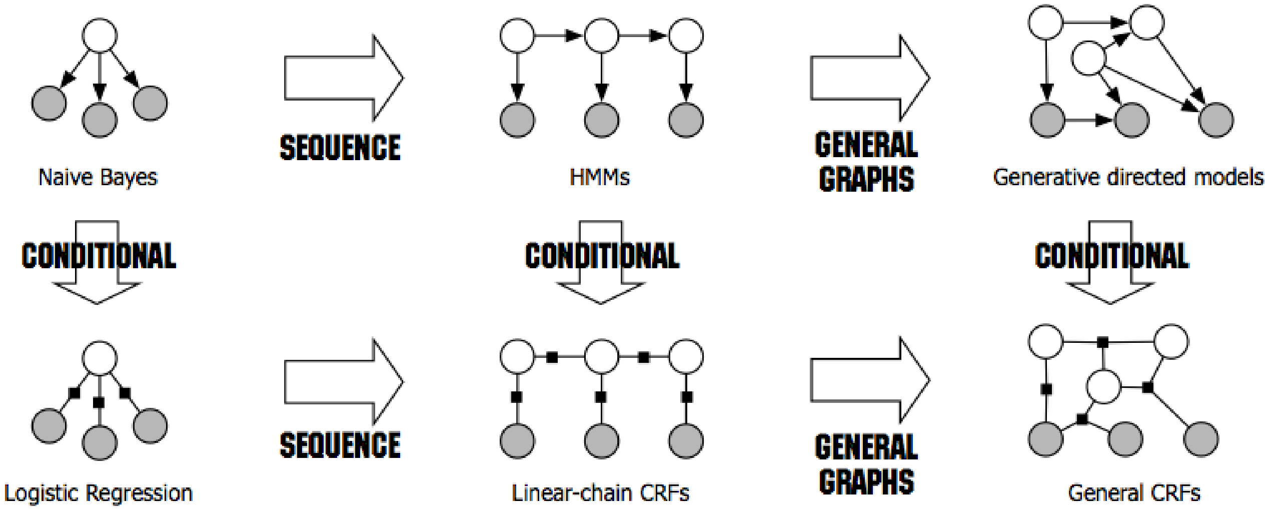

Figure 1: Diagram of relationships between models from Andrew McCallum's *An Introduction to Conditional Random Fields*

In classical machine learning, programs learn a function that maps input data to desired output labels. Data are usually represented as a set of features that are manually selected or crafted. Then a function is learned by usually optimizing an objective function (such as minimizing error). In this assignment, we are not interested in finding the best algorithm or how to optimize the solution. We will focus on the features that are used to represent data and try to find the best set of features.

You will train two models for all 3 tasks. The first one is Maximum Entropy or Logistic Regression models and the second one is Conditional Random Fields.

3.1 Maximum Entropy Classifiers - Multinomial Logistic Regression

MaxEnt Classifiers try to predict the probability of an outcome given a set of variables. In this assignment you will represent an instance of data variable with a set of features and MaxEnt models try to find the probabilities of each class label given those features. The learning algorithm for MaxEnt tries to maximize the likelihood of the correct class for a given sample.

You will extract features from the data and train models based on these features. See the references for example features.

3.2 Conditional Random Fields

Logistic regression classifies each sample without the context it lies in. We try to add context information as additional features to the model. CRFs are more general and can take context into account to handle sequences. The sentences will be tagged as whole sequences instead of one word at a time model of MaxEnt.

You will extract very similar features in your CRF models and probably have better results than Logistic Regression models.

4 Data

There is a separate data set for each task. Each file contains tab separated fields for each token in a single line. The sentences are separated with empty lines. In each line, the first field contains the token and the last field contains the label that we want to learn and predict. Intermediate fields contains additional information that can be used to train models. These additional fields, however, will not be provided while using your models. You can train your models to predict them or use already build models to extract these features.

4.1 Part of Speech Tagging

POS data contains there fields and two types of POS tags. We are interested in predicting the POS tags in the third field. The tags in the second field are called universal part of speech tags and UPOS is a simpler set that can be adapted to almost all languages. The tags in the third field are specific to English. Your models will predict these tags, however you can also build taggers for UPOS and use universal POS tags as features in your models.

You have *en-ud-train.conllu* and *en-ud-dev.conllu* files as training and development sections. A sample from the data can be seen below:

```
I          PRON    PRP
do         AUX     VBP
n't        PART    RB
think      VERB    VB
it         PRON    PRP
matters    VERB    VBZ
```

4.2 Chunking

The chunking data contains three fields per token. The chunks are given as B-CHUNK tag followed by zero or more I-CHUNK tags. Punctuations etc. have O tag. The second field contains POS tags derived by the Brill tagger. These are not gold (correct) POS tags. You can use these tags as features in your models, However, if you use them you also need to predict them during test time with your pos taggers (build on the POS tag data or retrained on this data) or some prebuilt taggger like Brill, Stanford, Senna or some tagger in NLTK. You can also re-tag the training data with a better tagger to improve the quality of your data. Using POS tags will affect your results dramatically.

You have only *train.txt* file to build you models. You can extract a small part as the development set or you can use cross validation. A sample from the data can be seen below:

```
Mr.          NNP    B-NP
Noriega      NNP    I-NP
was          VBD    B-VP
growing      VBG    I-VP
desperate    JJ     B-ADJP
.            .      O
```

4.3 Named Entity Recognition

The NER data contains four fields per token. The first and the last ones are the word and the entity tags. The second one is the POS tag and the third one is the chunk tag. Similarly to the chunking task you can use these as features in your models, however during the test phase only the words will be provided.

You can use *eng.train.txt* and *eng.testa.txt* files as training and development sections. A sample from the data can be seen below:

```
There           EX     B-NP    O
was             VBD    B-VP    O
no              DT     B-NP    O
Bundesbank      NNP    I-NP    B-ORG
intervention    NN     I-NP    O
```

5 Specifications

5.1 Implementation

1. You will implement 6 tagger classes in 6 modules:

Class	Module
TaggerLR_POS	tagger_lr_pos.py
TaggerLR_Chunk	tagger_lr_chunk.py
TaggerLR_NER	tagger_lr_ner.py
TaggerCRF_POS	tagger_crf_pos.py
TaggerCRF_Chunk	tagger_crf_chunk.py
TaggerCRF_NER	tagger_crf_ner.py

2. Each class will derive from the **Tagger** class in *tagger.py* module and have the methods below:
   train, test, evaluate, tag, tag_sents, save, load
   See Appendix A for details.

3. You are free to add other classes, modules and methods. Given source files and class hierarchy is for guidance only and designed to minimize your work.

4. The models share almost every method with minimal modifications (mainly on feature selection).

5. Do not forget that CRF and Logistic Regression differ in their views of data. In LR, all tokens are considered separate data samples. Each word is labelled individually even though some of them are in the same sentence. In CRF, a sequence is labelled as a single entity. These will effect feature extraction and tagging.

6. Submit the best performing taggers -with parameters and feature sets- in the 6 tagger classes.

7. Submit additional tests, experiments and supplementary materials in other files.

8. See provided test sections in source files for usage.

5.2 Report

1. Define and explain the following evaluation metrics. These are the same scores used to evaluate your taggers.

   - precision, recall, f-score, accuracy, confusion matrix
   - Use micro averaging while reporting multi-class system scores.
   - For the NER task, precision, recall, f-score scores should omit O tags. Accuracy and confusion matrix will not.
     - Notice the high percentage of O tags, and the importance of reporting only B and I tags.

2. Explain the following concepts

   - training, evaluation and test splits in the data, cross validation
   - learning curve, parameter optimization, over-fitting

3. Explain your feature selection process

- Rationale behind your features, why do you think they help your taggers?
- Plot the effects of adding different features, how does the accuracy change?
- Use 3-4 stages while adding different features and training your models. Report accuracies after adding word based, tag based, character based or position based features. One plot for each of the 3 tasks is sufficient.

4. How does data size effect your taggers?

   - Plot learning curves for a single tagger at 10% data intervals.

5. Analyze your taggers performance with the scores in the metrics above

   - Compare MaxEnt and CRF
   - Provide confusion matrices with heat map plots
   - Which classes are easy to classify, which ones are not?

6. Did you try to find better parameters for your classifiers? Report your parameter search results with plots.

6 Regulations

1. **Programming Language:** You will use Python.

2. You can use the libraries below:

   - nltk : data, preprocessing and maxent
   - scikit-learn : maxent (logistic regression), metrics, preprocessing, feature representation etc.
   - python-crfsuite : CRF, feature representation (and metrics)
   - sklearn-crfsuite : scikit-learn like interface for for python-crfsuite
   - numpy, scipy etc : data representation, arrays, numeric/scientific computation
   - matplotlib, pandas, seaborn: analysis, report and plots

3. **Late Submission:** You have 3 days of late submission that can be shared between all assignments.

4. **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations.

5. **R**emember that students of this course are bounded to code of honor and its violation is subject to severe punishment.

6. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

7 Submission

- Submission will be done via COW.
  Create a tar.gz file named `hw1.tar.gz` that contains all necessary files (including modules and log files, except saved models and data) and your report as a PDF file.

8 References

- A maximum entropy model for part-of-speech tagging, Adwait Ratnaparkhi, 1996
- Conference on Natural Language Learning shared tasks:
  `http://www.cnts.ua.ac.be/conll2000/chunking/`
  `http://www.cnts.ua.ac.be/conll2002/ner/`
  `https://www.clips.uantwerpen.be/conll2003/ner/`
- List of the state of the art methods:
  `https://aclweb.org/aclwiki/POS_Tagging_(State_of_the_art)`
  `https://aclweb.org/aclwiki/Named_Entity_Recognition_(State_of_the_art)`
  `https://aclweb.org/aclwiki/NP_Chunking_(State_of_the_art)`
- Documentation and download links for the libraries:
  - Natural Language Toolkit
    `https://www.nltk.org/`
  - scikit-learn
    `https://scikit-learn.org/`
    `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`
    `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html`
  - crfsuite
    `https://python-crfsuite.readthedocs.io/en/latest/`
    `https://sklearn-crfsuite.readthedocs.io/en/latest/index.html`

A Class Interfaces

A.1 Base Tagger Class

```
class Tagger():
    '''
    base class for all taggers
    '''

    def __init__(self, params=None):
        '''
        initialize the tagger (and the vectorizer if necessary)
        params is a dict of tagger arguments that can be passed to models
        '''
        raise NotImplementedError("__init__ not implemented")

    def train(self, file_name):
        '''
        extract the features and train the model
        file_name shows the path to the file with task specific format
        '''
        raise NotImplementedError("train not implemented")

    def test(self, file_name, labels_to_remove=[]):
        '''
        test the model, extract features and predict tags
        return metrics, confusion matrix and tagged data in the order below
        precision
        recall
        f1
        accuracy
        confusion matrix
        tagged data
        file_name shows the path to the file with task specific format
        labels_to_remove show classes to ignore
            while calculating precision, recall and f1 scores
        '''
        raise NotImplementedError("test not implemented")

    def evaluate(self, file_name, labels_to_remove=[]):
        '''
        return accuracy
        to be used for validation, learning curve, parameter optimization etc.
        '''
        raise NotImplementedError("evaluate not implemented")

    def tag(self, sentence):
        '''
        tag a single tokenized sentence
        sentence: [tkn1, tkn2, ... tknN]
        return the tagged sentence as list of fields
            as given in training/test files
        tagged sentence: [[tkn1, ..., tag1], [tkn2, ..., tag2], ...]
        put None for any field between token and tag that is not predicted
        '''
        raise NotImplementedError("tag not implemented")

    def tag_sents(self, sentences):
        '''
        tag a list of tokenized sentences
        sentences: [[tkn11, tkn12, ... tkn1N], [tkn21, tkn22, ... tkn2M] ...]
        return the tagged sentence as a list of fields
            as given in training/test files
        tagged sentences: [[[tkn11, ..., tag11], ...], ...]
        put None for any field between token and tag that is not predicted
        '''
        raise NotImplementedError("tag_sents not implemented")

    def save(self, file_name):
        '''
        save the trained models for tagger to file_name
        '''
        raise NotImplementedError("save not implemented")

    def load(self, file_name):
        '''
        load the trained models for tagger from file_name
        '''
        raise NotImplementedError("load not implemented")
```