THE UNIVERSITY OF SYDNEY

Named Entity Recognition

Due date: 5pm Monday of week 14 (2018-06-11)

This assignment is worth 20% of your final assessment.

In this assignment, you will build and evaluate a named entity recognition (NER) system via sequence tagging, **or** you will perform a systematic comparison of existing NER approaches.

3.0.1 Option I: Implementation

- Implement the *Viterbi algorithm* for predicting the best tag sequence according to a learnt model;
- Use this to construct a *Maximum Entropy Markov Model*;
- Explore possible feature sets and perform experiments comparing them;
- Evaluate the performance of your system on English and German;
- Describe your experiments, results and analysis in a report.

> **For Option I, your submission should include:**
>
> - **your report (~3 pages, not including tables/diagrams);**
> - **a zipfile containing your code and README instructions on how to run it. Please do not include the data, but assume the README directory contains the `conll03` subdirectory.**
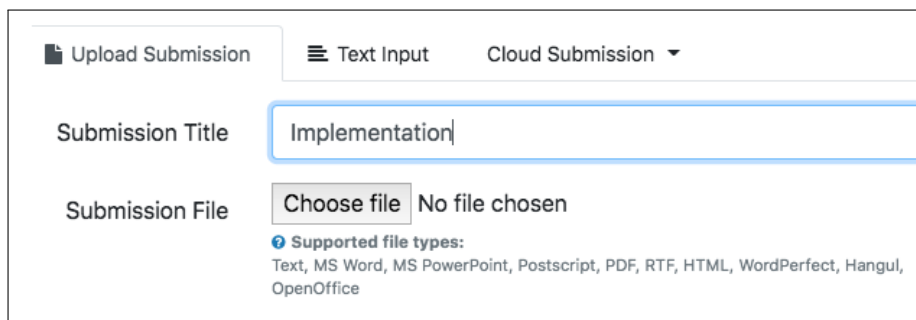
3.0.2 Option II: Application

- Find existing NER systems and apply them to given text;
- Critically compare NER system descriptions;
- Systematically analyse and compare the errors made by NER systems;
- Describe your experiments, results and analysis in a report.

> **For Option II, your submission should include:**
>
> - **your report (~4 pages, not including tables/diagrams).**
> - **a zipfile containing any code or notebooks used in analysis and README instructions on how to run it. Please include the `eng.testb` data tagged by each system.**

Reports should be submitted to Turnitin via Canvas, and code to the other Canvas assignment submission page, before the deadline. For the report, please set the Submission Title to begin with *Implementation* or *Application* depending on which option you took:

📄 Upload Submission	☰ Text Input	Cloud Submission ▾
Submission Title	Implementation	
Submission File	Choose file No file chosen	

ⓘ Supported file types:
Text, MS Word, MS PowerPoint, Postscript, PDF, RTF, HTML, WordPerfect, Hangul, OpenOffice

3.1 The Named Entity Recognition Task

3.1.1 Input

Input to the NER task consists of raw text, e.g.:

```
U.N. official Ekeus heads for Baghdad.
```

The typical preprocessing stack for NER includes sentence boundary detection, word tokenisation, part-of-speech tagging and syntactic chunking, e.g.:

```
U.N.         NNP  I-NP  I-ORG      ← POS          ← gold label NEG tag
official     NN   I-NP  O
Ekeus        NNP  I-NP  I-PER
heads        VBZ  I-VP  O
for          IN   I-PP  O
Baghdad      NNP  I-NP  I-LOC
.            .    O     O          ← syntactic chunking.
```

Here, each word has been put on a separate line (with a double newline between sentences). The first item on each line is a word, the second is a part-of-speech (POS) tag, the third is a syntactic chunk tag, and the fourth is the gold standard NER tag. You will receive a data set where this preprocessing has already been *automatically* performed.

3.1.2 Output

Output from the NER task consists of a tag for each word, which should be added as a new column, e.g.:

```
U.N.         NNP  I-NP  I-ORG  I-ORG
official     NN   I-NP  O      O
Ekeus        NNP  I-NP  I-PER  I-LOC
heads        VBZ  I-VP  O      O
for          IN   I-PP  O      O
Baghdad      NNP  I-NP  I-LOC  I-LOC
.            .    O     O      O
```

The chunk tags and the named entity tags have the format I-TYPE which means that the word is inside a phrase of type TYPE. Only if two phrases of the same type immediately follow each other, the first word of the second phrase will have tag B-TYPE to show that it starts a new phrase. *Note this differs a little from some of the lecture examples.* A word with tag O is not part of a phrase.

3.1.3 Data Set

The main dataset (eng) is a collection of newswire articles (1996-7) from Reuters, which was developed for the Computational Natural Language Learning (CONLL) 2003 shared task. It uses the typical set of four entity types: person (PER), organisation (ORG), location (LOC), miscellaneous (MISC). The second dataset (deu) is the German data from CONLL 2003, which has an extra column (the second) which holds the lemma (base form) of each word. Your system should be primarily developed for English, but also tested on German for comparison.

The prepared data can be downloaded from here. By downloading the data you agree 1) to only use it for this assignment, 2) to delete all copies after you are finished with the assignment, and 3) not to distribute it to anybody.

The data is split into subsets:

- training (eng.train) - to be used for training you system;
- development (eng.testa) - to be used for development experiments;
- held-out test (eng.testb) - to be used only once features and algorithms are finalised.

For further information, see `http://www.clips.ua.ac.be/conll2003/ner/`.

> **Do not download and build the data set from the above URL. It does not include the text.**

3.2 Maximum Entropy Markov Models for Sequence Tagging

3.2.1 Sequence Tagging

Our goal is to predict a tag sequence $t_1^* \dots t_n^*$ given an input sentence $w_1 \dots w_n$:

$$t_1^* \dots t_n^* = \underset{t_1 \dots t_n}{\operatorname{argmax}} P(t_1 \dots t_n | w_1 \dots w_n). \tag{3.1}$$

We formulate the probability of the tag sequence via the Markov assumption:

$$P(t_1 \dots t_n | w_1 \dots w_n) \approx \prod_{i=1}^{n} P(t_i | w_1 \dots w_n, t_{i-1}) = \prod_{i=1}^{n} P_{\text{MaxEnt}}(t_i | \mathbf{x}_i) \tag{3.2}$$

[handwritten: context → active attributes of w_i]

where each context $\mathbf{x}_i$ encodes the active attributes for the corresponding word w_i, and P_{MaxEnt} is obtained from a logistic regression / maximum entropy model that can assign a probability to each tag given $\mathbf{x}_i$.

Note: unlike in Assignment 2 where you used the hard predictions of Logistic Regression, here we use probabilistic predictions

[handwritten: → weights]

$$P_{\text{MaxEnt}}(t_i | \mathbf{x}_i) = \frac{\exp(\beta_{t_i} \cdot \mathbf{x}_i)}{\sum_{t'} \exp(\beta_{t'} \cdot \mathbf{x}_i)} \tag{3.3}$$

where the model is defined by weights (or coefficients) β. (We use β_t to refer to the vector of weights for class t.)

This probabilistic prediction can be obtained, for instance, with the `predict_proba` method in Scikit-learn, `predict(type="response")` in R's glm, `distributionsForInstances` in Weka, etc.

[handwritten: → features]

To encode tag history, you should include previous predictions in the attributes for each word. For example, you should include a history attribute for the previous prediction $KLASS_{i-1} = X$. This is in contrast to the HMM in Lecture 6 and Section 10.2.2 of Manning & Schütze, where there was an explicit *transition probability* term for the probability of a tag given the previous tag $p(t_i | t_{i-1})$.

3.2.2 Training the MaxEnt model

To train the MaxEnt model (i.e. estimate β in equations above), caluclate features for each token, including one or more tag history features, and use standard MaxEnt / logistic regression training to learn the association between each token's features and its class. That is, tokens and their context here take the place of documents and their content in Assignment 2.

3.2.3 The Viterbi Algorithm

We could evaluate Equation Equation 3.2 for all possible taggings $t_1 \dots t_n$ of a sentence of length n, but that would make tagging exponential in the length of the input. For the CONLL 2003 tag set ($\mathcal{T} = $ {B-LOC, I-LOC, B-PER, I-PER, B-ORG, I-ORG, B-MISC, I-MISC, O}), this results in a time complexity of $\mathbf{O}(9^n)$.

The Viterbi algorithm is a dynamic programming algorithm for finding the best sequence of tags in $\mathbf{O}(n \times T^2)$ time and $\mathbf{O}(n \times T)$. Thus, for the CONLL 2003 task, the Viterbi algorithm has a time complexity of $\mathbf{O}(n \times 9^2)$ and space complexity of $\mathbf{O}(n \times 9)$.

We use the following notation in this section:

$\mathcal{T}$	set of all tags
T	the number of tags
n	the number of tokens in the current sequence
i	an index over tokens
t_i	the tag selected for token i
c	a candidate tag for the current token
p	a candidate tag for the previous token
$\delta[i][c]$	*score* of the best tag sequence up to and including tag c at position i
$\psi[i][c]$	the tag p at the preceding position $i-1$ that gives $\delta[i][c]$ (i.e. the best preceding tag)
$\mathbf{x}_i^{(p)}$	the context for token number i assuming the best tag sequence such that token $i-1$ has tag p

For each test sentence, the Viterbi algorithm first requires filling in a $n \times T$ chart of sequence scores:

1. **for** $i \leftarrow 1$ **to** n **do**

2.   **for** $c \in \mathcal{T}$ **do**

3. $\delta[i][c] = \max_{p \in \mathcal{T}} \left(\delta[i-1][p] + P_{MaxEnt}(t_i = c \,|\, \mathbf{x}_i^{(p)}) \right)$

4. $\psi[i][c] = \operatorname*{argmax}_{p \in \mathcal{T}} \left(\delta[i-1][p] + P_{MaxEnt}(t_i = c \,|\, \mathbf{x}_i^{(p)}) \right)$

5.   **end**

6. **end**

Note you can calculate the max and the argmax in a single loop over each previous tag p. Note that the features and weights used in the score change to account for both the current tag c and the previous tag p, although most attributes may encode information from the observed context without regard for the tag context.

After the chart is filled in, the Viterbi algorithm works back through the chart to recover the predicted tag sequence:

7. $t_n^* = \operatorname*{argmax}_{p \in \mathcal{T}} \delta[n][p]$

8. **for** $i \leftarrow (n-1)$ **down to** 1 **do**

9. $t_i^* = \psi[i+1][t_{i+1}^*]$

10. **end**

11. **return** $t_1^* \dots t_n^*$

3.2.4   Advanced: Efficiently calculating $P_{MaxEnt}(t_i \,|\, \mathbf{x}_i^{(p)})$

> **You should only consider implementing this extension after a basic implementation of Viterbi is working. No additional marks will be awarded, but it may speed up experiment turnaround by making prediction faster.**

Viterbi involves repeatedly calculating the distribution $P(t|\mathbf{x}^{(p)})$ for some sample i, for each tag t and p. You need to calculate the probabilities for all t in order to calculate the probablity for any of them, because of the denominator in Equation 3.3.

You can also avoid entirely recalculating the probability for each p. Since most attributes contributing to the feature vector are identical regardless of the tag history, their contribution to score can be calculated, and then – while looping over the candidate previous tag – modified minimally to account for tag context.

That is, you can calculate the un-normalised score $\beta_t \cdot \mathbf{x}^{(-p)}$ for each tag t, where $\mathbf{x}^{(-p)}$ is the feature vector $\mathbf{x}$ modified so that features dependent on the tag history are set to 0. This score is the same for all p (for some $\mathbf{x}$); you can think of it as being like an emission probability in HMMs. You can separately calculate $\beta_t \cdot \mathbf{x}^{(+p)}$ for each p and t where $\mathbf{x}^{(+p)}$ has all features not dependent on tag history set to 0; if it only depends on p and t, it is independent of $\mathbf{x}$, like a transition probability in HMMs, and can be calculated once for each p and t. These two can be added together ($\beta_t \cdot \mathbf{x}^{(p)} = \beta_t \cdot \mathbf{x}^{(-p)} + \beta_t \cdot \mathbf{x}^{(+p)}$), exponentiated and normalised as per Equation 3.3 to get the required probabilities.

3.3   Typical NER Features

As mentioned above, you should encode tag history by including previous predictions in your attributes, e.g.:

Condition	Contextual predicate
$\forall w_i$	$\text{KLASS}_{i-1} = X$

Other typical attributes for NER include the words and parts of speech in a window of two tokens to either side of the current one, i.e.:

Condition	Contextual predicates
$\forall w_i$	$w_i = X$, $w_{i-1} = X$, $w_{i-2} = X$, $w_{i+1} = X$, $w_{i+2} = X$
$\forall w_i$	$\text{POS}_i = X$, $\text{POS}_{i-1} = X$, $\text{POS}_{i-2} = X$, $\text{POS}_{i+1} = X$, $\text{POS}_{i+2} = X$

3.3.1 Other Features

You should explore some other possible features for NER. For example, you might consider a few of the following questions:

- how could you capture the fact that words can be capitalized to be names but also due to other conventions (e.g. start of a sentence, month names, etc.)?
- how could you handle infrequent words?
- how could you incorporate external lists of common locations or common person names?
- how could you capture common word types (e.g. how capitalisation and punctuation appear) in a general way?
- how could you capture inflection and derivational morphemes?
- how could you model sentence-level information?

> **Hint: Lectures on sequence tagging and named entity recognition contain larger list of example NER features.**

Another challenging alternative would be to make your system handle a two word Viterbi history, rather than just the previous word.

3.4 Precision, Recall and F-score for Phrases

Precision, Recall and F-score are a family of metrics for measuring a gold standard set of results against a predicted set of results. But the set in NER is a set of phrases or mentions extracted, not merely token classifications.

Thus, calculating P, R and F for NER is *not* the same as just calculating the P, R and F of each token classification decision. For NER we take sets of (start offset, end offset, entity type) triples for each entity mention (by interpreting the IOB notation), and compare the sets between the predicted and gold standard annotations. (Token-wise performance would consider the set of (offset, class) pairs, but we're not interested in that.)

For example, overall precision is the number of entity mentions (i.e. phrases) with start, end and type matched between predicted and gold annotations divided by the number of entity mentions predicted.

Along with the data set, we have provided the conlleval perl script for scoring your predicted tag sequences against the gold standard. This script expects data on stdin in the format described above. The second-to-last column should contain the gold standard NER tag and the last column should contain your predicted NER tag.

3.5 Option I: Implementation

The assessment for Option I is about how well you can:

- implement Viterbi on top of probablistic MaxEnt predictions;
- implement and clearly/concisely describe basic features;
- implement and clearly/concisely describe additional features;
- devise and clearly/concisely describe sound experiments;
- devise and clearly/concisely describe insightful error analysis.

The assessment **is not** about the quality of your code. However, your code may be consulted to verify that it is original and consistent with your report.

3.5.1   Implementation

You are free to use a programming language of your choice to implement the assignment. You may use any implementation of logistic regression / maximum entropy with citation.

3.5.2   Report

The report should describe which features you included, and identify which types of features were most important for your classifier's accuracy. It should also characterise the kinds of errors the system made.

The report **should not** focus on your implementation of the Viterbi algorithm.

Features:

- What features does your tagger implement?

- Why are they important and what information do you expect them to capture?

- Are these new features or can you attribute them to the literature?

Experiments / results:

- Which features are most important?
  e.g., performance of different feature combinations
  e.g., subtractive analysis for many feature subsets

- What is the impact of sequence modelling?
  e.g., performance with and without Viterbi

- How does your system perform with respect to a simple baseline?
  e.g., majority class
  e.g., classification with only obvious features, for instance, words

- How does your system perform with respect to the best results reported at the shared task?

- How does your system perform training and testing on German?

- Which of your features are language-independent?

Error analysis:

- What are the typical tag confusions made by your system?

- What are the characteristic errors made by your system?
  e.g., manually characterise the errors on a sample of your predictions
  e.g., hypothesise and/or implement some possible solutions

- Are there problems or inconsistencies in the data set that affect your system's performance?

- You only need to do error analysis on English (unless you speak German!)

> **Although in general the choice of how to present your results is up to you, you *must* include micro-averaged Precision, Recall and F-Measure statistics. Development experiments should use the development data sub set and final results should be reported on the held-out test set.**

3.6   Option II: Application

The assessment for Option II is about how well you can:

- apply existing NER systems;

- clearly/concisely demonstrate understanding of features and models;

- devise and clearly/concisely describe sound experiments for analysing errors;

- report an insightful analysis of errors.

3.6.1 Task: NER System Comparison

Compare **two** publicly available NER systems to each other and to one of the CoNLL 2003 submissions.

1. Download a recent Named Entity Recognition system that you can train, or which includes a CoNLL 2003-style English NER model.

   - A system is recent enough if it has been developed or improved since 2008.
   - If using a pre-trained model, make sure that its outputs are using the same style of IOB notation as in our version of the CoNLL data.
   - Make sure you can find a technical description of the system, which you can use for step 6 below.
   - The system should be trained on (at least) the CoNLL 2003 `eng.train` data provided above.
   - The system should not be trained on `eng.testb`.

2. Use that model to label CoNLL's `eng.testb` data.

3. Use the `conlleval` script to quantify performance of the system.

4. Repeat from step 1 for a second system.

5. Select one of the top-performing systems from CoNLL 2003 (FIJZ, KSNM, ZJ, CMP and MMP all did well on both English and German). Read its system description paper from the CoNLL 2003 proceedings, and download its output. Output for CoNLL 2003 shared task systems can be found on the task web page (http://www.clips.ua.ac.be/conll2003/ner/) under the **CoNLL-2003 Shared Task Papers** heading. Follow the **system output** links.

6. Describe and compare the three system *descriptions*, focusing on: how does each incorporate information needed to perform NER well? what are key differences in how they represent the context?

7. Systematically *analyse errors* performed by each system on `eng.testb`, to answer: What kinds of errors do both systems make? What kinds of error does one system make, but others do not?

3.6.2 Report Structure

Your report should contain the following sections:

Systems Which systems are being analysed, and what publications are they described in? Summarise and *critically compare* the approaches. How are their approaches to NER similar or different? Do they capture the same or different contextual cues (e.g. features)? Do they model the same kinds of information in a different way?

Results Use `conlleval` to report performance on `eng.testb` and remark briefly on key results.

Method How did you systematically compare the outputs of different systems? (Don't give results here.)

Analysis Report your analysis of kinds of error. Think broadly when considering *kinds of errors*, and make sure you refer to examples from the data. The quantitative analysis gives precision and recall for each class at the level of matched phraes. It does not, however, tell you whether there were *boundary discrepancies* (where the tag was agreed, but a word was included or excluded relative to the other system or gold standard), or whether there were *tag discrepancies* despite mention start and end being agreed. It does not tell you which entity types or token-level tags were confused by a system; nor whether some part of speech sequences are particulary troublesome. Nothman et al. (EACL 2009) sections 3–4 might provide some ideas about systematic comparisons of NER taggings, though your approach need not be as sophisticated.

Discussion Consider at least one of the following questions, or your own. Please write the question you are answering. You do not need to answer all questions; it is better to answer one well:

   - Are some kinds of errors more problematic than others in application?
   - Can you explain differences in the systems' errors in terms of how their models (features etc.) differ?
   - Are there problems or inconsistencies in the data set that affect your system's performance?