CS4740/5740 Introduction to NLP
Fall 2016
Finding Uncertainty With Sequence Tagging

Proposal: due via CMS by Monday, 10/03 11:59pm
Final report: due via CMS by Friday, 10/21 11:59pm

1 Overview

For this project, your task is to **detect uncertain sentences**, i.e., sentences that express some sort of uncertainty or hedging, in Wikipedia articles. On top of that, we want you to pin-point the *exact fragment* of the sentence that triggers the uncertainty. As many of you noted in your critiques, this is important in the context of *explainable machine learning*: if we expect humans (or companies) to make a decision based on our models, we better be able to show them *why* our model thinks what it does!

You should tackle this as a **sequence tagging** task: predicting the spans of uncertain words and phrases within a text. You may use HMMs, CRFs, MEMMs, structured perceptrons, or other approaches. You may use existing packages or implement your own sequence tagger. If you implement your own, we recommend the HMM for simplicity. If you opt to use an existing implementation, you are expected to perform more extensive experiments.

2 Task and Dataset

You are given a modified version of the Wikipedia Weasel Words dataset.[1] The training data is organized into 1186 text documents, each consisting of a different Wikipedia article. Tokenization and part-of-speech tagging has been performed by us, to help you out.[2] You are also given 1000 test documents, without labels: this is what your Kaggle score will be calculated on.

Uncertain phrases or **weasel phrases** are markers of non-factuality or speculation. They should be avoided in writing, especially in an academic setting. (If your system works well, you could even think about running it on your own report!) Here is an example from the training dataset:

[1] Usage of this dataset is restricted to this class project. If you are interested in using it further for other work, please contact us.

[2] We used `https://spacy.io`.

He **is considered** *to be* **one of the leading** *British potters of his generation.*

This rather poorly-written sentence contains *two* weasel phrases, "is considered" and "one of the leading". In your training data, this sentence can be found in the form:

```
He          PRP     _
is          VBZ     CUE-1
considered  VBN     CUE-1
to          TO      _
be          VB      _
one         CD      CUE-2
of          IN      CUE-2
the         DT      CUE-2
leading     VBG     CUE-2
British     JJ      _
potters     NNS     _
of          IN      _
his         PRP$    _
generation  NN      _
.           .       _
```

This is a three-column tab-separated format. The first column is the word token, the second column is the part of speech tag, and the third column signals the uncertainty cue phrases. Important things to note about the data format:

1. Sentences are separated by empty lines.
2. Uncertainty cues are tagged with an incrementing ID (i.e. `CUE-1` instead of just `CUE`). This is done to eliminate ambiguity when two different cue spans are adjacent.

The *test documents* are in a similar format, but **without the third column**.

Directory structure. The data archive contains three subdirectories: `train`, `test-public` and `test-private`. The `train` directory contains 1187 documents with complete labels. The two `test-*` directories contain **unlabeled documents** (500 each) that you will run your prediction systems on, and submit the results to Kaggle.

Why are there two test directories? Simple: there are **two Kaggle leaderboards!** While the competition is still going, the public Kaggle leaderboard will report your BIO F1-score on the `test-public` data. Your final score, for fairness, will be based on the `test-private` dataset. This is because everytime you submit to Kaggle and see the public score, you learn some information about the hidden data (for example, which models work better.) But tweaking your model *too much* using the leaderboard can lead to *overfitting*. Since the `test-private` labels are never revealed until the very end of the competition, this makes the evaluation more fair. You can find exact instructions on how to submit in the Kaggle section below.

3 Overview

Here's a "helicopter picture" of the steps you are supposed to go through for this assignment.

Preprocessing. Sequence tagging often works better when the spans are annotated with a tag format like BIO (Beginning, Inside, Outside) or BILOU (Beginning, Inside, Last, Out, Unique). You have to convert the `CUE-n` labels to one such representation. Another reason not to use the provided encoding: machine learning models will treat `CUE-1` and `CUE-3` as completely different, rather than intelligently sharing information between them.

Experiment design. Think about what machine learning model you want to use, and whether you want to implement it or use an existing implementation. If using a model that supports the inclusion of "features" (the attribute-value pairs discussed in class and in the readings), think about what feature templates (i.e., what *types* of features) you could use. Think about what parameters you will have to tune on a held-out validation set. Think about possible baselines you can compare against (e.g., making a list of *hedge* words like "should", "could", and only predicting those words as cues.)

Training and evaluation. You will put in practice what you planned in the previous step, and choose one or a few promising model and/or feature configurations.

Error analysis and post-processing. Before uploading test results to Kaggle, you will need to convert the labels again: Remember, the end-goal is not to have some out-of-context word-level predictions: it is to find *uncertain phrases and sentences.* To submit to Kaggle, you will need to convert your labels back to some binary predictions — in one Kaggle competition, you will need to make a binary prediction for *each word*; in a separate Kaggle competition, you will need to make an overall prediction for each sentence (i.e., does the sentence express uncertainty or not). For example, if your model predicts at least one uncertain span inside a sentence, you might give that sentence a positive label. You are also strongly encouraged to examine some failed predictions on your development set (i.e., false positives and false negatives) to get a sense of how your model could be improved.

Rinse and repeat :). Your error analysis or Kaggle score might help you come up with some ways to improve your performance.

4 Implementation

You may use existing software for sequence tagging, or implement a sequence tagger yourself. If you decide to use an existing package/toolkit rather than implement a sequence tagger from scratch, you are expected to include more extensive experiments.

1. If you decide to implement the sequence tagging algorithm yourself, we strongly suggest implementing the **HMM** rather than more complicated discriminative models unless you have sufficient background knowledge. You may use any programming language that you'd like and may make use of helper libraries. If you implement the model yourself, make sure to describe all necessary details clearly in your report.

2. If using an existing sequence tagging toolkit or package to run your experiments, it is up to you to figure out how to use the packages. **In both cases, you will need to think up front about what kind of experiments would be interesting to run given your choice of algorithm.**

3. Similar to the previous project, you should reserve some portion of the training dataset for validation purposes, e.g., to decide which model to use or which features to include.

4. **Develop baseline systems to be compared with your own model**. You are required to implement baseline systems for comparison. One simple option is to first build a lexicon of uncertainty-marking words, like *may* or *probably*, either manually or from the training data. Then implement a simple system that tags all (and only) occurrences of these words as uncertainty cues. You must compare your proposed model with the baseline(s) and describe the differences, advantages and disadvantages in your report.

5 Kaggle Competitions

For this project you will enter **two** Kaggle competitions! In one, you will be evaluated on locating uncertain phrases, while in the other you will have to label entire setences as certain or uncertain.

For the second competition, you should use the output of the same system as the first competition: for example, if a sentence contains an uncertain phrase, then label it as uncertain. But you can think of other extensions to improve the sentence-level performance.

Include the final evaluation results in your final report before the Kaggle competition ends. Note that once the Kaggle competition closes, you cannot make additional submissions to Kaggle.

5.1 Uncertain phrase detection

For this competition,[3] you will submit **predicted spans**: for each uncertainty cue span you detect, submit its range `start-end` in terms of token indices. The ranges should be separated by a space.

For example, if the beginning of the public test data (complete with your model's predictions) is:

[3] `https://inclass.kaggle.com/c/uncertain-phrase-detection-cs4740-project-2-1`

```
I       PRP O
like    VBP O
pie     NN  O
.       .   O

I       PRP B-CUE
think   VBP I-CUE
you     PRP O
also    RB  O
do      VBP O
,       .   O
maybe   RB  B-CUE
.       .   O

...
```

then your Kaggle submission file should start with:

```
Type,Spans
CUE-public,4-5 10 ...
CUE-private,...
```

Your submission file should contain three lines lines: one header and a prediction line for each of the test folders provided. (E.g., `CUE-public` corresponds to the `test-public` folder.) For each folder, token ids start at 0, and are counted in lexicographic (alphanumeric) order of the filenames. Empty lines are not counted. To double-check your counting: the highest token id in the public folder is 55758, respectively 55663 in the private folder.

5.2 Uncertain sentence detection.

In this competition[4] your performance is evaluated at *sentence* level. So for the same example from above, the corresponding start of the Kaggle submission file would be (depending on your model's prediction):

```
Type,Indices
SENTENCE-public,1 ...
SENTENCE-private,...
```

This time, the output consists of space-separated **sentence ids** (so it goes from 0 to 2007, and to 2004, for the `test-public` and `test-private` data folders, respectively). Here, we predicted that the second sentence is uncertain (the one with id 1, counting from 0) because our system found two uncertain phrase inside that sentence. But you may use other rules (maybe based on the confidence of your sequence tagger, if available).

[4] `https://inclass.kaggle.com/c/uncertain-sentence-detection-cs4740-project-2-2`

Note: consecutive sentences should **not** be grouped into ranges: output `... 3 5 6 7 9 ...` instead of `... 3 5-7 9 ...`. This is different than the token-level submission.

6 Extensions

Here are several extensions you can implement and experiment with. **Doing at least one extension is mandatory.** Having more than one extension may be counted as bonus points with respect to the degree of your implementation, experimental results and write-up. **Note that all the extensions can be used in the Kaggle competition**.

1. If using a feature-based model (structured perceptron, CRF, etc.), experiment with different features and combinations of features. You can get creative and use external resources, e.g. lists of *weasel words* and *hedge words* from the Internet. Remember to thoroughly evaluate by comparing the performance not only with the baseline, but also with a simple feature set (say, bigrams and part-of-speech tags).

2. Most tokens in the dataset are **not** uncertainty cues, and many sentences don't have any uncertainty cues at all. This means the dataset is *imbalanced*, and your machine learning model might be inclined to predict `O`-tags more often than necessary because of this. Research ways to mitigate this. For instance, resampling methods. You could downsample the negative class by removing training sentences with no uncertainty cues (but this also removes precious training data.) You could upsample or reweight the positive class by repeating sentences with uncertainty. Again, remember to evaluate and compare the impact thoroughly.

3. If you're using toolkits, you might compare one sequence-tagging method with another, and vary the parameters and/or feature sets to see how they affect the performance of different methods.

4. Implement a secondary sequence tagging system that is different from your primary implementation, e.g. MEMMs.

7 Proposal

Describe your sequence-tagging system and implementation plan in 1 page. You should consider

- Which model are you planning to implement? Why? Briefly describe the model and its properties that led you to use it.

- Explain the algorithmic key points of your model. Especially think about which are hidden variables (e.g. tags) and observed variables (e.g. tokens in the input sentence) for our setting, and what are the corresponding model parameters.

- If using a feature-based model (structured perceptron, CRF, etc.), brainstorm which features you would incorporate to learn emission probabilities. Support your design decisions based on the real examples given in the dataset.
- State which extension you are planning to do. While you might end up implementing different extensions, it will help us to provide you feedback.

In addition to submitting a 1-page proposal document, you must also submit code for at least one baseline system and report its performance on Kaggle. If you split the training set into training and validation parts, include the details of how you did this in the report.

8 Report

You will submit a short document (5-6 pages will suffice) that contains the following sections. (You can include additional sections if you wish.) All the guidelines and feedback from Project 1 apply here as well: be concise and try to communicate well!

1. **Sequence Tagging Model**

   (a) **Implementation details.** Make clear which sequence tagging method(s) you selected. Make clear which parts were implemented from scratch vs. obtained via an existing package. Explain and motivate any design choices providing the intuition behind them.

   (b) **Pre- and post-processing.** Explain and motivate the pre-processing steps you apply to the data.

   (c) **Experiments.** Describe the motivations and methodology of the experiments that you ran. Clearly state what were your hypotheses/expectations; try to explain why they were or were not confirmed by the experiments.

   (d) **Results.** Summarize the performance of your system and any variations that you experimented with on both the training/validation and test dataset. **Note that you have to compare your own system to at least one other non-trivial baseline system.** Put the results into clearly labeled tables or diagrams and include your observations and analysis. An error analysis is required— what sorts of errors occurred, why? When did the system work well, when did it fail and any ideas as to why? How might you improve the system?

   (e) **Competition scores.** Include your team name and the screenshot of your best score from Kaggle.

2. **Extensions**

   Explain the extensions that you decided to do. Include the implementation details, the experiments, and the results similar to the previous section. If your extensions contribute to the competition score, analyze why they help. If not, try to explain why it was not useful.

3. **Contribution of team members**

   Briefly explain the contribution of an individual team member. You could report if working loads are unfairly distributed.

9 Grading Guide

- (10 pts) Proposal, clarity and feasibility of your plan.
- (40 pts) Design and implementation of the sequence tagging system if you implement the models yourselves; Design, thoughtfulness and comprehensiveness of your experiments if you choose to use existing toolkits.
- (35 pts) Report: clarity and quality of the report.
- (10 pts) At least one extension: implementation, experiments and discussion.
- (5 pts) **Submission to Kaggle.** (not optional!)

9.1 Things to avoid

Don't be ridiculously inefficient. You are not supposed to spend too much time optimizing your code, but it SHOULD NOT take forever either. The Viterbi algorithm is $O(sm^2)$ where s is the length of the sentence and m is the number of tags. Your implementation should have similar efficiency. Similar reasoning applies for memory efficiency: **use sparse structures** (e.g. dictionaries) whenever applicable, just like in Project 1.

10 What to Submit

10.1 Part One

- Proposal (one-page pdf file), **including your Kaggle team name.**
- Code and results for at least one baseline system. Include a brief description of the baseline.
- Upload the above to CMS. (due Monday, 10/03 11:59pm)

10.2 Part Two

- Source code with adequate comments (only include code that you wrote yourselves, DO NOT include code from existing toolkits.)
- Prediction output (the files you submitted to Kaggle)
- Report (pdf file), **including your Kaggle team name.**
- Upload the above to CMS. (due Friday, 10/21 11:59pm)