CS 4740 Introduction to NLP
Spring 2014
Sequence Tagging and its Application to Sentiment Classification

Proposal due via CMS by Tue, Apr 15, 1:25pm
Hardcopy due in class by Tue, Apr 15
Results and Report: Due via CMS by Mon, Apr 28 11:59pm
Hardcopy due in class by Tue, Apr 29

1 Introduction

In this project, you are to implement a hidden Markov model or experiment with a package/tookit that implements one or more of the sequence-tagging algorithms covered in class: HMMs, Maximum Entropy Markov Models (MEMMs) or even conditional random fields (CRFs).

Sequence-tagging methods have been successfully applied to many NLP tasks, including Part-Of-Speech (POS) tagging and Named Entity Recognition (NER). In this project, you will be applying the sequence-tagging method of your choice to **sentence-level sentiment classification**.

2 Task and Dataset

The task in this project is to classify sentiment at the sentence level. Sentiment classification has been studied extensively at the document level, e.g. determining the overall sentiment (e.g. *positive, negative* of a movie review. Recently, more research effort has been invested in sentiment analysis at the fine-grained level (e.g. sentence- or phrase- level). Classifying sentiment at the sentence level is a challenging task, since sentences contain less rich information than documents. Sequence tagging methods are particularly suitable for the task since it is able to model the sentiment transition structures within the document.

1. We will provide an Amazon review dataset, which consists of 294 reviews from 5 different domains: books, music, electronics, dvds, and video games. We split the dataset into training and test: the training set contains 196 reviews and the test set contains 98 reviews.

2. Each review starts with a title (e.g. books_neg_2) as the first line. The main text of the review starts from the second line. Each line corresponds to one sentence, and it contains two columns: the first column is the sentiment label of the sentence (*positive, negative, neutral*) and the second column is the sentence.

3. Reviews are separated by a blank line.

3 Implementation

The sentiment classification task can be formulated as a sequence tagging problem, where each review is treated as a sequence of sentences, and each sentence is tagged with a sentiment label. You are expected to implement a HMM or use an existing package/toolkit to train a sequence tagger using the training data, and use the sequence tagger to assign a sequence of sentiment labels to a sequence of sentences in an unseen review.

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. As usual, you will write up your findings in a short 5-6pg report.

1. If implement the sequence tagging algorithm yourself. We very strongly suggest implementing HMMs rather than MEMMs. If you decide to try both, start with HMMs. You may use any programming language that you'd like and any preprocessing tools that you can find. It might be possible, for example, to use a toolkit just to extract n-gram information, but to write the HMM code by yourself. If you decide to implement a CRF, clarify how you will do this in the proposal.

2. If using an existing HMM/CRF toolkit or package to run your experiments, it is up to you to figure out how to use the packages. Some suggestions are available from the course home page. **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, it could be beneficial to reserve some portion of the training dataset for validation purpose. Describe the details of your experimental designs in the report.

4. **Develop baseline systems to be compared with your own model.** In addition to the trivial baselines such as random guess, you can implement any baseline systems. One simple option is to identify certain words frequently used for each sentiment class, and to perform the prediction purely based on whether a test sentence includes those words. Note that baseline systems should be compared to your system in the report.

4 Kaggle

We will launch a Kaggle competition for the task. Your submission to Kaggle should be a single column file. Each line in the file contains an integer value: $\{1, 0, -1\}$, where 1 denotes *positive*, 0 denotes *neutral*, and -1 denotes *negative*. Suppose the testing input contains two reviews as follows:

review$_1$
...
...
...

$review_2$
...
...
...
...

Your output should only contain 7 lines where each line contains the prediction value for a sentence in the order of its appearance in the test file. When you upload your predictions you will see the evaluation results on a subset of the test data. You will be able to see your score on the full test set after the submission deadline.

You should include the final evaluation result in your final report before the Kaggle competition ends. Note that once the Kaggle competition closes, you cannot make additional submissions to Kaggle. The gold standard predictions for the test set will be released via CMS at the close of the competition. These could be of use as you analyze system performance or characterize the kinds of errors that your system made for inclusion in the project report.

5 Required Extension

Here are several extensions you can implement and experiment with. **Doing at least one extension is mandatory. Having more than one extension will 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 for Kaggle competition**.

1. Experiment with different sets of n-gram features: will bigrams be adequate or will trigrams (or 4-grams, etc.) be better?

2. Experiment with different smoothing methods: what smoothing method will you employ? How do different smoothing methods affect the results?

3. If you're using toolkits, you might compare one sequence-tagging method with another, and try vary the feature settings 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.

5. Improve your sentence-level prediction task by incorporating document-level sentiments (the labels can be read from the title of the review, for example, a review with title "books_neg_2" has a negative sentiment). Discuss how the incorporation of document-level information affects the prediction of sentence-level sentiment.

6 Proposal

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

- Which model are you planning to implement? (If you try to implement CRF, briefly explain your preparation for understanding the model since we did not cover it in class.)
- Explain the algorithmic key points of your model. Especially think about which are hidden variables & observed variables for our setting, and what are the corresponding model parameters.
- 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 are you planning to do. While you might end up implementing different extensions, it will help us to provide you feedback.

7 Report

You should submit a short document (5-6 pages will suffice) that contains the following sections. (You can include additional sections if you wish.):

1. **Problem Setting**
   (a) "Configuration" Clearly describe the parts you have implemented on your own and the packages you used from other libraries. Explain any non-trivial design decisions with providing the intuition.
   (b) "Data Processing" Explain and motivate the pre-processing steps you apply to the data.
2. **Sequence Tagging Model**

(a) "Implementation Details" Make clear which sequence tagging method(s) that you selected. Make clear which parts were implemented from scratch vs. obtained via an existing package. Explain and motivate any design choices.

(b) "Experiments" Describe the motivations and procedures of the experiments that you ran. Clearly state what were your hypotheses and what were your expectations.

(c) "Results" Summarize the performance of your system on both the training 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 – e.g. what sorts of errors occurred, why? When did the system work well, when did it fail, why? How might you improve the system?

(d) "Competition Score" Include your team name and the screenshot of your best score from Kaggle.

3. **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, explain why it was not useful.

4. **Individual Member Contribution**
   Briefly explain the contribution of an individual group member. You could report if working loads are unfairly distributed.

8 Grading Guide

- Proposal: [10 pts]
- Implementation: [50 pts]
- Report: [40 pts]
- Required Extension: [10 pts]
- Optional Extensions: [10 pts]

8.1 Things to avoid

- Don't be ridiculously inefficient. You are not supposed to spend time optimizing your codes, but it SHOULD NOT take forever either. Bigram Viterbi is $O(sm^2)$ where s is the length of the sentence and m is the number of tags. Your implementation should have this property.

9 What to submit

- Proposal (pdf file into CMS, **hardcopy** at class)
- Source code (only the code that YOU wrote, not for any of the packages that you used).
- Prediction output (the csv files you submit to Kaggle)
- The report (pdf file into CMS, **hardcopy** submission at class)
- Archive all of these except the proposal, and upload it to CMS.