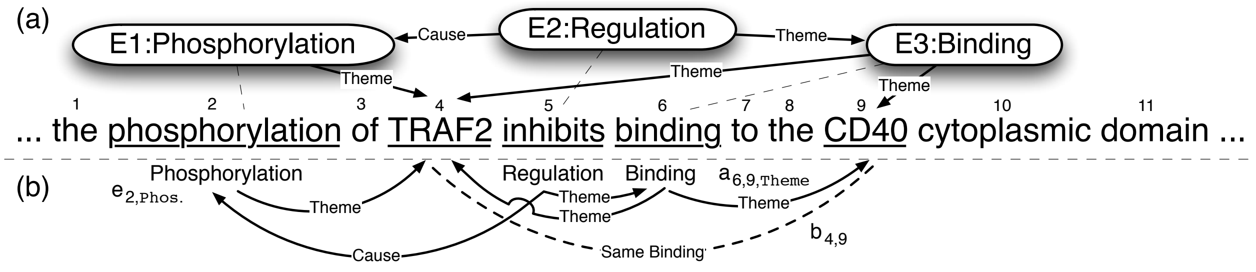
Assignment 2: Event Extraction

Sebastian Riedel
Due: Sunday 7th December

November 11, 2014

1 Introduction

Every day more than 2000 publications are added to the PubMed database of biomedical papers. Leveraging this large body of knowledge is becoming increasingly difficult for scientists, and this slows down progress we make in understanding biology, finding new drugs etc. Pathway databases such as Kegg, EcoCyc and MetaCyc store structured representations of known biomedical processes, and make the scientist's life easier, but are expensive to create and suffer from low coverage. This has led to major efforts in the automatic constructions of such knowledge bases from natural language text. In particular, since 2009 a bi-yearly BioNLP Shared Task on "biomedical event extraction" has been held. In this task sentences are mapped to structured representations of the biomedical events they describe [2, 3]. The (a) part of the following figure shows the type of structure we are supposed to extract.



In the lower (b) part of the figure you see an alternative representation of this structure. Here you have

- labelled tokens (such as "Regulation") indicating whether the token is a *event trigger* (a word that expresses a particular event).
- labelled edges between tokens, indicating whether the event grounded at the source trigger word has as argument an event or protein at the target word of the edge, and what the label of this argument is (e.g. Theme).

In this assignment we will restrict us to predicting this alternative "labelled token graph" representation of the task, and will consider only the 2011 competition and the "Genia track". For more information on this competition please refer to the shared task overview paper [3].

The assignment will cover a few aspects of the course:

- Generative vs Discriminative training and modeling
- Feature engineering
- Structured Prediction

1.1 Rules

You are expected to work in **teams of up to 4 people**. You will write one joint report, and you are free to allocate responsibility as you see fit. Again, you can use any programming language you like. There will be some Scala helper code to load the data files into Wolfe data structures.

One of main learning goals of this module is for you to not just re-use existing models and learners, but to be able to design and implement your own. This assignment will therefore require you to implement both a very simple generative and discriminative learning algorithm. Do not use existing learning packages for this task (of course you can use libraries for all other kinds of things you want to do, like JSON processing). To minimize less related coding you have to do, we provide preprocessed data for you (tokenization, parsing, tagging etc.) in JSON format.

The general plagiarism rules apply.

2 Data

The training data for the task can be found here at
`https://www.dropbox.com/s/bo4d6gfmenjojxj/bionlp2011genia-train-clean.tar.gz?dl=0`.

The data comes in JSON format, where each JSON file contains a document that specifies the source text and a list of tokens with information about character offsets within the source, tags, stems etc. It also contains a list of syntactic dependencies that can be useful as features. There are also protein mentions with their labels and token offsets on a per sentence basis. Finally, and most importantly, it contains a list of candidate event triggers, each with a list of candidate argument edges. The true triggers and arguments are labelled with their gold labels, the other ones with None. Your task is to predict the true labels for these trigger and argument candidates.

For evaluating your models in your report you will need to split off a development set from the training set. You will also be given a test dataset, but it will not contain the true labels. You will need to predict triggers and arguments for this test set, and submit your predictions with your report. You won't be graded based on your results on the test set, but we will use them to validate your reported results.

3 Background

3.1 Naive Bayes

This assignment requires knowledge of Naive Bayes (NB). We have covered Naive Bayes-like models when discussing the noisy channel model. The Naive Bayes model is a generative model in which first a class c is generated and then, conditioned on this class, a set of features or observations. For example, say you want to predict whether a token $\mathbf{x}$ has event label c, and you want to take into account the word $w_{\mathbf{x}}$ of the token and its tag $t_{\mathbf{x}}$. NB models the joint probability of token and class

$$p(\mathbf{x}, c) = p(c) \times p_w(w_{\mathbf{x}}|c) \times p_t(t_{\mathbf{x}}|c)$$

or more generally $p(c)\prod_f p_f(f(\mathbf{x})|c)$ where $f(\mathbf{x})$ is some property of the input (such as the word at token $\mathbf{x}$), and we have a different categorical distribution for each type of observation. The NB model can be trained using MLE, and you know how to do this.

3.2 Loglinear Models and the Perceptron

Loglinear models are defined through a set of feature functions $\phi_i(\mathbf{x}, c)$ (or $\phi_i(\mathbf{x}, \mathbf{y})$ for structured predictions $\mathbf{y}$). These feature functions are often binary, but don't have to be. One example feature function is

$$\phi_i(\mathbf{x}, c) = \begin{cases} 1 & \text{if } w_{\mathbf{x}} = \mathit{inhibits} \wedge c = \mathit{Regulation} \\ 0 & \mathit{otherwise} \end{cases} \tag{1}$$

The loglinear model then defines the conditional probability

$$p(c|\mathbf{x}) = \frac{1}{Z_{\mathbf{x}}} \exp\left[\sum_i \lambda_i \phi_i(\mathbf{x}, c)\right] = \frac{1}{Z_{\mathbf{x}}} \exp\langle \boldsymbol{\phi}(\mathbf{x}, c), \boldsymbol{\lambda}\rangle \tag{2}$$

where $Z_{\mathbf{x}}$ is a normalisation constant. Notably, the conditional model does not need to make any independence assumptions among features (it does not model the distribution over inputs at all).

A simple way to train the log-linear model is the perceptron algorithm [1]:

1. Initialize weights $\boldsymbol{\lambda}$
2. **for** epoch e in $1..K$

```
(a) for instance $(\mathbf{x}_i, c_i)$ in training set
    i. find the current solution $\hat{c} \leftarrow \arg\max_c p(c|\mathbf{x})$
    ii. if $\hat{c} \neq c_i$ change weights: $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \alpha\boldsymbol{\phi}(\mathbf{x}_i, c_i) - \boldsymbol{\phi}(\mathbf{x}_i, \hat{c})$
```

Crucially, you can use the perceptron also to train structured models where your set of possible solution is not just a set of labels, but, say, the set of all possible parse trees etc. The difficulty in this scenario is the argmax in algorithm above.

As mentioned in class, in practice it is very useful to average learned parameters from each instance and epoch when doing classification. This is called the "averaged perceptron" and is described in the original perceptron paper [1] in section 2.5.

3.3 Coding Hint

It's very useful to introduce a FeatureVector class to represent the feature vectors $\boldsymbol{\phi}(\mathbf{x}, c)$ and weights $\boldsymbol{\lambda}$. You would find such a class in almost all NLP libraries. One easy and convenient, but not necessarily most efficient, way of doing this are HashMap[String,Double] objects. In Scala (you can translate this to your favorite language) you can create a feature vector where $\phi_i(\mathbf{x}, c)$ from equation 1 is active as follows:

```
val vector = new HashMap[String,Double]
vector("w:inhibits c:Regulation") = 1.0
```

For a given token $\mathbf{x}$ and label c you can generically add the correct binary feature simply by

```
vector("w:%s c:%s".format(word(w),c)) = 1.0
```

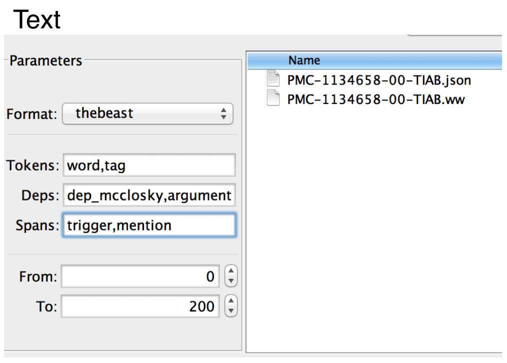
This creates a feature vector where only one feature active: the one consistent with token $\mathbf{x}$ and label c.

3.4 Evaluation

You will report, for triggers and arguments, the precision, recall and F1 measures. Precision in this case is the number of correct predictions you made that are not None, divided by the total number of predictions that are not None. Recall is the number predictions of correct predictions that are not None, divided by the total number of gold labels that are not None. F1 measure is the harmonic mean of precision and recall.

4 Visualization

It's generally very useful to visualize the data you are working with. The data archive comes with "ww" files that can be visualized with "What's wrong with My NLP?" (`https://code.google.com/p/whatswrong/`). To use these files install whatswrong, and then load a gold corpus. You need to use "thebeast" format, and configure the loader according to this screenshot:



PMC-1134658-00-TIAB.json
PMC-1134658-00-TIAB.ww

Format: thebeast
Tokens: word,tag
Deps: dep_mcclosky,argument
Spans: trigger,mention
From: 0
To: 200

5 Assignments

Problem 1: Naive Bayes (30 pts)

Build Naive Bayes Trigger and Argument Extractors. These should be classifiers that consider one trigger or argument at a time and predict their label, including "None". Add features to the model and evaluate their performance using precision, recall and F1 measures on the development set you choose. You will get 20 points if you implement the NB model with three features of your choice, and describe your results and choices well in your report. Further points are given for additional features. You can also read about other systems that participated in the BioNLP shared tasks to get inspiration for features.

Notice that for arguments there is a large number of candidates with the NONE label. These may affect your learner and slow it down, so depending on the scalability of your implementation you may need to sub-sample the negative data.

Problem 2: Perceptron (30 pts)

Develop two (log)linear models, one for trigger and one for argument extraction, and train models using the perceptron algorithm. Again you will get 20 points if you implement the log-linear model with three features of your choice (can be the ones from Problem 1), and describe your results and choices well in your report. Further points are given for additional features.

Problem 3: Joint Perceptron (20 pts, no code required)

Develop a joint model of trigger and argument extraction. By joint model we mean a model that assesses the quality of a trigger label jointly with the quality of its argument extractions. In the most basic joint model the score of a trigger-argument structures factorizes into the sum of scores for trigger label and all argument labels. More formally, for a given token $\mathbf{x}$ and candidate argument tokens $c_1 \ldots c_m$ let e be the label of the trigger, and $\mathbf{a} = a_{c_1}, \ldots, a_{c_m}$ the labels of the argument candidates. Then the fully factorized model scores this "frame" of e and $\mathbf{a}$ using $s(e, \mathbf{a}; \mathbf{x}, \boldsymbol{\lambda}) = s(e; \mathbf{x}, \boldsymbol{\lambda}) + \sum_j s\left(a_{c_j}; \mathbf{x}, \boldsymbol{\lambda}\right)$. Here the scoring functions themselves are dot products of feature vectors and weight vectors, as in equation 2.

To train such a model using the perceptron algorithm you need to define the search routine in its inner loop. How would the search routine look like for this joint model?

The next possible extension is a model that is still factorized in the same way, but that limits its set of "legal solutions" to assignments that fulfill the following constraints:

- A trigger can only have arguments if its own label is not NONE
- A trigger with a label other than NONE must have at least one THEME
- Only regulation events can have CAUSE arguments

In perceptron training you again need to solve the argmax problem in the inner loop of the training algorithm.

Your goal in this problem is to propose two search algorithm: one for efficiently finding the argmax in the unconstrained joint model, and one for finding the argmax for the constrained model. We expect the algorithms in pseudo-code, together with a description of what it does.

Problem 4: Implementation for Problem 3 (20 pts, optional)

Implement the fully factorized but constrained joint model from above. This means you have to: implement the argmax function, and then train the model jointly using the perceptron algorithm. Compare against the isolated models in per-trigger and per-argument evaluation.

Problem 5: Joint Conditional Likelihood (25 pts, no code required, optional)

Say we want to train the joint model by maximizing the conditional likelihood of the data (equivalent to Maximum Entropy Training) and use Stochastic Gradient Descent/Ascent. What is the gradient? How can you efficiently compute it?

Problem 6: Error Analysis (20 pts)

Choose the best model you have developed so far (can be NB, perceptron or the joint model) and analyse the errors it makes. For full points we expect a classification of error types, frequency of errors of these types, and examples of errors. We also expect you to propose possible remedies based on the analysis. If you have not implemented the joint model, discuss if (and why/why not) a joint model could help.

6 Test Data

Along with your report please send us your predictions on the test data. The data is here: `https://www.dropbox.com/s/g0wb1n9vag8c8r0/bionlp2011genia-statnlp-test-clean.tar.gz?dl=0`. The required format is the JSON format of the input, but with an additional "predicted" attribute for each trigger and argument. The order of the sentences, events and arguments in the input data needs to be maintained.

References

[1] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '02)*, volume 10, pages 1–8, 2002.

[2] Jin-Dong Kim, Tomoko Ohta, Sampo Pyysalo, Yoshinobu Kano, and Jun'ichi Tsujii. Overview of bionlp'09 shared task on event extraction. In *Proceedings of the Natural Language Processing in Biomedicine NAACL 2009 Workshop (BioNLP '09)*, 2009.

[3] Jin-Dong Kim, Yue Wang, Toshihisa Takagi, and Akinori Yonezawa. Overview of the Genia Event task in BioNLP Shared Task 2011. In *Proceedings of the BioNLP 2011 Workshop Companion Volume for Shared Task*, Portland, Oregon, June 2011. Association for Computational Linguistics.