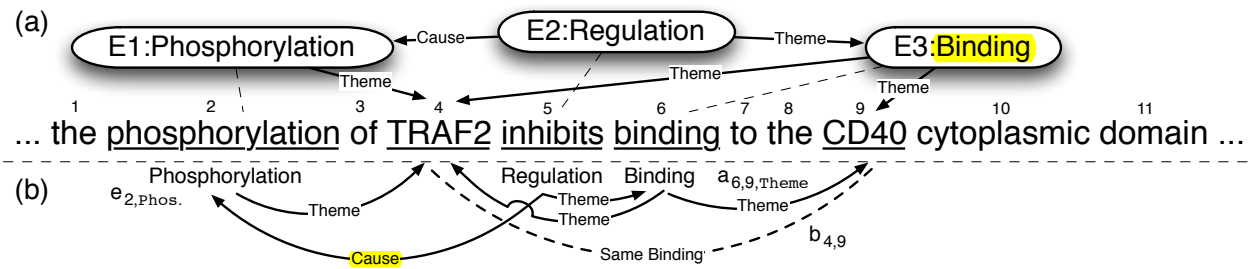
Assignment 2: Event Extraction

Sebastian Riedel
Due: Monday 21st December

November 23, 2015

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 construction 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.

- Tokens are labelled with annotations such as "Phosphorylation" or "Regulation", indicating whether the token expresses a particular event. In this case, we refer to this token as an *event trigger*.
- There are labelled edges between the event trigger and other tokens which specify the structure of the event. These edges indicate whether the event grounded at the source trigger word has an argument (the target word of the edge), which can be another event or protein. These arguments again have a label (e.g. Cause, Theme) that specifies how the argument relates to the event trigger.

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

2 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.

To get a feeling for "low-level" NLP implementation, for this assignment you are are not allowed to use ANY existing NLP packages, outside of the code provided and the `stat-nlp-book` code. The general UCL plagiarism rules apply.

2.1 Report and Format

Your submission needs to consist of a report, in which you will summarise what you did and how you addressed the posed questions, and the code you hacked (filled in code stubs).

You will use the NIPS[1] style files for writing your reports. You can find out more about these styles on `https://nips.cc/Conferences/2015/PaperInformation/StyleFiles`. There are LaTeX, Word and RTF templates available. Your submission should be 6-8 pages maximum, with maximum 2 additional pages for references. Images do not count against the maximum. Rules written here trump all the non-formatting instructions in the NIPS style files. If something is not clear regarding the style and instructions for the report, please post your question to Moodle. You need to submit a camera-ready version (not anonymised!) in PDF format, strictly.

2.2 Submission

Please upload your submission to Moodle as a zip file, containing: your report in PDF format (including the names of all members of your team); your code in a zip file; and your predictions for the provided test set. The code you send needs to be compilable, as we will execute it on the test set.

3 Background

3.1 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 of a feature function is

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

The loglinear model then defines the conditional probability

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

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 $\hat{c} \leftarrow \arg\max_c p_{\boldsymbol{\lambda}}(c|\mathbf{x})$
      ii. if $\hat{c} \neq c_i$ change weights: $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \boldsymbol{\phi}(\mathbf{x}_i, c_i) - \boldsymbol{\phi}(\mathbf{x}_i, \hat{c})$

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 computing the *argmax* in the 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 *structured* perceptron paper [1] in section 2.5.

[1] Conference and Workshop on Neural Information Processing Systems (NIPS) is one of the top conferences in Machine Learning

3.2 Evaluation

You will be asked to report the precision, recall and F1 measures, for triggers and arguments. These can be computed per class or averaged across classes of interest. In this case:

- Average precision is the number of correct predictions you made that are not None, divided by the total number of predictions that are not None.
- Average recall is the number of correct predictions that are not None, divided by the total number of gold labels that are not None.
- Average F1 measure is the harmonic mean of average precision and average recall.

The *Evaluation* class provides an interface for calculating these evaluation metrics.

4 Data

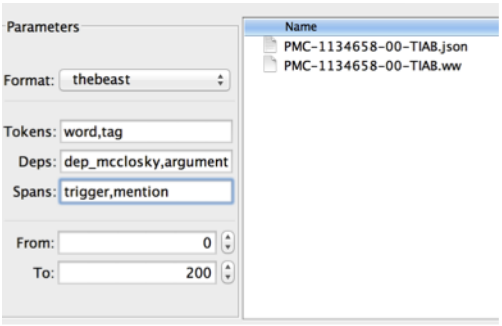
Data for the task can be downloaded from `https://www.dropbox.com/s/jgrsoqnpp5zff2t/bionlp.zip?dl=0`. Unzip the folder inside the root of your project, under */data/assignment2/*.

4.1 Training Data

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 words, tags, stems etc. We provide IO code for this JSON format. The data also contains a list of syntactic dependencies that can be useful as features. There are also protein mentions with their labels. 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 you should use the training-development set split as provided by the code.

4.2 Visualisation

It's generally very useful to visualise 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:



4.3 Test Data

The test set does 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. Submit two text files, for the triggers and arguments, respectively. Each line should contain the predicted label. You have to maintain the order in which the data is loaded. You should not sub-sample the test data!

You can use the *Evaluation.toFile()* method to write your predctions to a file. An example of its usage can be seen at the end of the main functions in *Problem3*.

5 Problems

Problem 1: Implementation of Perceptron Algorithm (15 pts)

The data generally comes in the form of event *candidates* whose labels have to be correctly predicted. *None* is one potential label for event candidates that don't refer to an actual event. Implement the Perceptron training algorithm in the *trainPerceptron* method in file *Problem1*. Use it to then train two (log)linear models, one for Trigger extraction and one for Argument extraction. Your solution code to this problem should be contained within the file *Problem1* and must be compilable. Report your evaluation results on both tasks in your write-up.

Hint: Familiarise yourself well with the code that is provided. You can, for example directly use the *predict* function that is provided as input argument inside the perceptron loop.

Problem 2: Implementation of Average Perceptron (10 pts)

Implement the Average Perceptron training algorithm. The main obstacle for this task is to do this in an efficient way. Describe and motivate your solution in the report.

Hint: You have to be clever with the accumulation of weights for features which have not been updated. Naive accumulation of the weights for all features after every instance will be slow and inefficient.

Problem 3: Feature Engineering and Evaluation (25 pts)

This problem requires you to develop both trigger and argument classifiers. For each you are asked to explore two aspects: the choice of features and the choice of learning algorithm. You should come up with your own features, but they should comprise at least one instance of each of the feature classes below:

- at least one lexical feature (Hint: tokens);
- at least one entity-based feature (Hint: mentions);
- at least one syntax-based feature (Hint: dependencies);

You should come up with at least one more feature not mentioned above. Notice that trigger and argument classifiers need quite different types of features.

As possible learning algorithms you should test the Naive Bayes (MLE) algorithm and the perceptr on algorithm. Notice that we provide implementations for both of these algorithms in the code.

In your report you need to:

- Describe and motivate the features your developed. Show positive examples in the training set that each feature can help identifying, or negative examples that the feature can help to down-weight.
- List and discuss, for each task and learning algorithm, the best set of features and the corresponding evaluation results.

Implementation Hints You will need to provide your own implementation of feature functions in the file *Features* and call them from inside *Problem3*. To see how features can be defined, you can e.g. inspect the function *defaultTriggerFeatures*. It is also worth examining the members of the *Candidate* class. Finally, you can also read about other systems that participated in the BioNLP shared tasks to get inspiration for new features. Argument extraction is also very sensitive to conjunction of features.

Problem 4: Joint Perceptron (20 pts, no code required)

We want to derive a joint model of trigger and argument extraction. Such a model assesses the quality of a trigger label jointly with the quality of its argument extractions. Your goal in this problem is to propose two search algorithms: one for efficiently finding the *argmax* in an unconstrained joint model, and one for finding the *argmax* for a constrained model. We expect each algorithm in pseudo-code, together with a description of what it does.

Unconstrained joint model: In the most basic joint model, the score of a trigger-argument structure is decomposable as a sum of scores for trigger label and all argument labels. More formally, for a given token $\mathbf{x}$ and candidate argument tokens $c_1 \dots c_m$ let e be the label of the trigger, and $\mathbf{a} = a_{c_1}, \dots, a_{c_m}$ the labels of the argument candidates. Then the fully factorized model scores this "frame" of e and $\mathbf{a}$ using $s\left(e, \mathbf{a}; \mathbf{x}, \boldsymbol{\lambda}\right) = s\left(e; \mathbf{x}, \boldsymbol{\lambda}\right) + \sum_j s\left(a_{c_j}; \mathbf{x}, \boldsymbol{\lambda}\right)$. Here the scoring functions themselves are dot products of feature vectors and weight vectors, as used in equation 2. This

model can be trained by reusing the perceptron algorithm. You will need to define a search routine for finding the *argmax* in its inner loop. How would the search routine look like for this joint model?

Constrained joint model: The next possible extension is a model that still factorizes in the same way, but limits its set of "legal solutions" to assignments that fulfill some constraints on the structure of the event. Again, this requires you to define the search routine (*argmax*) in the inner loop of the perceptron training algorithm. The events should satisfy 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

Problem 5: Implementation and evaluation for Problem 4 (20 pts)

For this problem, use the best set of features from your experiments with the perceptron trainer in Problem 3.

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 re-using the perceptron algorithm. Evaluate by comparing against the isolated models in per-trigger and per-argument evaluation.

You will need to implement the *predict* method of the *JointConstrainedClassifier* in the file *Problem5*. Your solution to this problem should be contained within the file and must be compilable.

Problem 6: Error Analysis (10 pts)

For this problem, choose the best model that you have developed so far (can be the NB, Perceptron or one of the joint models). Analyse the errors that it produces. In particular, you need to:

- identify types of errors and give examples
- report the frequency of errors of these types
- discuss what types of errors the joint models helps to prevent
- propose possible remedies for unresolved errors

Problem 7 (Optional): Joint Conditional Likelihood (10 pts, no code required)

Suppose we want to train the joint model by maximizing the conditional log-likelihood of the data (equivalent to Maximum Entropy Training) and use Stochastic Gradient Descent/Ascent:

$$\sum_{(\mathbf{x}_i, c_i)} \log p_{\boldsymbol{\lambda}} \left(c_i | \mathbf{x}_i \right)$$

where $\{(\mathbf{x}_i, c_i)\}$ is the training set.

Derive the gradient of this objective with respect to $\boldsymbol{\lambda}$ and a method to efficiently compute it. Hint The gradient will require you to calculate some form of model *expectations*. There is a naive and prohibitively expensive way to calculate these expectations. Describe this naive approach, and then develop a tractable method for calculating the required expectations.

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.