

# COSI 134: Homework 1

**Due: October 5, 2010**

## **Policies:**

Assignments are due at the beginning of class on the specified due date. Each student receives 5 late days for homework assignments (a late day extends to mid-night of the following day). Weekend days do not contribute to late days<sup>1</sup>. When no late days are available, 10% of the value of the assignment is lost each day. You are welcome to work together to discuss the ideas involved for each problem, but each person should write-up their own work. You are also welcome to look at existing implementations, text books, etc. However, you must cite these sources.

Each assignment should be computer formatted using Word, Open Office, LaTeX, etc. Included figures or tables should be properly labeled and referenced.

## **Problem 1 (20 points)**

Maximum Entropy models are simple, but powerful discriminatively trained probabilistic models that perform well for many NLP tasks that involve many features.

Write out the pseudo-code for an algorithm to estimate a maximum entropy model from a training set (using maximum likelihood estimation). Assume the following function is already defined:

$$\Lambda \leftarrow \text{Update}(\Lambda, L_D, \nabla L_D)$$

This function takes the current set of parameters,  $\Lambda$ , the current log-likelihood,  $L_D$ , and the gradient of the log-likelihood,  $\nabla L_D$ , and produces a new set of parameters  $\Lambda$  that will (typically) increase the log-likelihood value. Hint: The main function you need to write is one that computes the log-likelihood and its gradient given the current parameter values.

## **Problem 2 - Programming/Experimentation (50 points)**

This problem involves text classification. You will train and evaluate one or more classifiers using the Reuters text classification dataset. There are two options for this part of the assignment.

---

<sup>1</sup> A couple of examples related to an assignment that is due on Thursday at beginning of class: If the assignment is turned in at 11:59pm on the subsequent day (Friday) one late day is used; if the assignment is turned in at 11:59pm on the following Monday, two late days are used.

### Option A)

For this option you will use NLTK to build a set of different classifiers to classify articles in the Reuters text classification dataset. Note that the dataset actually assigns multiple categories to each document. How can this be handled using classifiers? After devising your approach, experiment with at least 3 different classifiers in NLTK. Experiment with the various options and hyper-parameters available to each classifier. Also try at least 2 different methods for selecting features. What classifier(s) work best on this data? What feature selection methods worked best? Provide a detailed report of your results along with a summary. Discuss what might be done to further improve your results. Also hand in any code used in your experiments to select or construct features and to invoke various classifiers.

### Option B)

For this option, you will implement a Naïve Bayes classifier. You are free to decide whether to implement the Bernoulli or Multinomial version of Naïve Bayes. You should build an estimator that estimates the class-conditional posterior probabilities as well as a predictor/decoder that will classify new instances. You may choose any text classification dataset or similar problem to evaluate your algorithm. You are free to select whatever programming language you wish.

It may be easiest to use one of the datasets provided in NLTK and dump the data out in a format suitable for your program. Or – if you are using Python to develop your algorithm, you will probably find it easiest to just work off of the existing NLTK representations.

Provide results indicating that your algorithm works properly in terms of training and testing accuracy. Try selecting the “best” features for your model by either a simple count cut-off or using mutual information. Report results for different numbers of features based on your chosen selection method: e.g.  $N=100$ ,  $N=1000$ ,  $N=10000$ . You should also hand in a copy of your source code, properly commented.

### Problem 3 – Experimentation with CRFs (30 points)

Carafe is an open-source tool developed at the MITRE Corporation that implements Conditional Random Fields and related methods for training and applying sequence models for NLP applications.

For this assignment you will experiment with Carafe using different feature sets as well as different learning parameters and methods. You should experiment with different feature sets aiming to reach at least 92.5 phrase F-measure on the development/test data. Results above 94 will warrant some extra points (up to 20 points).

Both stochastic gradient descent training (the `--psa` flag) and conditional log-likelihood maximization training should be used and compared (you need only do this with your best feature set). For CLL maximization, you should experiment with Gaussian prior values of 0.5,

1.0, 10.0 and 1000.0. With stochastic gradient descent, you should experiment with 1, 3, 5, 10 and 15 epochs of training. Also, try using different learning rate parameters 0.1, 0.3, 0.5.

Write up your results. The experiment results should be in a table or as plots on a graph. You should also explain your results clearly yet briefly as if writing them for someone unfamiliar with the task (but you need not explain CRFs in detail). In particular, discuss any interesting trends in your experiments with different learning methods and parameters and describe at least three feature set configurations in detail along with any linguistic intuitions that motivated you. Finally, look at the output of the system, identify a set of errors and report any interesting observations.

## (Very) Short Guide to Using Carafe

Below are some example invocations to get started.

### *Training:*

```
java -Xmx512m -jar jcarafe-0.9.6.jar --train --input-file
./wsj_15_18.sgm --model ./m1.psa5 --psa --max-iters 5 --tag NP -
--seq-boundary "s" --fspec ./fspec1
```

This will use the provided input file for training and place the resulting model in the file `./m1.psa5`. In this instance stochastic gradient descent is being used (the `--psa` flag) for 5 epochs (`--max-iters 5`). The `--tag NP` option is necessary to tell it to look for NPs and the `--seq-boundary` argument indicates that XML elements matching the single character “s” should delimit sequences. `--fspec ./fspec1` indicates the feature specification contained in the file `./fspec1` should be used to generate features for the model.

To train a model WITHOUT using stochastic gradient descent, just remove the `--psa` flag and `--max-iters` flag (or set `--max-iters` to something like 300).

Note that if you include a LOT of features you may need to increase the amount of Java heap space. When carrying out training without stochastic gradient descent, you can speed up the training by adding `--parallel` along with `--nthreads 5` to, for example, use 5 threads which should speed up training if you have a multi-processor machine.

### *Decoding/evaluating:*

```
java -Xmx128m -jar jcarafe-0.9.6.jar --input-file ./wsj_20.sgm -
-model ./m1.psa5 --tag NP --seq-boundary "s" --evaluate
./ev1.psa7
```

This call will run the decoder on the file `wsj_20.sgm` and evaluate its output against the gold-standard annotations within the file. The results of the run will be placed in the file `./ev1.psa7`. To see what the actual output was, execute the command:

```
java -Xmx128m -jar jcarafe-0.9.6.jar --input-file ./wsj_20.raw -
-model ./m1.psa5 --tag NP --seq-boundary "s" --output-file
./wsj_20.OUT
```

Note that the input file here is the file `wsj_20.raw` which doesn't include the existing, gold-standard NP phrases (since the algorithm will be generating NP phrases).

### *Feature Customization*

Richer types of features can be introduced by modifying the feature specification file. A fuller description is provided in the documentation for jCarafe. The main operations to be aware of are the **over** and **ngram** operators. These allow one to easily create window-type or ngram-type features. For example, the specification line: “`mywindowfn as wdFn over (-2,-1,0,1,2)`” will create window functions for words at the specified relative positions to the current word. (Note: the token to the left of the word ‘as’ is simply an arbitrary name given to the feature that is ignored by the software – it’s just for human reference).

The feature spec “`myngamfn as wdFn ngram (-2,-1)`” would create a bi-gram feature consisting of the word two to the left and one to the left of the current word.

You should experiment with using a lexicon. A simple lexicon is provided in the `hw3` directory. You can include this by adding the option `--lexicon-dir ./lexicon/` when you invoke the trainer. The lexicon is stored in the resulting model file so you need not specify its inclusion when you run the decoder.

You can combine various features in different ways. So, for example, to include an ngram lexicon features, you could add “`lexNGram as lexFn ngram (-1,0)`” or something along those lines.

Finally, note that the `allTagFn` feature will add any and all attributes of a `<lex>` tag as features. Currently, in this dataset this means the part-of-speech tags, which are attributes of the `<lex>` tags will be added as features. If you are interested in pre-processing the data to add additional features computed by other means, including the `allTagFn` will ensure they are automatically picked up as features.

Example initial feature specification:

```
node as nodeFn;
edge as edgeFn;
words as wdFn;
pos as allTagFn;
```