1 General

• The assignment will be done in teams. Since we have not received any complains the teams will be the same as before:
  – Team 1: Ryan Cunningham, Maryam Karimzadehgan, Majid Kazemian
  – Team 2: Kavita Ganesan, Hyun Duk Kim, Parikshit Sondhi
  – Team 3: Sarah Borys, Daniel Schreiber, Scott Wegner
  – Team 4: Gourab Kundu, Oscar Sanchez Plazas, Mehwish Riaz

• Your final report on the assignment is due on Monday, April 6. At this time, each team will turn in its results along with a short report describing what you did, what were the difficulties and what are your conclusions.

• Also we would like to ask each of you to send me a short email describing your role in the team for this assignment also on or before April 6.

• On Wed, March, 25 you should send me a short email (one from each group; 2-5 sentences) describing where you are (what is done, what is left to do, problems if any).

• Feel free to e-mail me questions (we will be away until Mar, 31 but we will be checking and responding to email regularly).

2 Assignment

The goal of this assignment is establishing a pipeline for syntactic parsing and semantic role labeling (SRL). The pipeline should include the following set of stages:

1. Part-of-speech tagging
2. Syntactic Parsing
3. SRL labeling

For the first and the second stages you are expected to use third-party tools downloadable from the web (see below). For the third stage (SRL) you should implement your own system, though, if needed, you are allowed to use a general-purpose learning package (e.g., SNoW). The defined task is directly related to the CoNLL 2008 and CoNLL 2009 shared tasks (their closed challenge).
You will need to convert the provided data to apply tools on every stage of this pipeline. You may perform feature engineering for first two stages (this is optional – you may use feature models provided with the tools). You will have to design a feature representation for the last stage (SRL). The task was described in the lecture on Mar, 6, please see the slides.

3 Reading

Before starting to implement the pipeline we strongly recommend that you read the following papers/info:

- CoNLL-2008 shared task overview
- Explore the shared task web-site
- Read papers for at least a couple of top systems in the closed challenge (the papers are downloadable here and the score table is available here)
- Discriminative Classifiers for Deterministic Dependency Parsing, CoLING 06 by Hall, Nivre, Nilssen (Description of the MALT parser)

4 Project Report

1. Describe what you did, the specifics of your models, and the rational behind your decisions
2. Provide the code for your preprocessing and for your model estimation and evaluation.
3. Present the output of your program on the test corpus.
4. Package the code so that one can run it also with a different corpus (details below)

5 Data

On the first stage you will receive two files: train.dat and dev.dat, development and training sets. They will be in the format of the CoNLL-2008 shared task format, however fields 3, 5, 6, 7, 8 will be empty (“.”). See the description of the data format on the shared task web-page.

The final testing file (test.blind.dat) will contain ONLY the first and the second columns (i.e., only word forms), you will receive the final testing file 3 days before the deadline and you are not supposed to use it for anything expect the final testing of your model. As an example, in the initial distribution, we are additionally providing the development set dev.blind.dat in the same ‘blind’ format.

Note: you are not allowed to distribute this data, please do not place it in any publicly accessible locations (e.g., on your web-page).
6 Tools

We will use the same software for scoring and the same evaluation metric (macro F1 score) as were used in CoNLL-2008 shared tasks (see CoNLL 2008 shared task description mentioned above). Please download the evaluation script here.

7 Training the pipeline

As it is described above, in order to complete this assignment you need to assemble a pipeline where the predicted output of one stage is used in the next stage (e.g., predicted POS tags are used as the input for the dependency parser and the SRL model, predicted dependency structures are used as the input to the SRL stage, etc). There are different approaches how to train a pipeline of classifiers, the most standard are:

- **Each stage is trained using gold standard annotation (features) instead of the predicted ones**: the drawback of this approach is that the classifiers learn to “over-rely” on the output of the previous pipeline stages since they are trained on correct (noiseless) labels. However when tested they will use annotation predicted by the (non-perfect) classifiers instead.

- **Training in folds**: training data is split on \( N \) folds (\( N \) is generally large, e.g., 10), \( N \) first stage models are trained each using different \( (N - 1)/N \) portions of the training set. Then the outputs of each such a classifier on the remaining \( 1/N \) portions are used to train the second stage classifier. This guarantees that these first stage classifiers have never observed these portions of the training data in their learning and, thus, the distribution of errors on these chunks is comparable with the distribution you would get on testing data. The hope is that in this set-up the second stage classifier will learn how to predict from noisy labels. This approach can be trivially generalized to pipelines longer than 2 stages. The problems with this approach are:
  - the approach is computationally expensive, especially for long pipelines,
  - the second stage models correct models which are different from the one which will be used on final testing (i.e. from the first-stage model trained on the entire dataset).

Even though in practice one would expect to get (considerably) better results with the second approach, for this assignment we recommend you to use the first approach as it will be simpler and computationally less expensive. In the following sections we will describe each step of the pipeline.

7.1 POS tagging

For POS tagging you are encouraged to use the SVM tagger. Please refer to the documentation provided on their web-site for format conversion, installation and use.

7.2 Dependency Parsing

For syntactic parsing we recommend the MaltParser. Again, please refer to the documentation provided on their web-site for format conversion, installation and use.
7.3 Semantic Role Labeling

You will have to implement a semantic role labeling component. For this assignment the semantic role labeling component will be quite basic, in the following assignment (the last one) you will have to improve this component. The SRL component you should develop is a pipeline consisting of 4 elementary stages:

- deciding which words are predicates (i.e. the predicates are the words which can be heads in the semantic dependency structure) – a binary classification problem, one decision for each word;
- deciding which arcs should present in the data, a 3-class classification problem for each pair of words in the sentence, classes are
  - arc from left to right (the left word should be selected as a predicate on the previous stage)
  - arc from right to left (the right word should be selected as a predicate on the previous stage)
  - no arc

(you do not need to consider pairs where both words are not predicates)
- labeling all the arcs (multiclass classification problem for every arc)
- deciding on the sense of the predicate (among senses observed for this word in the training set)

You are not allowed to use specific SRL packages to implement your SRL stage. However, you can use standard learners (e.g., SVMLight and SVMStruct, etc). If you decide to implement your own learners, we suggest implementing the Averaged Perceptron. You may want to consider the version of the Perceptron which enforces thick separation. In this version the parameter vector is updated even if the prediction is correct but the score is within $\gamma$ from 0, where $\gamma$ is a parameter you tune on the development set. The multiclass Perceptron was discussed in the problem set distributed on March 6 and explained by Alex Klementiev, but you can also read here.

In either case, whether you use standard learning packages or not, you will have to use features from the previous stages of the pipeline. E.g., features used to decide whether to create an arc may include:

- forms and POS tags of the words;
- the label of the syntactic dependency relation between them (or a feature value indicating that there is no dependency link between them)
- syntactic heads (their POS and form) for both words;
- the label of the syntactic dependency relation of each of these words to their syntactic heads and others.
8 Software Requirements

We require that you complete this assignment in a Linux/Mac environment. You are required to submit three shell scripts, a method for compiling your source code, and running it. You can expect that reasonably recent versions of java, gcc, python, perl, awk, bash are install on the system. If you need anything ‘exotic’ your scripts should be able to install it.

8.1 Compilation

The preferred method for compiling your source code would be a single Makefile, such that we could simply type make in the submitted directory and everything would compile. However, since we are not restricting the programming language, compilation methods may differ. Therefore, we ask that you create a simple method for us to compile your source code. Seemingly possible options would include makefiles and/or shell scripts, but the only requirement is that it is easy for evaluation purposes.

8.2 Preprocessing

As stated, you will need to modify the data provided into a form appropriate for the used tools. You can assume that FEX is installed and you do not need to specify path to them (if you need this tool). You can also assume the same about svmtagger, mwtaparser and snow.

As many of you requested for the first assignment, you do not need to have a separate script for data conversion. Assumption is that the data conversion is performed before training and testing within train.sh and test.sh scripts.

8.3 Learning / Estimating Parameters

The name of the script used to train the model should be train.sh (no parameters). You should try to send us all the needed files so that we do not need to modify your script. You should place the provided train.dat, dev.dat, dev.blind.dat or any other files you can possibly need (e.g., the evaluation script eval08.pl) in the package you send to us by email. train.sh should also output the accuracy number on dev.dat (output of eval08.pl with -q option).

8.4 Testing

The final script we require is the testing script, which we will call test.sh and it should accept a single parameter - the file name of a test file. It should output a file in the original format (the same as for train.dat). Important: You should also include this output in you package (test.model.dat).

The minimal set of experimental results that must be included in the reports are:

- Classification errors on each of sub-tasks (POS tagging, dependency parsing, macro F1)
- More detailed information on SRL (accuracy of predicate sense prediction, labeled/unlabeled semantic arcs F1)
- Experiments suggesting influence of different types of features for SRL (evidence of feature engineering)
8.5 Summary

If you do everything correctly, we should be able to type something along the lines of

tar zxvf team3.tar.gz

cd team3
make
./train.sh
./test.sh ~/TTT/test.blind.dat > test.model.dat
./eval08.pl -g test.gld.dat -s test.model.dat > test.eval
vi test.eval