[Intro to NLP, CMPSCI 585, Fall 2014]

NER tagging for Twitter

The final project will be to construct an NER tagger for Twitter. The task of *named entity recognition* is to take tokenized sentences as input, then recognize *spans* of text that correspond to a name. We will not require entity types to be predicted.

Here are two examples, showing the bracketed name spans **like this**, and then the BIO representation.

Football game tonight !!!!! **V-I-K-I-N-G-S** !!!!!!!

Football	O
game	O
tonight	O
!!!!!	O
V-I-K-I-N-G-S	B
!!!!!!!	O

@paulwalk It 's the view from where I 'm living for two weeks . **Empire State Building** = **ESB** . Pretty bad storm here last evening .

@paulwalk	O
It	O
's	O
the	O
view	O
from	O
where	O
I	O
'm	O
living	O
for	O
two	O
weeks	O
.	O
Empire	B
State	I
Building	I
=	O
ESB	B
.	O
Pretty	O
bad	O
storm	O

here	O
last	O
evening	O
.	O

This task is possible but plenty hard: in current research, systems typically get F-scores in the 60% range. The task is described in Ritter et al. 2011, and we use their data for the training and development sets.

Groups of 1 or 2, and collaboration policy

You can either do this project individually, or in a group of two. Please decide this by the milestone report date.

If you decide to do this as a group, when you submit the milestone and final reports/code, please submit a single copy of the materials, and make sure the names of both team members are on them. You both will receive the same grade.

This project is more open-ended than the problem sets. After building the base system, you are free to explore different types of features and analyses. Feel free to discuss the project with anyone else in the class. Discussions can help spur creativity for features or experiments to try. However, all code should be written individually or within your two-person group. The project reports should also be written only within the groups.

Data and Evaluation

- We provide an initial training and development set.
- We provide an evaluation script so you can test your tagger on the development set, on your own machine.
- You can also submit your development set predictions to the Kaggle leaderboard, which will evaluate them for you.
- The final evaluation will be done with a test set you don't have access to until the day of. This is intended to simulate a real situation, where you care about how your system does on new data it sees in the future.

This format is sometimes called a *shared task* or *bakeoff competition* (an obscure cultural reference, maybe).

Also, all students in the course will annotate a small amount of data as part of a problem set or exercise. These annotations will comprise part of the final test set. (Exercise 9)

Unlabeled tweets

We also are providing an external resource, a large corpus of unlabeled tweets, for possible use in features or additional data analysis. It consists of about 1 million English-language tweets that have been tokenized. We'll post the URL on Piazza (since we are not supposed to redistribute the data freely on the internet).

The 1-million tweets version has 986,784 tweets sent over Jan-Sep 2014. If this is too much data for what you want to do, just take a subset of it. (The file is sorted in a random order). The versions we have available include:

- Just the tokenizations: 13,560,980 tokens, 78 MB
- With metadata (timestamp, original text, author's username): 203 MB
- Original tweets with complete metadata: 3.3 GB

If you want more data than this let us know and we can give you as much as you want.

Software

You can use whatever NLP/ML software or resources you like. We will provide a small bit of starter code to use CRFSuite, a software package that does first-order CRF sequence tagging. It requires you to run your own script to extract observation features as a text file. Then you tell it to train and predict with these feature files.

We provide starter code for

- A very simple feature extractor.

- The evaluation script to evaluate accuracy locally. It computes precision and recall of name spans.

More details on the milestone page.

Deliverables

There are several points to turn things in. **Update 11/30:** see posts on Piazza for the current timeline. I strikethrough'd changes here.

1. (Due Sunday, Nov 16) Milestone: create a very bare-bones tagger for the NER task, with the training and development set, and submit predictions to the Kaggle board, plus submit a short document about it.
2. ~~(Released Monday, 12/1; due 12/2 at noon)~~ Test set evaluation for the competition: ~~on Monday morning,~~ We will provide the final test set, and it will be blind --- no labels given! You have to run your system to create predictions and submit them. We will set up Kaggle to allow ~~up to 3~~ a small number of submissions if you want to try improving your model in the meantime. **Only submit predictions that your system produced.**
3. ~~(Due Friday, 12/5)~~(Due 12/12) The code submission, and final project reports, turned in via Moodle. We will accept the final reports (and accompanying code) until 12/12. ~~but please submit at least a basic draft on Friday 12/5 along with your code.~~

Grading

The project is, in total, 20% of your grade. The milestone is 5%. It's designed to help the success of the final result. The rest is derived from both:

1. System building.

- Required: core system with lexical, character affix, and shape features, plus positional offset versions.
- Optional: One or more possible extensions of new features to try. (At least one is required if you are working in a group of two.)

2. Analysis and exploration in your project report, such as analyzing the model weights, or doing ablation tests.

Finally, there will be extra credit for the top performers on the final test set. Extra credit will also be available for doing additional extensions, or doing some sort of additional analysis project.

Random tips

When feature engineering, it will be useful to write a shell script that runs the entire pipeline of feature extraction, training, testing and evaluation. Here's one example, which you will need to customize for different feature extraction scripts or whatever you're using:

```
#!/bin/sh
set -eux
python simple_fe.py
crfsuite learn -m mymodel train.feats > train.log
crfsuite tag -m mymodel dev.feats > predtags
python tageval.py dev.txt predtags
```

When the test set comes out, you will want to train your system not just on train.txt, but on the concatenation of both train.txt and dev.txt, since that's more data to work with. Do not train on the test set, of course, since that's a form of cheating (and in order to prevent honest mistakes with this, we will distribute the test set with `?' labels for all the tags).

Details on milestone

These are in milestone.html

Details on final project requirements

If you implement the minimum possible basic feature extractor with a basic analysis, that will earn you a B+. (A group of two additionally requires one major extension.) Both feature extensions as well as more analysis will earn you a higher grade. Especially high performance on the test set will earn extra credit (we are currently planning on awarding extra credit to the top two teams). All implementation code should be submitted along with the final report.

We expect the final project report to be at least 8 pages, but shorter than 20. (Once you start writing it, you'll find it's much easier to write more than you might have originally thought.) This is intended to include tables and graphs (which take up a lot of space). These page limits are not hard and fast, but are intended to give you an idea of how much analysis and detail we're expecting.

System building

We require a feature extractor that produces, at the very least, the following types of features. These are fairly typical features in NLP systems. There are many approaches to the following features; please report the approaches you used and whether you found them helpful, in your final report.

- Lexical/wordform: features for the word (this is included in simple_fe.py). Lower cased version may be helpful too.
- Character affixes: the first 1, 2, and 3 characters in the word; also the last 1,2,3. At the very least, you need to get whether the first character is # or @, since the conventions are to never tag those as names.
- Shape features: whether the word's is uppercase followed by lowercase characters; or is all uppercase, or is all lowercase. Many variants are possible. One popular approach to shape features is to map characters to a few classes and eliminate repetition, e.g. "Twitter" to "Aa" and "Twitter3" to "AaD".
- Positional offset versions of the above, like whether it holds for the word before or after.

As for major extensions, many are possible. Some examples of what we will consider to be a major extension include:

- Use a different ML algorithm or tagging model software, such as one of the following. For credit you need to report results that are compared to results from a CRFSuite-based system.
  - a CRF with a different network architecture: for example, a higher-order CRF. (CRFSuite only does a first-order model.)
  - a structured perceptron, or max-margin structured SVM, that you implement yourself.
  - Span classifier, that independently classifies every span, say with multiclass logistic regression. A span classifier is more powerful in some ways than a BIO linear-chain CRF because it can use features specific to the entire span (and potentially a very long span), but it is less powerful because it allows for contradictory inferences and cannot exploit contextual classifications. *Notes:*
    - You could try using scikit-learn or another library for multiclass logreg.
    - If you have contradictory span classifications, you won't be able to use tageval.py (though you can still submit to Kaggle if you write your own output code); or you could do something to resolve them.
  - a stacking model: you run one model to make basic predictions, then run a second model that uses its predictions as features to make the final predictions. One popular approach is to use token-level classifications as the first model.
  - *Notes:*
    - Modifying the crfsuite options doesn't count for this extension, sorry, though it can for the analysis section.
    - It is typically a good idea to try to find open-source software that implements what you're interested in, but implementing something yourself is very educational.
    - The Factorie library, developed at UMass, may be of use. David Belanger has contributed to Factorie can answer questions about it.
  - More difficult options:

        - A semi-CRF, which combines the best of both linear-chain CRFs (like crfsuite) plus span classifiers.
        - A neural network model (we didn't go into these).
        - Adjust the precision/recall tradeoff (CRFSuite can't do this, but if you have your own model ask if you want some ideas how to do this).
- Use external lexical resources for additional features. The idea is, you look up words to see if they're in these dictionaries; if so, this gives you features that generalize beyond individual words. Several options are shown below.
    - One option: already-made resources you download from the internet. Please include at least 2 to count.
        - Name lists, a.k.a. "gazetteers". For example, lists of common person names, place names, sports teams, etc. Ratinov and Roth 2009 has a list of examples that they found useful for their non-twitter NER setting. Possible sources of these include Freebase, the U.S. Census, etc.
        - Word clusters that were learned automatically from Twitter text. The CMU ARK TweetNLP webpage has one hierarchical Brown word clustering that's available (which was found to be useful for POS tagging).
        - Word embeddings (continuous-space vectors per word) that were learned automatically from Twitter text. The Stanford GloVe webpage has a few that are available.
        - *Notes:*
            - It may be useful to perform some sort of fuzzy lookup, instead of only exact string match. Also note that multiword lookups may be useful too.
            - When assessing the usefulness of a lexical resource, it may be helpful to look at its coverage -- percent of train and/or dev tokens that have at least one entry in the lexical resource.
            - The train/dev data was collected in 2011. But our final test data will all be from tweets that were written during 2014, so word and name frequencies will be different. If you have a name list that includes celebrities who became popular after 2011, it may not help the devset much, but may be useful on the testset.
- Use the unlabeled data. Possibilities include:
    - Use it for self-training.
    - Train your own word clustering or word embeddings on it and use them as features.
        - *Note:* One thing to watch for is preprocessing (e.g. normalize @-mentions to a single symbol, and URLs to a single symbol). For examples of preprocessing, see Owoputi et al. 2013.
- Use separate NLP tools to produce features or outputs.
    - Part-of-speech tags are often used as an input for NER; you could use an external POS tagger to generate them.
    - Train a different open-source NLP tagger.
    - *Notes:*
        - When experimenting with the dev set, be careful that no information in the devset is already on the training pathway into the final results. For example, you may not want to directly use the github.com/aritter/twitter_nlp system, because its released models were trained on data that includes everything in

our dev.txt. This will give you overly inflated scores for the devset, but you won't do as well on the real test set!
- Please only use open-source tools. Do not use any proprietary tools.
- On what counts as sufficient: if you're just running an external tool and taking its outputs as features, you have to do that for at least 2 tools to count as a major extension. If you are retraining a pre-existing system, that counts as a major extension.

Another idea (we don't know how to grade this yet but): You can annotate new data yourself to use for training. More annotated data always helps, and while this is sometimes a controversial point, some researchers believe it's more important to have more data rather than fancier machine learning or linguistic algorithms. It's useful to graph the learning curve before deciding to do this -- see below

Analysis and exploration

Report your performance on the test set for the competition, plus any other results you made before or after the competition's result submission. (Typically, you may find interesting things to do after the competition is over.)

Please explain your features and system and the choices you made for it. Explain your reasoning and any experimental results you have. Things like "we tried X but it didn't work and here is why" are fantastic and will give you full credit for attempting X.

Your analysis/exploration section is expected to have at least one additional analysis component. Some examples of analysis components include:

- Explore different hyperparameter settings or algorithm choices in the ML algorithm provided by CRFSuite (or another library you choose to use) -- for example, different types of regularization and their hyperparameters, or different types of learning algorithms. Make at least one graph of regularizer strength against devset performance (regularizer strength is what we called "lambda" in lecture, and CRFSuite calls "c1" and "c2"). For results that don't make sense to show as a graph, show results as a table. To count as a basic analysis for credit, this option requires at least 5 different parameter settings.
- Graph a learning curve: see how devset performance improves when you train with more data (or how it worsens with smaller subsamples of the training data). A learning curve has the y-axis as devset performance, and the x-axis as training set size. If it hasn't flattened out yet, it may be useful to annotate more data. (Doing just a learning curve on the original training data is good, but it's not enough to count as a full analysis component.) To count for credit, make at least 1 graph similar to the Miller et al. 2004 graph (or like the Koo et al. 2008 table) that was in the 11/18 lecture.
- Ablation testing: remove classes of features from your final model and see how much performance goes down. And/or, take a base system and add classes of features to see how much performance improves. To count for credit: do at least 4 ablation settings (4 cases of adding/removing an interesting class of features).
- Error analysis: look at errors your tagger makes and analyze why, and what sorts of features might help in future work. You can also compare different versions of your tagger to each other: examine the differences between their taggings, to help understand whether certain new features did or did not help. (We don't have a clear

standard of how much error analysis will count for the requirement, but a few pages would make sense.)

- Data exploration: run your tagger on the unlabeled data and analyze some interesting aspects of the named entities it outputs. For example:
    - What are the most common named entities?
    - Plot some over time. What are their trends? Are there names that increase, decrease, or spike in popularity?
    - What named entities tend to co-occur with other ones? What named entities tend to co-occur with happy or sad emoticons or emoji?

Final project grading

Here's our guidlines for how grading will be done. Two-person teams require one major extension to make the B+. Therefore "extra extension" for a solo team would mean one extension, but for a duo team would mean two total.

- B-: Basic features but no analysis
- B+: Basic features and an adequate analysis component
- A-: An extra major extension, or an extra analysis component
- A: An extra major extension *and* an extra analysis component / really great analysis in general

The projects will also be judged by the quality of writing, the quality of results, the insightfulness of the analysis, and the amount of thought and work that went into the project.

Project report organization

Here's one possible organization of the final report. If you're not sure how to organize it, we strongly suggest using this outline.

This starts with the "WHAT" and "HOW" of your system. Then it moves into "WHY": *explanations* for what works or doesn't work.

In terms of organization, the "Results" and "Additional Experiments / Analysis" sections may blend into each other a little bit, depending on what you're doing. That's fine. The point is to present things in a way that is clear to the reader.

Remember that your analysis/exploration is expected to have at least one additional analysis component.

The page lengths are rough guidelines.

Title and Author Names

Abstract (0.5 pages)

- One paragraph stating, at a high level, what the major features are, and what main experiments you did, what the major results were, and any conclusions about what works or didn't. Try to make this 6 sentences or less.

Description of Implementation (1 page)

- Describe how your system works: describe the major classes of features, and if you have multiple different systems, describe their differences.

Major extensions(s)

- If you did any major extensions, describe each in its own section. Be very clear what they are and what you did. If you used additional data, be very clear where it came from. Ideally, a reader should be able to replicate what you did.

Results (2-3 pages)

- Describe the system that you used for the kaggle competition, and what the score was.
- Describe other major performance results of your system as well -- not just for the competition, but also any later changes or variants you did. You can report either on the devset or on the test set (after we release it). For any results you present, be very clear about what data was used for training and what data was used for testing.
- The main results should be together in a table with four columns: Name of system, Precision, Recall, and F-score (similar to the analysis in PS4).
- It may be convenient to present ablation results in this section as well.

Analysis / Exploration (4-5 pages)

- Present your additional experiments and analyses here: error analysis, learning curves, exploratory analysis, further ablations, lexicon coverage analysis, unlabeled data analysis, etc. Ideally, you should have hypotheses why certain things worked or didn't work, and then you should investigate them to see if your hypotheses are true or not.

Discussion and Future Work (1 page)

- Anything further you want to say can go here. Can you draw any important conclusions from your experiments and work? Please give some suggestions for what could help in the future, if you were to do more research in this area. Even if you have no intention of doing so, it's important to practice thinking like this! It will help other people who read your report in the future.