Project 2: Sequence Labeling

CS 4650/7650: Natural Language Processing

January 31, 2013

Your goal in this project is to develop sequence labeling algorithms to perform part-of-speech tagging on Twitter. As before, you will be given a training and development set at the beginning, and a test set later.

Please bring a paper printout of your writeup to class. Please also submit a PDF of your response online, along with your code.

1 Data processing

Download the data from `http://www.cc.gatech.edu/~jeisenst/classes/cs7650_sp13/projects/proj2.tgz`.

The part-of-speech tags are defined in this paper [GSO+11] [`http://www.ark.cs.cmu.edu/TweetNLP/gimpel+etal.acl11.pdf`], which also describes the data and gives some benchmark results.

2 Supervised classification

Apply your classifiers from last time to perform supervised classification. The input is the word to be tagged; the output is the tag. You can use the `scorer.py` script again to compute accuracy.

If you augmented your classifier in some way (like MIRA), you may use this augmentation. Do not perform any stemming, downcasing or other preprocessing for now.

2.1 Basic classification

Deliverable 1 Using only the word itself as a feature (and an offset feature for each class), train your averaged perceptron and report the accuracy.

Sanity check I get a little better than 65%

Deliverable 2 Again using only the word itself, train your Naive Bayes classifier and report the accuracy.

Sanity check They should perform roughly the same.

2.2 More features

Now you will add some more interesting features. Try at least the following:

- The words to the left and right of the current word
- A feature indicating if the word is capitalized
- A feature indicating if the word is in Title Case.
- The first character of the current word
- The last character of the current word

The paper linked above has many more ideas for features that you could try [GSO+11].

Deliverable 3 Add (at least) these features to your averaged perceptron, and plot the accuracy versus iterations on the dev and training data.

Sanity check I get above 81% accuracy with the features above. With some more features, I get to nearly 85%.

2.3 Evaluation: recall and precision

Now we want to see how well we are classifying each individual tag. We need to distinguish two kinds of mistakes: false negatives (we should have predicted a tag, but we didn't) and false positives (we predicted the tag, but it wasn't applicable). True positives are when we predict a tag, and it was correct.

Using these counts, we can compute each tag's **recall** and **precision**. We can then combine recall and precision into a single **F-measure.**

$$R_t = \frac{TP_t}{TP_t + FN_t}$$
$$P_t = \frac{TP_t}{TP_t + FP_t}$$
$$F_t = \frac{2R_tP_t}{R_t + P_t}$$

Deliverable 4 What tag is the most difficult to predict? Using the confusion matrix from the previous task, compute each tag's recall, precision, and f-measure. (You can just copy-paste the confusion matrix into a spreadsheet if you like.) Present the results in two tables: one for the confusion matrix, one for the recall/precision/f-measure of each tag.

3 Viterbi algorithm

In this section you will implement the Viterbi algorithm. To get warmed up, let's work out an example on paper. There are only two tags, N and V. Here are the parameters:

$\log P_E(\cdot\|\text{N})$	*they*: -1, *can*: -3, *fish*: -3, ...
$\log P_E(\cdot\|\text{V})$	*they*: -10, *can*: -2, *fish*: -3, ...
$\log P_T(\cdot\|\text{N})$	N: -5, V: -2, END: -2
$\log P_T(\cdot\|\text{V})$	N: -1, V: -4, END: -3
$\log P_T(\cdot\|\text{START})$	N: -1: V: -1

In class we discuss the sentence *They can fish.* Now work out a more complicated example: *They can can fish.*

Deliverable 5 Show the trellis, and give the score for the best scoring path(s).

Sanity check There are two paths that each score -18.

Deliverable 6 Which of the two paths do you prefer? What is wrong with the other one, and why does the HMM still give it a good score?

Next, implement the Viterbi algorithm. You will want to be able to specify arbitrary weights between words and tags, and between pairs of tags. There are lots and lots of pseudocode (and real code) examples of this online. The textbook is also a good place to start.

Deliverable 7 Use your implementation to construct the trellis for the example above.

Sanity check The best scoring path should be the same as when you worked it out by hand.

Deliverable 8 Use your implementation to tag the sequence *they can can can can can fish.* Report the best scoring tag sequence and the score.

4 Hidden Markov Models

Use maximum likelihood estimation to obtain transition and emission distributions from the training data. Since your Viterbi implementation will be additive, you need to take the logs of these probabilities. Let α be the smoothing parameter for both the transitions and emissions.

Deliverable 9 For $\alpha = 0.01$, estimate the parameters from the training data, and run your algorithm on the dev data. Report the accuracy.

Sanity check I get around 73%. Running on the dev set takes around 30 seconds.

Deliverable 10 Now try a range of at least 5 different α values. Plot the resulting accuracy, with the log of alpha on the x-axis.

5 Structured Perceptron

You will now train a structured perceptron. Recall that the basic update rule is

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}_{\boldsymbol{x}}} \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) \tag{1}$$

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} + \eta^{(t)} (\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}^*) - \boldsymbol{f}(\boldsymbol{x}, \hat{\boldsymbol{y}})), \tag{2}$$

where $\mathcal{Y}_{\boldsymbol{x}}$ is the set of all possible taggings for input $\boldsymbol{x}$, $\boldsymbol{y}^*$ is the ground truth tagging, and η is a learning rate ($0 < \eta < 1$).

Your Viterbi algorithm can compute Equation 1, as long as the feature function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$ factors across edges. This means you can include any feature that has either a single y_i (with any subset of $\boldsymbol{x}$) or $\langle y_i, y_{i+1} \rangle$.

After identifying $\hat{\boldsymbol{y}}$, you need to compute the two feature vectors, and update the weights for all the features that are affected. For these experiments, you should use either your averaged perceptron or MIRA (or both).

Sanity check Because you have to run Viterbi for each training example, this may be a little slow. In my case, it takes a little less than a minute per iteration. This means that you need to develop your code on a subset of the data, so that you don't wait 20 minutes to find a bug that you could fix in 10 seconds.

5.1 Simple features

Include the following features only:

- Adjacent tag pairs $\langle y_i, y_{i+1} \rangle$
- Word-tag pairs $\langle y_i, x_i \rangle$

Deliverable 11 Run the structured perceptron with these features. Plot performance on the dev and training set every five iterations (or more often).

Sanity check I get 75.8% accuracy after 20 iterations using averaged perceptron. The entire procedure takes 16 minutes (evaluating on the dev set only every five iterations). Accuracy doesn't improve much after the first 10 iterations.

5.2 More features

Now add additional features, including at least the character-based and capitalization features from the classification part of the assignment. You can optionally add the left and right neighbor word features if you can figure out how (there's nothing conceptually difficult about it, but it may make your implementation a little more complicated).

Deliverable 12 Run the structured perceptron with these features. Plot performance on the dev and training sets every five iterations (or more often).

Sanity check I get 85% accuracy after 20 iterations when adding the character-based and capitalization features. The training time is not noticeably worse than when we had only unigram and transition features.

6 Literature review

This part is optional for 4650, mandatory for 7650

Deliverable 13 Find a research paper about supervised sequence labeling, for a task other than POS tagging. The paper should be published in the last ten years at one of the following venues: ACL, NAACL, EACL, EMNLP. Describe the following:

- The tag set
- The learning algorithm
- The features, and why they are expected to be effective
- The difference from prior work. Is it a new task? A new learning algorithm? A new feature set? Something else?

One way to search is to start with a machine learning paper (on conditional random fields or HMMs) and then use Google scholar to find application papers that cite it.

7 Bakeoff!

We're going to do another bakeoff. I plan on being more competitive this time. Think you can beat me? I will send out test data roughly 48 hours before the deadline.

Deliverable 14 Run your (one) best system, and include it in your submission as `lastname-firstname.response`. Please be exact about the filename, otherwise I might miss it.

References

[GSO+11] Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. Part-of-speech tagging for twitter: annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, HLT '11, pages 42–47, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.