Inference and Representation, Fall 2014

Problem Set 5: Structured Prediction for Part-of-Speech Tagging

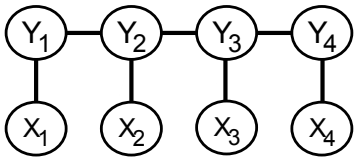
Due: Monday, December 8, 2014 at 5pm as a zip file sent to pg1338@nyu.edu. Please make sure the filename is in the format `xyz-ps5.zip`, where `xyz` is your NetID.)

The zip file should include a PDF file called "solutions.pdf" with your written solutions, separate output files, and all of the code that you wrote.

Important: *See problem set policy on the course web site.*

In this problem set, you will experiment with structured prediction using the Structured SVM algorithm on a chain-structured conditional random field (CRF). You will tackle the task of part-of-speech (POS) tagging, a problem from the natural language processing domain. POS tagging is a classification problem where the goal is to accurately predict the part of speech (e.g., noun, verb, adjective) of each word in a sentence.

The CRF used in this prediction task is a Markov model. Let L_i be the length of sentence i. Then, the CRF for sentence i has variables $Y_1, \ldots, Y_{L_i}$, where Y_l is a discrete variable denoting the part-of-speech of token l. The tokens are denoted by the variables $X_1, \ldots, X_{L_i}$. The CRF for a sequence of length 4 is shown below.



Raw POS tagging data takes the form of a set of sentences and tag sequences. Each token (normally a word but sometimes a number or punctuation symbol) in each sentence is associated with exactly one tag. Standard POS tag sets typically include around $C = 40$ distinct tags. For the purpose of this assignment, we have prepared data with a simplified tag set consisting of $C = 10$ groups of tags. Each token in the data set is assigned one of these 10 tags. To learn prediction models for the POS tagging task, we need to select a feature space to represent the tokens. For the first part of the assignment you will use simple features that we have already extracted from the tokens for you, following which you will design and implement new features.

The data consist of a set of training sentences *train-i.txt* and a set of test sentences *test-i.txt*. Each file contains one sentence. Each row in each file contains the raw token, the tag ID for that token, and five feature values in standard comma-separated-value (CSV) format. The format of each row is as follows: <*Token*>, <*TagId*>, <*Bias*>, <*InitCap*>, <*AllCap*>, <*Pre*>, <*Suff*>. A description of the fields is given below.

Column	Field	Description	Value
1	Token	The raw token string	Any string
2	TagID	The ID of the tag	$\{1, ..., 10\}$
3	Bias	Feature: Bias	1
4	InitCap	Feature: Initial Capital	$\{0, 1\}$
5	AllCap	Feature: All Capitals	$\{0, 1\}$
6	Pre	Feature: Prefix ID	$\{1, ..., 201\}$
7	Suff	Feature: Suffix ID	$\{1, ..., 201\}$

The 10 tag ID's correspond to the following 10 tag groups: verb, noun, adjective, adverb, preposition, pronoun, determiner, number, punctuation, and other. The Bias feature is a constant 1 for all tokens. The InitCap feature is 1 if the token string starts with a capital letter and 0 otherwise. The AllCap feature is 1 if the token string is all capital letters and 0 otherwise. The Pre feature takes one of 201 values corresponding to the most common two-character token string prefixes. The Suff feature takes one of 201 values corresponding to the most common two-character token string suffixes.

For each word l we have a feature vector $\mathbf{x}_l$ of dimension 5 (corresponding to rows 3–7 in the table), where x_{la} takes values in the set V_a (given in the last column of the table). The CRF has one parameter w^A_{cav} for each of part-of-speech tag c, feature a, and feature value v (note: different features a have different numbers of values as given by the set V_a). These parameters encode the compatibility between feature values and class labels. The CRF also has one transition parameter $w^T_{cc'}$ for each pair of labels c and c'. The transition parameters encode the compatibility between adjacent class labels in the sequence. All of the parameters can take arbitrary (positive or negative) real values. To be clear, there are exactly $(1 + 2 + 2 + 201 + 201) \cdot 10 + 10^2 = 4170$ parameters to be learned. The log-potentials are then given by:

$$\begin{aligned} \theta_l(y_l, \mathbf{x}_l) &= \sum_{a=1}^{5} w^A_{y_l,a,x_{la}} \\ \theta_{l,l+1}(y_l, y_{l+1}) &= w^T_{y_l,y_{l+1}} \end{aligned}$$

Given a new sentence of length L, we predict its part-of-speech tagging by MAP inference in this CRF,

$$\text{Predict\_POS}(\mathbf{x}; \mathbf{w}) = \arg\max_{\mathbf{y}} \sum_{l=1}^{L} \theta_l(y_l, \mathbf{x}_l) + \sum_{l=1}^{L-1} \theta_{l,l+1}(y_l, y_{l+1})$$

Since the CRF is chain-structured, MAP inference can be performed in linear time using variable elimination or max-sum belief propagation. Suppose that $\mathbf{w}$ is the weight vector (i.e., of dimension 4170) and $\mathbf{f}(\mathbf{x}, \mathbf{y})$ is the feature vector (the sufficient statistics). Then, $\text{Predict\_POS}(\mathbf{x}; \mathbf{w}) = \arg\max_{\mathbf{y}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})$.

You will be using an off-the-shelf Python library **PyStruct** to solve the structured SVM. The installation instructions and documentation can be found at `http://pystruct.github.io`

As we discussed in class, structured SVM has several advantages including a clear objective function (based on hinge loss, which is an upper bound on 0-1 loss) and max-margin regularization.

1. Learn a structural SVM to model the above POS-tagging task using PyStruct. To help you get started, we have provided you with a skeleton code `sample_code.py` that reads in the training data, learns a structural SVM on a subset of the training data, and performs inference on a subset of the training data. (For more details, refer to the code.) There are two hyperparameters *max_iter* and C, that need to be supplied to `OneSlackSSVM`. For the purpose of this assignment, you can fix *max_iter* to 200. Choose C using a held-out set: learn the structural SVM using the first 4500 training samples, and use the last 500 training points as validate data. Let C^* be the optimal value of C (giving the smallest validate error) obtained from the above process. Then, use the entire training set to learn a model with $C = C^*$. Finally, report the test error (i.e., for the `text_XXX.txt` files) using the learned model.

   PyStruct by default uses margin scaling with Hamming loss. Its "score" function will report the average Hamming error (normalized by sentence length). This is the error that we want you to report in this assignment.

2. Repeat the experiment of question 1, but varying the number of training sentences to be the first $100, 200, 500$, or 1000. The value of the hyperparameter C could, in general, be different for different sizes of the training set. Hence, you should use the validate data (keep this fixed to the last 500 training points) to find the optimal value of C for each size of the training data separately. Produce one plot showing both the test and train error as a function of the amount of training data (also include in the plot the result for 4500 training sentences, as computed in the previous question).

3. For the next two questions, use the results from question 1. Out of the 10 given POS tags/classes, pick any three classes. For all ordered pairs of the three classes above, report the corresponding transition parameters. Comment on any interesting observations. For each of these three classes, which 10 features have the largest absolute value weights (i.e., contribute the most to the predictions)? You should look up the corresponding prefix and suffix IDs using the provided files `prefixes.txt` and `suffixes.txt` (do not simply report the IDs themselves, as this is not meaningful). Do the results make sense?

   The learnt parameters of the model are stored in a vector `w` of the `OneSlackSSVM` object. When training a `ChainCRF` using `OneSlackSSVM` with m features per training sample and k classes in the directed setting, the size of weight vector `w` will be $mk + k^2$, with mk features for the single-node potentials and k^2 features for the pairwise potentials. All the single-node features are followed by all the pairwise features. The single-node features have the following order in `w`: m features for the 1^{st} class/POS tag (verb), m features for the 2^{nd} class/POS tag (noun), ... As we described earlier, the pairwise features directly give the values of the pairwise potential; the order is (class 1, class 1), ..., (class 1, class k), ..., (class k, class 1), ..., (class k, class k). Thus, for our problem setting, the last 100 values of the vector correspond to the transition parameters.

4. Choose any 10 test sentences and, using the `predict` function of `OneSlackSSVM`, find the POS tags assigned by the model to the words in the sentences. Compare the predicted tags with the actual tags, and report any mismatches.

5. [**Extra credit**] The accuracy of the CRF model is limited by the features used. Consider defining some of your own features and adding them to the existing set of features. Re-run the evaluation using your new features. Can you find new features that lead to a reduction in POS tag prediction error?

   Note that the original set of features correspond to a chain-structured CRF where the edge potentials do not depend on the sentence, which is why we used the simpler `ChainCRF` model. If you want to add skip edges or parameterize the edge potentials to depend on the input, you should instead use PyStruct's `EdgeFeatureGraphCRF` model. If the model is still tree-structured, you can continue using `max-product` as the inference method. If it isn't, you can use `ad3`, which performs dual decomposition. This will be slower to train, and it is recommended that you test it on smaller training sets before applying to all of the data.

Acknowledgement: This problem set is partly based on an assignment developed at UMass Amherst by Ben Marlin, Andrew McCallum, Sameer Singh and Michael Wick.