Structured prediction: PoS tagging Twitter

Ildar Nurgaliev, Innopolis university

***Abstract*—We address the problem of part-of-speech tagging for English data from the popular microblogging service Twitter. We develop features, and report tagging results nearing 86.8% accuracy. The goal is to enable richer text analysis of Twitter and related social media data sets.**

***Index Terms*—HHM, L1, L2, PoS tagging, NLP, Machine Learning**

I. Introduction

A Part-Of-Speech Tagger (POS Tagger) is a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc., although generally computational applications use more fine-grained POS tags like 'noun-plural'. The English taggers use the Penn Treebank tag set. Moreover Social networks like a Twitter could have specific PoS tags like HT (hash-tag).

II. Problem Statement

Given a test set of sentences we have to predict PoS tag of every word in sentences.

- **Item precision** = $\frac{\wedge(W)}{W}$, where W - count of all the words and $\wedge(W)$ - count of correct words.
- **Instance precision** = $\frac{\wedge(S)}{S}$, where S is count of sentences and $\wedge(S)$ - count of correct sentences.

Our task is to maximize the objective function of precision for both - item and instance.

III. The data

We use *pos_train.conll* that has 551 sentences and *pos_test.conll* that has 118 sentences. Data format is **CoNLL**:

- One example per line (item)
- Tab-separated
- Reference label in the first position
- Other columns are features
- a sentence (instance) is divided from another by a blank line

IV. Data analysis

The analysis shows us that in training set we have 7 tags that are not presented in test set, thus we decided to do not consider these tag and remove these words from training set. {TAG[count] examples}

TRAIN $\setminus$ **TEST**:

- LS[1] ['1']
- TD[1] ['a']
- FW[3] ['Etc', 'Etc', 'Etc']
- NNPS[6] ['kids', 'Engineers', 'queens', 'Monsters', 'Eats']
- VPP[1] ['please']
- RBS[1] ['most']
- O[1] ['"..']

TEST $\setminus$ **TRAIN**:

- NONE[2] ['[', ']']
- PDT[1] ['Half']

TRAIN $\wedge$ **TEST**:

{ CD, RB, NNP, JJ, URL, JJR, VBD, CC, TO, (, ,, VBP, :, JJS, DT, NN, VBN, EX, VBG, "", RBR, RP, HT, PRP, VBZ, WP, PRP$, POS, WRB, IN, USR, ., UH, NNS,), MD, RT, VB, SYM, WDT }

V. Features generation

A word is plenty of features that should be extracted. Below we present survey of features extracted for every word of a sentence.

len	len(word)
word	word.lower()
is_hashtag	x[0] == '#'
is_user	x[0] == '@'
is_url	'http' in x or 'com' in x
is_digit	x[0].isdigit()
is_upper	x[0].isupper()
is_word	x.lower() in words
s_one_elemseq	len(set(x)) ≤ 1
s_two_elemseq	len(set(x)) $== 2$
postfix	word[-2:], word[-3:]
PoS of a word	lesk(word).pos()

After data analysis it become clear that in some cases a word tag is ambiguous and it could depend on neighboring words. Thus we should extend feature set by relative words through position of a given one. For example:

1) :]
2))]
3) NONE]

Thus we use its position in a sentence and provide the same features (the table before) for previous word **-w** and the next one **+w**

VI. Test algorithms

POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. POS-tagging algorithms fall into two distinctive groups: rule-based and stochastic. We used next 6 stochastic algorithms with different tunning options as for

example a search approach in Stochastic Gradient Descent as the next 3:

1) StrongBacktracking
2) MoreThuente
3) Backtracking

A. Limited-memory Broyden-Fletcher-Goldfarb-Shanno

lbfgs - abbreviation
Maximize the logarithm of the likelihood of the training data with L1 and/or L2 regularization term(s). Typically ridge or L1 penalties are much better for minimizing prediction error rather than L1 penalties. The reason for this is that when two predictors are highly correlated, L1 regularizer will simply pick one of the two predictors. In contrast, the L2 regularizer will keep both of them and jointly shrink the corresponding coefficients a little bit. Thus, while the L1 penalty can certainly reduce overfitting, you may also experience a loss in predictive power.

B. Stochastic Gradient Descent with L2

l2sgd - abbreviation
Maximize the logarithm of the likelihood of the training data with L2 regularization term(s) using Stochastic Gradient Descent with batch size 1. It approaches to the optimal feature weights quite rapidly.

C. Averaged Perceptron

The algorithm takes the average of feature weights at all updates in the training process. The algorithm is fastest in terms of training speed. Even though the algorithm is very simple, it exhibits high prediction performance.

D. Passive Aggressive

Given an item sequence (x, y) in the training data, the algorithm computes the loss: s(x, y') - s(x, y) + sqrt(d(y', y)), where s(x, y') is the score of the Viterbi label sequence, s(x, y) is the score of the label sequence of the training data, and d(y', y) measures the distance between the Viterbi label sequence (y') and the reference label sequence (y). If the item suffers from a non-negative loss, the algorithm updates the model based on the loss. Always is uses hinge loss.

E. Adaptive Regularization of Weights

AROW - abbreviation
Given an item sequence (x, y) in the training data, the algorithm computes the loss: s(x, y') - s(x, y), where s(x, y') is the score of the Viterbi label sequence, and s(x, y) is the score of the label sequence of the training data. The algorithm initialize a vector of feature weights as a multivariate Gaussian distribution with mean 0 and variance VALUE. Also it has tradeoff between loss function and changes of feature weights by GAMMA parameter.

VII. Results

Our evaluation was designed to test the efficacy of this feature set for part-of-speech tagging given limited training data. For studying purpose we decided to test different algorithm for PoS tagging problem with the same set of features. There we present best results of every of 6 algorithms (with tuning) described before.

A. lbfgs

	Precision
Item	0.867
Instance	0.206

B. l2sgd

	Precision
Item	0.839
Instance	0.1466

C. Averaged Perceptron

	Precision
Item	0.844
Instance	0.1552

D. Passive Aggressive

	Precision
Item	0.838
Instance	0.1638

E. AROW

	Precision
Item	0.814
Instance	0.1466

VIII. Conclusion

Every feature selected were crucial for such good result. The best score that we achieved was by tuned algorithm lbfgs(L1=0.1, L2=0.01) and additional +2% was achieved by using lesk algorithm on a word with sentence for revealing supposed part of speech tag as an additional feature. Item precision = 0.867 (1916 / 2211), Instance precision = 0.206 (24 / 116).