Computational Linguistics Assignment

Candidate 680817

January 15, 2011

Question 1 (Incremental Parsing with the Perceptron Algorithm)

Main features

The paper [1] describes a statistical approach to parsing natural language sentences based on machine learning techniques (the perceptron learning algorithm).

As with any such approach, the method requires a preliminary stage in which the system is trained on a set of examples.

In the linear model for NLP used, the question of training equates to finding the 'best' parameter vector $\bar{\alpha}$, where 'best' can be taken to mean the parameters which, when used in the incremental parsing algorithm as applied to the input sentences of the training data, will most closely produce the corresponding expected output parses.
[1]

Crucially, the examples are transformed by a *representation* function Φ, which aims to capture the significant features of a parse tree.

See below for a more detailed description of the training process.

Data Requirements

External input

The training data will consist of sentences paired with their corresponding correct parse trees.

In addition to the training data, the system will clearly also need to be given some sentences to parse.

[1] In fact, Theorem 1 of [1] states only that the training method does this when the example data is separable. However, as they note, stronger guarantees can be obtained via [**?**] and [**?**].

Internal data structures

The goal of the training stage is to produce a parameter vector containing perceptron weights. Since the only operation required on this vector is a scalar product, it can be simply stored as an array.

Although there is some flexibility in the ruleset used for classification, we shall assume that this is hardcoded - however,

Training

The representation function maps from a sentence paired with a corresponding candidate parse tree into a real vector space with dimension equal to the number of discrete features under consideration.

The parameter vector $\bar{\alpha}$ encodes the weights of a single-layer perceptron which is to be trained to separate correct parses for a sentence (as seen by the representation function) from incorrect ones.

This means that $\bar{\alpha}$ will come to encode the relative significance (positive or negative) of each feature in performing this classification.

Averaged Perceptron Improvement

The simplest training method involves repeatedly running the basic perceptron algorithm on the example data either a fixed number of times or until a set of weights is found which produces no errors on the example set. An alternative is to use an average of the parameter values, sampled after each example has been processed. Intuitively, this is a good idea since it reduces the degree to which the produced parameters depend on the ordering of the example data (which ought not to be significant).

Parsing

Parsing can be seen as the problem of searching the space of possible parse trees for a sentence to find the one which scores best according to the model trained from the example data. This space can be very large, but the perceptron approach allows for a heuristic search strategy. The parser proceeds incrementally, from left to right through the sentence. After each word, the score for each candidate parse under the perceptron model is calculated, and the candidates can then be filtered based on these scores.

In training we can think of the perceptron as a linear classifier, but for parsing we are interested in more than a simple boolean decision about whether a parse is good or not; we wish to find the *best* parse for a sentence

x. This is in fact the parse z for which the vector $\Phi(x, z)$ is closest to being perpendicular to the hyperplane which forms the perceptron's decision boundary. Hence we seek $\text{argmax}_{z \in \mathbf{GEN}(x)} \Phi(x, z) \cdot \bar{\alpha}$.

Decoding

'Decoding' is the process of finding the parse z of a sentence x which maximises the value of $\Phi(x, z) \cdot \bar{\alpha}$

The filtering process after each word has been processed means that the overall strategy amounts to a beam search: low-scoring partial hypotheses will be dropped from consideration to ensure that space costs remain manageable.

Grammar Transformations

Although using beam search makes the problem more tractable, there may still be very many possible parses to consider at each iteration of the search depending on the grammar used in the generation of candidates. The technique of *grammar transformations* is employed to ameliorate these difficulties.

Question 2 (Synchronous Context-free Grammars)

```
John eats fish
John mange du poisson

Pierre does not eat fish
Pierre ne mange pas du poisson

Maria likes green apples
Maria aime les pommes vertes

James thinks that Maria eats snails
James pense que Maria mange des escargots

Proper nouns


Masculine noun
MN -> <fish, poisson>
```

```
Plural noun
NS -> <apples, pommes>
NS -> <snails, escargots>

NN -> <PN[], PN[]>
NN -> <MN[], MDET MN[]>
NN -> <NS[], PDET NS[]>

ON -> <MN[], du MN[]>

Masculine determiner
MDET -> <the, le>

Plural determiner
PDET -> <the, les>

Adjective
JJ -> <green, vertes>

JN -> <NN[], NN[]>
JN -> <JJ[] JN[], JN[] JJ[]>

Verb present tense
VPRES -> <eats, mange>
VPRES -> <likes, aime>
VPRES -> <thinks, pense>
VPRES -> <NEGVERB[], NEGVERB[]>

Verb infinitive
VINF -> <eat, mange>

Negated verb
NEGVERB -> <does not VINF[], ne VINF[] pas>

Verb phrase
VP -> <V[] that S[], V[] que S[]>
VP -> <VPRES[] JN[], VPRES[] JN[]>

TODO ?
```

```
NP -> <NN[3], NN[3]>

NOUN_PHRASE_1 -> < NOUN_M , du NOUN_M >
NOUN_PHRASE_1 -> < NOUN_PL , des NOUN_PL >

NOUN_PHRASE_2 -> < NOUN_PL, les NOUN_PL >

NOUN_PL -> < NN_PL, NN_PL >
NOUN_PL -> < JJ_PL NOUN_PL, NOUN_PL JJ_PL >

NN_PL -> < snails, escargots >
NN_PL -> < apples, pommes >

NN_M -> < fish, poisson >


VERB_PRESENT_1 -> < does not VERB_INF_1, ne VERB_INF_1 pas >

VERB_PRESENT_1 -> < eats, mange >
VERB_PRESENT_2 -> < likes, aime >

VERB_PHRASE -> < VERB_PRESENT_1[3] NP_1[4], VERB_PRESENT_1[3] NP_1[4] >
VERB_PHRASE -> < VERB_PRESENT_2[3] NP_2[4], VERB_PRESENT_2[3] NP_2[4] >

PROPER_NOUN -> < John,   John   >
PROPER_NOUN -> < Pierre, Pierre >
PROPER_NOUN -> < Maria,  Maria  >
PROPER_NOUN -> < James,  James  >
S -> < PROPER_NOUN[1] VERB_PHRASE[2], PRPOER_NOUN[1] VERB_PHRASE[2] >
```

Question 3 (Sentiment Analysis and Opinion Mining)

Overview

The term 'sentiment analysis' refers to the task of automatically determining, given a text which expresses some opinion on some subject, to what degree that opinion can be considered positive or negative. This concept can be applied on the level of whole documents or to individual sentences, and naturally the two problems are closely related. The following explanations focus more on the problem at the document scope, since that is the problem that is most relevant for assessing the favourability or otherwise of online product reviews.

Methods

'Bag of words'

Perhaps the most obvious technique that is applicable is that of the 'bag of words', in which the words contained within the document are considered without regard to their ordering; instead it is their frequency which is used to determine a result. The system will include a lookup table which lists a 'polarity' value for some set of significant words, which value represents the degree to which the corresponding word should be considered positive or negative.

For instance, in a system where 1.0 is maximally positive and -1.0 is maximally negative, the word 'good' might be assigned the value 0.9 while neutral words like 'the' would be 0.0. This table could be constructed manually, but a better approach would be to learn it from a set of example documents whose correct sentiment is already known. Word-polarity learning is a more widely applicable problem, and is discussed later.

Assuming the word polarities are already available, it is simple to combine these with the word frequencies in the document under consideration to produce an overall sentiment value.

Formally, if W is the set of words in the document, $f : W \to \mathbb{N}$ maps words to their frequencies, and $\Phi : W \to [-1, 1]$ their polarities, then the sentiment value is given by $\alpha = \sum_{w \in W} f(w) \cdot \Phi(w)$.

Note that this can be considered as a scalar product in a real vector space, and is in fact analagous to the linear NLP model used in question 1.

The main advantage of the bag-of-words is its simplicity - while it can be a good starting point for building more advanced techniques for sentiment

analysis, it is rather nave by itself. Since words are considered individually, the phrase 'not good at all' might falsely be taken to reflect positive sentiment, for instance.

Learning word polarities

References

[1] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics - ACL '04*, pages 111–es, 2004.