Comparing classification algorithms for chunking

Efraim Laksman
efraim.laksman.884@student.lu.se

Abstract

We will consider an algorithm for chunking text, classifying tags for words as a way of specifying the chunks by using a decision tree (trained by the algorithm known as J48). We will estimate the chunks going from the beginning of the text to the end of the text (forward) meaning that previously estimated tags can be used as attributes for estimation of other tags. We will analyse the effects of using different attributes as parameters for the estimation.

1 Credits

This article, and the project on which it is based were made as examination for the course DAT171 at Lunds university, Lund. I have throughout the project received valuable help and comments from Richard Johansson, PhD student at LTH, Lund.

2 Introduction

What is chunking?

Computational linguistic chunking (there are other meanings of the term chunking) is a way to analyse a sentence by dividing it into as few parts possible such that all the words in any of the parts are consecutive and refer to the same entity (object, event, relation, etc. . .).

We will take an example from the corpus that I've used:

[The investment **NP**] [is **VP**] [worth **ADJP**]

[about $ 130 **NP**] [today **NP**].

While the chunks are uniquely determined by the sentence, the data structure by which we store them is not. In the example above, we have tagged the spaces between words. Each space has been tagged with either a right bracket and some letters identifying the type of chunk (stating the end of a chunk), no tag at all (stating that the current chunk stretches over the next word as well) or a left bracket (stating the beginning of the sentence). Observe that before storing the tags, all left brackets except for the first one in the sentence would be removed, as they are redundant (and they are therefore not mentioned in the description above).

What data structure to use?

What data structure one ought to choose depends on how one's classifier (the program that gives an estimate of the chunk tags for a text) works. Naturally, one can choose any structure and alter it appropriately before and after chunking, but we save work by choosing an appropriate structure from the start.

I will tag words, rather than spaces (as was done in the example above). All words already carry tags (their POS), so this is a natural way to do it, but as there is (almost) one–to–one correspondence between words and spaces, this is of little importance, except for our interpretation of the tags.

When attempting to classify the chunk for a given word, w_i, my chunker (the most powerful of the chunkers I'll use, that is) will consider words in the proximity of w_i, i. e. $w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}$, where w_{k+1} is the word directly following w_k. It will also use any other information attached to these words, i. e. POS tags and chunk tags (in the cases where they have been assigned). Whether we will be able to read the chunk tags of the present words predecessors or successors will depend on whether we

work forward (from left to right, giving us access to our estimates of chunk tags of predecessors) or backward (from right to left, giving us access to our estimates of chunk tags of successors).

In the example above, information that one chunk ended and another chunk begun was stored between the chunks. As we tag words, instead of spaces, this information will be "pushed" either to the first word of a chunk or to the last word of a chunk. (NB! We don't want to push this information in both directions as this would result in more types of tags than otherwise needed, and as machine learning techniques will be used, this would result in the need of a larger training set.) Which of these is the best depends on whether we chunk forward or backward. As I'll chunk forward, I'll push the information of where a chunk ends and a new chunk begins to the first word of a chunk.

We consider the example above once more and observe that information about the type of a chunk was stored only at one place (end of the chunk, could easily have been placed at beginning of chunk instead). This has a downside. Should we have a large chunk, looking only two words back (as I'll do when I chunk forward), I may be able to determine that the three previous words are part of one chunk whithout knowing the type of this chunk. In order not to lose information in this manner, every word will be tagged with information of what type of chunk it belongs to.

Thus, a chunk tag will state whether the word is inside or at the beginning of a chunk, as well as what type of chunk it is. We repeat the example we had above:

The DT B-NP

investment NN I-NP

is VBZ B-VP

worth JJ B-ADJP

about RB B-NP

$ $ I-NP

130 CD I-NP

today NN B-NP

. . O

(The last tag stands for "outside", not part of any chunk.)

3 Technicalities

The corpus

The corpus we will use will be the same as was used in CoNLL–2000, the Wall Street Journal corpus, section 15–18 for training purposes and section 20 for testing purposes.

The algorithm

In order to classify the chunk tag for one word, we consider the word itself and it's POS, the two following words and their POS and the two previous words, their POS and our classification of their chunk tags. We denote these attributes $w_{i-2}, \ldots, w_{i+2}$, $t_{i-2}, \ldots, t_{i+2}$ and c_{i-2} and c_{i-1} where w_k denotes the k:th word, t_k denotes the POS tag of the k:th word and c_k denotes the chunk tag of the k:th word. Observe that words that are unusual (I've decided to call words that appear less than 100 times unusual, though there are other, more reasonable ways to set a limit) are exchanged to some tag, in order to reduce time required to train the decision tree. We then train a decision tree by feeding it a trainingset consisting of a subset (not neccessarily proper) of the given attributes along with correct classification. As a training algorithm for the decision tree, we will use J48 which works roughly as ID3, i. e. maximises entropy gain when choosing what attribute to branch on next. By using different subsets of the given attributes, and using the resulting decision tree on a testset, we will be able to analyse the effect of different attributes.

Comparison

We need to decide what measure we will use when stating that one decision tree is better than another one. We will use the F–measure, defined as

$$F = \frac{2PR}{P+R}$$

where P stands for precision and R stands for recall. For any given chunk C, the precision is

$$P = \frac{\text{number of words we correctly tagged as } C}{\text{number of words we tagged as } C}$$

and the recall is

$$P = \frac{\text{number of words we correctly tagged as } C}{\text{number of words that should be tagged as } C}.$$

Observe that the measure of a correctly tagged text is 1 and that the measure of a text with only flawed

tags is 0. We are thus interested in models (sets of attributes) resulting in high F-measures.

When comparing two models, the difference in F–measure does not reveal the full story. For example, a model with F–measure 0.51 is only slightly better than one with the F–measure 0.50, while a model with F–measure 0.99 is a great improvement over a model with F–measure 0.98. Therefore, when comparing two models with eachother, we will be interested in the error reduction we get when exchanging the poorer model, m_1, for the better one, m_2. We define error reduction as:

$$e(m_1, m_2) = \frac{F(m_2) - F(m_1)}{1 - F(m_1)}.$$

4 Results

Limitations

We start by finding some limits for our results by testing the smallest reasonable model (lower limit) and the largest model we intend to use (upper limit).

The smallest model will only use the attribute t_0, i. e. the POS of the word for which we attempt to estimate the chunk. This is often referred to as the baseline, the simplest model anyone would ever use. The F–measure turns out to be 0.7707.

The largest model we will use will contain all 12 previously mentioned attributes, and since no model will use any attribute not used in this model, we may well expect that this model will have the highest F–measure, which is 0.9055. As we will see, this can be surpassed slightly.

Windows

By a window, we mean a distance (not neccessarily the same distance forward as backward) from the word we're attempting to classify beyond which we do not venture when choosing attributes to use for the classification. We have already specified that we will stay within the window ranging from -2 to 2 (we will denote this model $W_{[-2,2]}$). We will compare this with the models $W_{[-1,1]}$, $W_{[-2,0]}$, $W_{[-1,0]}$ and $W_{[0,0]}$ (in all these cases we will use as many attributes as possible, staying within the given window). We get $F(W_{[-2,2]}) = 0.9055$, $F(W_{[-1,1]}) = 0.9032$, $F(W_{[-2,0]}) = 0.8672$, $F(W_{[-1,0]}) = 0.8665$ and $F(W_{[0,0]}) = 0.7901$. We immediately observe that there is at least one attribute indexed -1 and at least one attribute indexed 1 that are of great importance for correct classification. The question remains as to whether or not there is some important attribute indexed -2 or 2 or not. $e(W_{[-2,2]}, W_{[-1,1]}) = 0.0237$, and 2% error reduction is not insignificant. To use 5 additional attributes to get this error reduction does however seem somewhat expensive. We'll look into whether the information needed to get the error reduction is spread evenly among these 5 attributes, or if at least some of them may be omitted.

We let $W_{[-2,2]-w_{-2}}$ denote the model using all attributes in the window ranging from -2 to 2 except for w_{-2} (and similarly when w_{-2} is exchanged for some other attribute), and get $F(W_{[-2,2]-w_{-2}}) = 0.9043$, $F(W_{[-2,2]-w_2}) = 0.9054$, $F(W_{[-2,2]-t_{-2}}) = 0.9057$, $F(W_{[-2,2]-t_2}) = 0.9039$ and $F(W_{[-2,2]-c_{-2}}) = 0.9048$. What strikes us first is that we in one of these cases actually surpassed the F–measure which we set as an upper limit. This is the result of overtraining the tree when using the "full" model. As it turns out, $F(W_{[-2,2]-w_2-t_{-2}}) = 0.9059$, a better result still. In theory, by letting the size of our training corpus approach infinity, this would not occur, as insignificant attributes would simply not be used. With our limited corpus however, it is possible that insignificant attributes can have a negative effect (using a better tree training algorithm would usually solve this problem, but require more time). Either way, what we see is that the attributes w_2 and t_{-2} probably do more harm than good.

We will attempt a similar analysis of the model $W_{[-1,1]}$. We already know that the attributes w_{-2}, t_2 and c_{-2} aid the process of classification, so based on the assumption that the covariance between words decrease as the distance between them increase, we may draw the conclusion that the attributes w_{-1}, t_1 and c_{-1} are beneficial. We compute the F–measures $F(W_{[-1,1]-w_1}) = 0.9049$ and $F(W_{[-1,1]-t_{-1}}) = 0.8939$. While t_{-1} is clearly an important attribute (error reduction of using this attribute rather than not using it is almost 10%, our model improves by removal of the attribute w_1.

There is yet (at least) one more model worthwhile trying, namely $W_{[-2,2]-w_1-w_2-t_{-2}}$. We find that $F(W_{[-2,2]-w_1-w_2-t_{-2}}) = 0.9053$, i. e. not an improvement over the model $W_{[-2,2]-w_2-t_{-2}}$. However the error reduction for using the best model we have had so far rather than this

model, $e(W_{[-2,2]-w_1-w_2-t_{-2}}, W_{[-2,2]-w_2-t_{-2}}) = 0.0063$, is really small, almost insignificant.

We conclude this section about windows by stating that while exact choices of which attributes to use are dependent on problem specifications (such as limits on model size), sticking to "full" windows models is probably a sub–optimal approach, even if it may appear to be a "natural" choice.

Illiterate models vs. literate models

While we have in the section about windows probably built as good models as is possible within the bounds of construction we have set for ourselves, we are not quite done with our analysis of the importance of the different attributes.

Inspite of that we throw all unusual words (in our case meaning words that appear less than 100 times in the training corpus) in one big heap, the number of distinct words is large, larger than the number of POS or chunk tags. Indeed, the attribute would be almost useless if it did not allow for many distinct words. While taking the attributes w_i (for suitable values of i) into account (giving us a literate model) may improve on the results of classification, it is costly both in terms of the size of the model and in terms of time required to train it. Illiterate models (not using the attributes w_i at all) may therefore be of some interest.

When we looked at the "full" windows models, the models that appeared to be most useful were $W_{[-2,2]}$ and $W_{[-1,1]}$, which is why we will compare these models with their illiterate "counterparts", $W_{[-2,2]-L}$ and $W_{[-1,1]-L}$, to get an idea of the effects of literacy.

For the larger model we get $F(W_{[-2,2]-L}) = 0.8867$ and $e(W_{[-2,2]-L}, W_{[-2,2]}) = 0.166$ and for the smaller model we find $F(W_{[-1,1]-L}) = 0.8836$ and $e(W_{[-1,1]-L}, W_{[-1,1]}) = 0.168$. As we can see, the actual words have large impact on the results, and should probably not be discarded. How to set the limit for which words that are to be discarded (or thrown into the heap of unusual words) may still be of interest, but it isn't anything that I will deal with here.

Static models vs. dynamic models

Finally, we will consider how well static models (models that don't make use of previous estimates, i. e. the attributes c_{-2} and c_{-1}) hold up against dynamic models (models that make use of previous estimates). After all, using dynamic models creates risk of error propagation (one erroneus classification causing errors several steps ahead), even if this propagation would be limited to stay within a sentence.

We compare $W_{[-2,2]}$ to it's static counterpart. This yields $F(W_{[-2,2]-D}) = 0.9025$ and $e(W_{[-2,2]-D}, W_{[-2,2]}) = 0.031$,where $-D$ in the index indicates that the model is static. While a dynamic model certainly is an improvement over a static model, it is nowhere near the importance of having a literate model rather than an illiterate one. If demands on the models correctness are somewhat lax, while demands on speed and model size are strict, using static models may be a reasonable choice.

References

No reference literature was used.