Thesis

Philip White

November 20, 2011

1 Parsing and Classification

1.1 Choice of Language

With very few exceptions, the code I wrote in support of this thesis was done in Clojure, a dialect of LISP designed to work on top of the Java Virtual Machine (JVM). The choice of a language was easy: a heavy dependence on the Stanford Parser and the WEKA package, both written in Java, necessitated a JVM-based language. The slowness of Java's compile/debug cycle eliminated that language as an option, leaving a handful of possible languages, from which I chose Clojure for its speed, functional style, and elegance.

1.2 Parsing

The Stanford Parser software package, version 1.6.7, configured with the probabilistic context-free grammar (PCFG) [Klein and Manning 2003], was used to generate all syntactic parse trees and grammar dependency graphs. In brief, PCFGs have their origins in the work of

1.3 The Tests

The crux of this project was the design and creation of a suite of tests, each of which identifies a number of closely related grammatical characteristics of the text samples. These tests operate on the output from the Stanford parser, i.e. parse trees and grammatical dependencies. As output they generate training or testing cases to be used by the Weka classifier. Each of these cases consists of multiple attributes, corresponding to grammatical features, each with continuous values indicating the relative frequency (probability) of that particular feature. For a case with n attributes where the number of occurrences of the grammatical feature associated with the ith attribute is g_i, the value f_i for that attribute is given by $g_n / \sum_{i=1}^{n} g_i$. For instance, one test measures the relative frequencies of the various tense/aspect/voice combinations of finite verbs. English has twenty-four such combination, so the case generated by this test has twenty-four attributes.

In addition to the attributes, each case has a class which can be *es* or *en*, indicating that the class is associated with a text sample written by an L1-Spanish speaker or by a native English speaker, respectively. For training cases, the classes are known beforehand and are assigned to the cases manually. For testing cases, the classes have missing values, until such values are determined by a classifier, as discussed in the following section.

1.4 Classification

I used the Weka machine learning package, version 3.6 [Hall et al. 2009], to create, train and test classifiers based on the cases discussed above. I primarily used two classifiers: J48, which is Weka's implementation of the C4.5 classifier [Quinlan 1993] and the RandomForest classifier, which is based on the random forest algorithm described by Breiman [2001]. The former is useful for its highly readable decision trees, which clearly indicate which attributes are involved in the classification and their roles. In later sections of this paper are found linguistic explanations for why these particular attributes should be useful in classification.

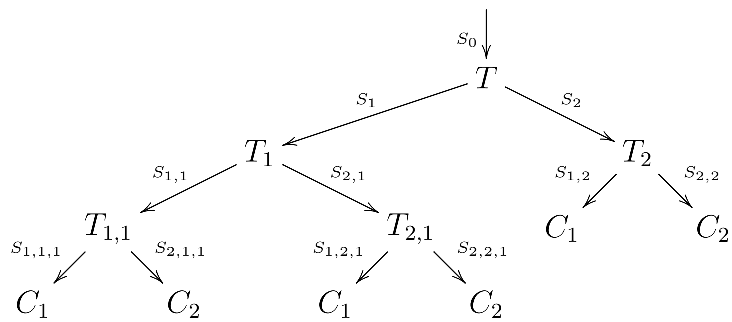

Figure 1.1: A decision tree showing the partitioning of the set of all training cases S into subsets $S_{1,2}$, $S_{1,1,1}$, and $S_{1,2,1}$ whose elements are of class C_1, and $S_{2,2}$, $S_{2,1,1}$, and $S_{2,2,1}$ whose elements are of class C_2. The nodes T, T_1, etc. are partitioning operations.

1.4.1 C4.5

This section describes the C4.5 partition as it applies to this project. That is to say, C4.5 can deal with a number of circumstances that do not arise here. What is described here is a version of the C4.5 algorithm that is restricted to continuous attribute value and to exactly two class values, and which does not permit missing attribute values. That having been said, the C4.5 algorithm consists of two phases, *tree construction* and *tree pruning*.

In the tree construction phase a decision tree is built which successively performs binary partitioning of a set of training cases (see fig. 1.1). Let us consider a full binary tree where each edge represents a set of cases and each non-terminal node a partitioning operation. These partitioning operations take one set, represented by the parent edge, and divide it into two subsets, the daughter edges. The root node operates on an initial set S_0, and a leaf node simply indicates that its parent edge is a set consisting of cases of a single class. Let the first partitions of S_0 be called S_1 and S_2, and of S_1 let them be called $S_{1,1}$ and $S_{2,1}$ and so forth. Likewise, let the partitioning operation that operates on a particular set be designated by T with the same subscripts as that set.

The partitioning operations are performed by applying a binary test to each case within S, the set to be partitioned, and dividing the set based on the results. Each test considers a single attribute A and compares that value to a threshold value. All case where .

Table 2.1: Relation abbreviations

advmod	adverbial modifier
aux	auxiliary
complm	complementizer
mark	marker
prt	phrasal verb particle
purpcl	purpose clause modifier
rcmod	relative clause modifier

into a number of subsets, each containing cases of a single class.

2 Grammatical Relations

The simplest classification approach considered the relative frequency of different grammatical relations. For this approach, the governor and the dependent of the dependencies were ignored, with only the relation itself being used.

Each data set instance contained 52 numerical attributes, one for each relation in the Stanford Dependency system. For each attribute A_r corresponding to the relation r, the corresponding value was n_r/n_t, where n_r and n_t were the number of occurrences of the relation r and the total number of relations in the text, respectively.

A C4.5 decision tree classifier trained on these instances produces the decision tree shown in Algorithm 2.1. The full names for the seven relations are shown in Table 2.1. The following subsections explore the linguistic reasons why these particular relations should be so useful in classifying the texts.

Algorithm 2.1 C4.5 decision tree classifier

```
if complm ≤ 0.011635 then
    if purpcl ≤ 0.000856 then
        if rcmod ≤ 0.012254 then
            en(34.0)
        else
            if prt ≤ 0.002113 then
                es(4.0/1.0)
            else
                en(6.0)
    else
        if purpcl ≤ 0.001191 then
            if advmod ≤ 0.045825 then
                es(6.0)
            else
                en(2.0)
        else
            en(7.0)
else
    if mark ≤ 0.00808 then
        en(6.0)
    else
        if aux ≤ 0.044037 then
            en(6.0/1.0)
        else
            es(60.0)
```

2.1 Adverbial Modifier

2.2 Auxiliary

2.3 Complementizer

A complementizer is a word that signals the beginning of a clausal complement. The Stanford Parser recognizes the complementizers *that* and *whether* as shown in example 2.1 (Sulec,Micusp). The governor of a complementizer dependency is the root of the clause, which is generally a verb or, in the cause of copular clauses, the subject complement. The dependent is the complementizer itself.

complm

(2.1) a. … I will consider … whether the world is a safe place

b. At least you choose whether to go to a pub or not.

c. They state that climate generally predicts that temperatures should rise ...

Whitley [1986] points out that while English tends to allow complementizers introducing clausal complements in the object position to be deleted, Spanish is much more restrictive in this regard (see examples 2.2 and 2.3). [Whitley 1986, p.278].

(2.2) a. I say that he'll do it.

b. I say he'll do it.

(2.3) a. Digo que lo hará.

b. *Digo lo hará.

2.4 Marker

2.5 Phrasal Verb Particle

The phrasal verb particle relation ties the head word of a phrasal verb to its particle as shown in Example 2.4.

(2.4) ...the reduction of superfluous proteins will free up resources ...

2.6 Purpose Clause Modifier

2.7 Relative Clause Modifier

References

BREIMAN, L. 2001. Random forests. In *Machine Learning*. 5–32.

BUTT, J. AND BENJAMIN, C. 2004. *A New Reference Grammar of Modern Spanish* Fourth Ed. McGraw-Hill.

DE MARNEFFE, M.-C., MACCARTNEY, B., AND MANNING, C. D. 2006. Generating typed dependency parses from phrase structure parses. In *LREC 2006*.

DE MARNEFFE, M.-C. AND MANNING, C. D. 2008. Stanford typed dependencies manual.

HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., AND WITTEN, I. H. 2009. The WEKA data mining software: An update. *SIGKDD Explorations 11,* 1.

KLEIN, D. AND MANNING, C. D. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*. 423–430.

MOORE, A. W. 1994. Efficient algorithms for minimizing cross validation error. In *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 190–198.

QUINLAN, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

WHITLEY, M. S. 1986. *Spanish/English Contrasts: A Course in Spanish Linguistics*. Georgetown University Press.

WITTEN, I. H. AND FRANK, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques* Second Ed. Morgan Kaufmann.