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

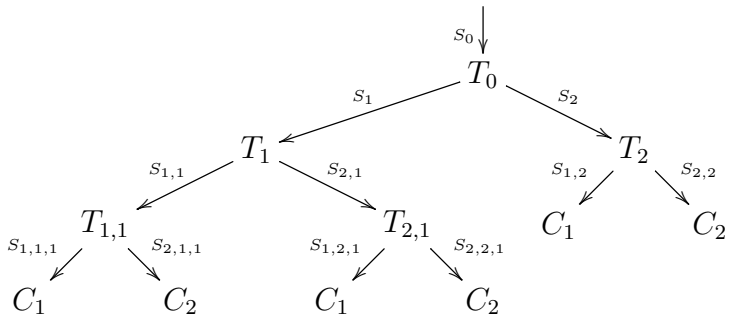
I used the Weka machine learning package, version 3.6 [Hall et al. 2009], to create, train and test classifiers based on the cases discussed above. I primarily used two classifiers: J48, which is Weka's implementation of the C4.5 classifier [Quinlan 1993] and the RandomForest classifier, which is based 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.

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 values 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. Consider a full binary tree where each edge represents a set of cases and each non-terminal node a partitioning operation, as shown in Figure 1.1.

Figure 1.1: A decision tree showing the partitioning of a set of training cases S_0 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_0, T_1, etc. are partitioning operations such that for any operation T operating on a set S_a the generated sets are $S_{1,a}$ and $S_{2,a}$ where $S_{1,a} \cup S_{2,a} = S_a$ and $S_{1,a} \cap S_{2,a} = \emptyset$.



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 where $S_1 \cup S_2 = S_0$ and $S_1 \cap S_2 = \emptyset$, 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 the value of that attribute, V_A, to a threshold value, V_C. All cases where the $V_A \leq V_C$ will be put into one subset and all other cases into the other.

The decision of the attribute and threshold value for a particular test is determined using what Quinlan calls the "gain ratio criterion" which is calculated as follows. If the probability of randomly drawing a case of class C_1 from a set S is p_1 and of drawing a case of the other class is p_2 where $p_2 = 1 - p_1$, then the average amount of information needed to identify the class of a case in S can be defined in terms of entropy as

$$\text{info}(S) = -p_1 \cdot \log_2(p_1) - p_2 \cdot \log_2(p_2).$$

A similar measure can be applied to the two partitions S_1 and S_2 created by applying the partitioning test T to S. The entropy after partition is given by taking a weighted sum of the entropy of the two sets as

$$\text{info}_T(S) = \frac{|S_1|}{|S|} \cdot \text{info}(S_1) + \frac{|S_2|}{|S|} \cdot \text{info}(S_2)$$

The decrease in entropy, expressed as a positive value (an information gain), due to partitioning S using the test T is then

$$\text{gain}(T) = \text{info}(S) - \text{info}_T(S).$$

Maximizing this gain can be and, in ID3 the predecessor to C4.5, was used as measurement of test fitness. However, in the more general case of C4.5, where one test can partition a set into more than 2 subsets, using this gain criterion to choose tests favors tests that partition sets into numerous subsets. To mitigate this, Quinlan added another factor to the criterion, the split info which for this special case is given by

$$\text{split info}(T) = -\frac{|S_1|}{|S|} \cdot \log_2\left(\frac{|S_1|}{|S|}\right) - \frac{|S_2|}{|S|} \cdot \log_2\left(\frac{|S_2|}{|S|}\right).$$

Then the fitness of a test T can be measured using

$$\text{gain ratio}(T) = \frac{\text{gain}(T)}{\text{split info}(T)}$$

It should be noted that in this special case where partitioning operations are always binary, the gain ratio criterion favors tests that split S into disparately sized sets, as split info is at its maximum (unity) when $|S_1| = |S_2|$.

In choosing a test T, the C4.5 algorithm tries each attribute A from the set S of cases to be partitioned. For each, it orders the cases in S on the value of A. If the values of A

corresponding to this ordered set are $\{v_1, v_2, \ldots, v_m\}$, then any threshold between v_i and v_{i+1} will result in the same partitions. From this it can be seen that the total number of possible partitions is $m - 1$. The algorithm tries all such partitioning schemes, measuring the gain ratio of each. When an optimal attribute and corresponding partitioning scheme has been chosen, the algorithm than chooses a threshold value that will produce this result. Again, to partition S into two sets where the values for A are $\{v_1, v_2, \ldots, v_i\}$ and $\{v_{i+1}, v_{i+2}, \ldots, v_m\}$, a threshold value v_C must be chosen such that $v_i \leq v_C < v_{i+1}$. For this, it chooses the largest value for A from the entire training set S_O that does not exceed the midpoint of this range.

References

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

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.

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