Computer Science 401 10 January, 2010
St. George Campus University of Toronto

Homework Assignment #1
Due: Fri, 5 February, 2010 at 12 PM

Genre Classification

TA: Siavash Kazemian

1 Introduction

This assignment will give you experience in working with corpora, particularly the Brown Corpus, in Python programming, in part-of-speech tagging, and in building decision trees using the C4.5 package.

Your task is to break the texts of the Brown Corpus into sentences, tag them with the Brill tagger (a popular part-of-speech tagger), gather some statistics on the texts, and then use these statistics to build a decision tree that classifies texts according to genre (as defined by the Brown Corpus genre codes).

You should check the course bulletin board, *http://csc.cdf.toronto.edu/bb/YaBB.pl?board=CSC401H1S*, frequently for announcements and clarifications pertaining to this assignment.

2 The Brown Corpus

The Brown Corpus was intended to be a representative "snapshot" of American English in 1961. We will use 495 texts from this corpus, each of which is located on CDF in a separate file, */u/cs401/Brown/corpus/Xnn*, where X is a letter indicating the subgenre of the text (see table 1), and nn is a two-digit number.

Each text is (just over) 2000 words long. The first 15 characters of each line give the text number (same as the file name), along with the word offset in the file rounded to a multiple of 10 and an additional offset mod 10. There are then four spaces, and possibly additional spaces to indicate the start of a paragraph or an indent. Note that there is no special marking of periods that don't end sentences. Here is an example, where ␣ indicates a space:

```
E02␣1590␣␣2␣␣␣␣␣␣␣Dr.␣Wilson␣C.␣Grant,␣of␣the␣Veterans'␣Administration
```

The corpus occasionally contains the 'word' `**f`, which denotes a scientific formula or other mathematical object that can't be reproduced in ASCII characters. Such 'words' should be left in the corpus, even though the tagger doesn't handle them very well, because syntactically they take the place of the part of speech that the original formula had.

3 Your tasks

An overview of the assignment is given in Figure 1.

Note that the files that you create for the Brill tagger will add up to about 6 MB, and the tagged files will add up to about 9.5 MB. While the CDF disk quota will be increased for students in this course, you will have to be careful about how you use the space. Do not copy the original Brown corpus (which is about 7.8 MB); use it directly from */u/cs401/Brown/corpus*.

1. Pre-processing and finding sentence boundaries [20 marks]

The Brill tagger that you will use in the next task requires its input data to be presented one sentence per line, with special treatment of punctuation and clitics. So first you must write a Python program, named *Brown2Brill*, written in accordance with the "General specifications" section below, that will take a Brown file in its present form and convert it to the format that the Brill tagger requires:

- Each line of the file must contain exactly one sentence (no more, no less!).
- Each token, including punctuation and clitics, must be separated by spaces. (Clitics in English are contracted forms of words, such as *n't*, that are concatenated with the previous word. Note that the possessive *'s* has its own tag and is distinct from the clitic *'s*, but nonetheless must be separated from the preceding part of the word; likewise, the possessive apostrophe on plurals.)
- However, periods in abbreviations such as *U.S.* and *etc.* count as part of the word, not as punctuation, and the same is true of hyphens in words such as *well-balanced.*

Table 2 shows an example of some original Brown text and its conversion to the input format for the Brill tagger.

The main complication is determining exactly where each sentence finishes. Manning and Schütze (section 4.2.4) suggest a number of heuristics for this, which you should include in your program. A list of common English abbreviations for use with the heuristics is available to you in */u/cs401/Wordlists/abbrev.english*. You may copy and modify this file.

Once you have debugged your program, run it on all the files of the Brown Corpus. Since you probably don't want to sit at a workstation running the same program nearly 500 times on different files, we have provided a script named *RunBrown2Brill* to do the work for you in the directory */u/cs401/Scripts*; copy it to your working directory, but do not modify it. This script takes two directory names, and runs *Brown2Brill* on each file in the first directory, depositing the output in the second directory with *.b2b* appended to the filename. For example:

```
RunBrown2Brill /u/cs401/Brown/corpus myb2bfiles_dir
```

(Make sure that the output directory exists before running the script.)

2. Part-of-speech tagging [5 marks]

The next step is to tag each word in each file with its part of speech. You will use the popular Brill part-of-speech tagger (developed by Eric Brill at Microsoft Research) as a black box. (Later in the course, we will be studying how part-of-speech taggers work.) The Brill tagger is available on CDF in the directory */u/cs401/Brill.* It uses the Penn tagset (see tables 3a and 3b below; *cf* Manning and Schütze's tables 4.5–4.6). Table 2 shows an example of the output of the tagger.

The executable of the Brill tagger is in the directory */u/cs401/Brill/Bin_and_Data*; it is invoked by the following command:

```
tagger LEXICON filename BIGRAMS LEXICALRULEFILE CONTEXTUALRULEFILE
```

where *filename* is your pre-processed file to be tagged—the tagger does *not* read from *stdin.* The other (uppercase) filenames are the names of files of information that the tagger uses, which are also in */u/cs401/Brill/Bin_and_Data.* If you execute the tagger from your working directory, you'll need to specify complete paths for the tagger and its information files; if you execute the tagger from its own directory, you'll need to specify a complete path for your input and output files. Either way, you'll probably want to make an alias for this command to save yourself a lot of typing when you are debugging. The tagger sends its output to *stdout.*

Run the Brill tagger on each of your pre-processed Brown Corpus texts to produce a set of files of tagged texts ready for the next stage. Again, we have provided a script, *RunTagger*, to do this for you; it is in */u/cs401/Scripts*. Because there is considerable overhead in each run of the tagger, this speeds up the work a lot by tagging texts in batches. (Tagging each file separately takes approximately 7 seconds of CPU time, almost all of which is initialization of the tagger; this works out to about an hour for 500 files. By batching the files, the job can be done in about 10 minutes.) This script works by concatenating the text files, inserting a marker before each file so that the output can be split back into separate files after tagging.

```
RunTagger myb2bfiles_dir mytaggedfiles_dir
```

The files in the input directory will be combined and run through the tagger; the results will be split and placed in the output directory *mytaggedfiles_dir* (which must exist before the script is run) as files with the extension *.tag*. The script *RunTagger* uses the script *splitre*, also in */u/cs401/Scripts*.

3. Gathering statistics [20 marks]

The third step is to write a Python program named *GatherStats*, written in accordance with the "General specifications" section below, that will take one of the tagged texts and compute the statistics listed in table 4 (see below for explanations). The output will be a single line of numbers, separated by commas, like this:

```
E02, 24.50, 12.0, ..., 23.67, 5.84, 0.56, II
```

The first field is the Brown Corpus file name, and the last field is the Brown genre code for this text—I, II, III, or IV, as defined in table 1. The statistics must appear in the same order as they are listed in table 4 and be rounded to two decimal places.

Run your program on each of the tagged texts to produce a single file named *BrownStats.data* that contains the results from all the texts, one line for each text. Again, we have provided a script, *RunGatherStats*, to do the work for you in the directory */u/cs401/Scripts*. This script takes a directory name, and runs *GatherStats* on each file in the directory, appending the output to a file whose name must be specified as its second argument. For example:

```
RunGatherStats mytaggedfiles_dir BrownStats.data
```

The statistics to be gathered

Most of the statistics listed in table 4 are the frequency, per thousand tokens of text, of tokens that have certain characteristics; these can be determined by looking at the token, its tag, or both. Table 5 gives definitions of some of the terms that are used; the lists in the definitions are available on CDF in the directory */u/cs401/Wordlists*, along with some other useful resources such as common given names and surnames. You may copy and modify these files, but do not change their filenames. For the public and private verbs and verbs of saying, you should recognize all inflected forms—for example, `address/VB`, `addressing/VBG`, and `addressed/VBD`—so you will need to expand the lists accordingly. The easiest way to do this is with editor macros or a small Python script, followed by manual editing to deal with exceptional cases (a more-principled method would be to add lemmatization to your program, but this requires far more work for no extra payoff).

The type–token ratio is the number of different types of words in the text divided by the total number of tokens (not counting punctuation tokens in either case). For simplicity, we regard two tokens as being the same word-type if they match by simple case-independent string-equality, without regard to their tags or any lemmatization; for example:

Same word-types:	`Horse/NN, horse/NN, horse/VB`
Different word-types:	`horse/NN, horses/NNS, horsing/VBG`

The type–token ratio will always be a real number in the interval (0,1].

4. A statistical report on the Brown Corpus [10 marks]

After you have run *GatherStats* on all 495 tagged texts to create the file *BrownStats.data*, write a Python program named *Report*, in accordance with the "General specifications" section below, that will take as standard input the file of statistics and produce on standard output a neatly formatted report, with column headings on each page, listing the statistics for each text, and, at the end, the average and standard deviation of each statistic over all the texts.

5. Classify new texts using C4.5 [15 marks]

Now you will use your statistics to train a decision tree to classify texts into the four major genres used by the Brown Corpus. Once you have trained your tree, you will use it to classify new texts based on their statistics.

You will use the machine-learning system C4.5 to train a decision tree to classify the Brown Corpus text. Given a training set of feature vectors, each with its correct classification, C4.5 tries to build an optimal decision tree for that classification, which is then pruned to reduce the possibility of overfitting. Your file *BrownStats.data* is in the correct format for input to C4.5. Details of how to use the C4.5 programs and their options are given in appendix 2 at the end of this handout.

First, experiment with the `-m` and `-c` options to find the best decision tree. Be sure to save the report from each run in a file (using the filenames *E1.c45, E2.c45, . . .*), so that you can hand some of them in with your results (see "Discussion" and "What to hand in", below). Do not alter them.

Notice that C4.5 tells you how well it was able to classify the training data and also any separate test data that was provided; these figures can be used to calculate accuracy, precision, and recall. So secondly, perform four-fold cross-validation with C4.5 on your training data, compute accuracy, precision, and recall for each run, and then take the average of the results of all four test runs to compute the best estimate for these three measures. It is not necessary to write a program to do these computations—a pocket calculator is a lot easier and quicker. See appendix 1 for more details of how to compute accuracy, precision, and recall. See appendix 2 for how to use the script *xval.sh* to perform N-fold cross-validation.

Finally, classify some as-yet-unseen texts by running your best decision tree on their statistics, which can be found in */u/cs401/unseen/BrownStats.cases*. You can do this using *consultall*, as described in Appendix 2. The output of this process should be a list of pairs of document identifiers and genre classifications along with confidence scores. Save this output to a file named *Z-results.c45*.

6. Discussion [10 marks]

Have a look at the decision trees that C4.5 produced in your *E*.c45* files, and determine which features of your training data are actually selected by C4.5 as relevant to the classification and which are not. Briefly (250 words or less) discuss your results. Which option settings did you use to get your best results? What variation in precision and recall scores did you witness? Which features did C4.5 use and which did it discard? Why?—that is, for each feature why was it useful or not useful for genre classification (or for what other reason might it have been discarded)? **Note:** Not every one of your experimental runs will necessarily merit discussion.

4 General specifications

As part of grading your assignment, the grader may run your programs and/or decision trees on test data that you have not previously seen, and may compare the output to standard answers. This may be done in part by Unix scripts. It is therefore important that your each of your programs precisely meets all the specifications, including its name and the names of files that it uses. **A program or tree that cannot be evaluated because it varies from specifications will receive zero marks.**

All programs: If a program uses a file, such as the word lists, whose name is specified within the program, it must read it either from the directory in which the program is being executed, or it must read it from a subdirectory of */u/cs401* whose path is completely specified in the program. Do **not** hardwire the absolute address of your home directory within the program; the grader does not have access to this directory.

All your programs must contain adequate internal documentation to be clear to the graders. (External documentation is not required.)

Brown2Brill: The program must read a Brown Corpus file and send its output to standard output (*stdout*). When you redirect the output to a file, the filename you choose should be *.b2b* appended to the input file name; for example, Brown Corpus file *A01* would produce file *A01.b2b*. (The script *Brown2Brill* follows this convention for you.) For example:

```
Brown2Brill A01 >A01.b2b
```

GatherStats: The program must read a tagged Brown Corpus file (a *.tag* file) and send its output to *stdout*. Don't forget that *stdout* may append its output to a pre-existing file. The final statistics file produced must be named *BrownStats.data*. For example:

```
GatherStats A02.tag >>BrownStats.data
```

Report: The program must read a statistics file and send its output to *stdout*. The report file produced must be named *BrownStats.report*. For example:

```
Report BrownStats.data > BrownStats.report
```

5 What to hand in

Your assignment should be submitted electronically only. You should submit:

- All your code for *Brown2Brill, GatherStats*, and *Report* (including helper scripts, if any).
- Your modified word lists that *Brown2Brill* and *GatherStats* use.
- The pre-processed Brown files *K09.b2b* and *L06.b2b*.
- The tagged Brown files *K09.tag* and *L06.tag*.
- The statistics gathered in part 3, as a file named *BrownStats.data*.
- The statistical report from part 4, as a file named *BrownStats.report*.
- The C4.5 name declarations file from part 5, *BrownStats.names*.

- Your three best decision tree results, as files named *E1.c45*, *E2.c45*, and *E3.c45* (even if they were not your first three trees).
- The C4.5 output file, *BrownStats.tree*, from your best run.
- The classifications of the unseen texts, *Z-results.c45*.
- Your discussion, *discussion.txt*.

Do **not** submit the entire pre-processed or tagged corpus!

In another file called *ID* (available on the course web page), provide the following information:

- your first and last name
- your student number,
- your CDF login id,
- your preferred contact email address,
- whether you are an undergraduate or graduate
- this statement: *By submitting this file, I declare that my electronic submission is my own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters and the Code of Student Conduct, as well as the collaboration policies of this course.*

Use the template provided on the course website for your *ID* file. **Electronic submissions missing ID files will not be marked.**

Electronic submission: The electronic submission must be made from CDF with the *submit* command:

```
submit -N a1 csc401h filename filename ...
```

Do **not** `tar` or `compress` your files, and do not place your files in subdirectories. Do not format your discussion as a PDF or Word document — plain text only.

6 Working at home

If you want to do some or all of this assignment on your home computer, you will have to do the extra work of downloading and installing the requisite software and data. You take on the risk that your home computer might not be adequate for the task. You are strongly advised to upload regular backups of your work to CDF, so that if your home machine fails or proves to be inadequate, you can immediately continue working on the assignment at CDF. When you have completed the assignment, you should try your programs out on CDF to make sure that they run correctly there. **A submission that does not work on CDF will get zero marks.**

The course web page, *http://www.cs.toronto.edu/~csc401h*, will shortly offer links to versions of Python for various systems, the Brill tagger for Unix and Windows, a gzipped file of all the Brown Corpus texts, and the lists of abbreviations and word-category definitions. It will also offer a link to the source code for C4.5; but note that it runs only under Unix (including Linux, probably), not Windows. The author, Dr Ross Quinlan, has kindly allowed us to use it in this course, but the source code may not be distributed or used for other purposes. Note that Dr Quinlan's free *c5.0* and *see5* programs are **not** suitable for this assignment.

Appendix 1: Precision, recall, and accuracy

Unfortunately, C4.5 does not compute precision and recall for you, but when it is invoked with the `-u` option (which it is when invoked by *xval.sh*), it gives you a table summarizing how the cases in the test data were classified. These figures allow you to compute precision and recall yourself. Here is an example table:

```
  (a)  (b)  (c)  (d)      <- classified as
 ---- ---- ---- ----
  132    6    9    4      (a): class I
    2   75    1    6      (b): class II
    8    9  119   12      (c): class III
    8    3    4   97      (d): class IV
```

Because the names of the classes could be very long, C4.5 uses the labels `(a)`, `(b)`, etc for the columns and rows, and maps them to class names on the side; for our classes, the mapping is obvious. The columns represent the classifications that the decision tree made on the test data, and the rows represent the correct classification. For example, this table says that 75 texts from class II were correctly classified, and 1 text from class II was incorrectly classified as class III.

In lectures, we saw precision and recall defined only for classification into two classes. The generalization to four classes is easy: For each class C, precision is the fraction of cases classified as C that truly are C, and recall is the fraction of cases that truly are C that were classified as C. In other words, precision is, for each column, the ratio of the diagonal element to the sum of the column, and recall is, for each row, the ratio of the diagonal element to the sum of the row.

Thus precision and recall have to be measured separately for each of the four classes. The sets of precision measures and recall measures can then each be averaged over the four classes to get an overall estimate of the precision and recall of the method.

On the other hand, accuracy—the fraction of correct decisions—is meaningful only at the overall level and not at the class level. This is because a correct identification in one class also counts as a correct rejection in the other classes; and a false positive in one class will be a false negative in another. Accuracy is simply the complement in 100% of the error rate reported by *c4.5* for the test data.

Appendix 2: Using C4.5

The C4.5 system, which includes programs named *c4.5, consult,* and *consultall*, is available on CDF in the subdirectories of */u/cs401/C4.5/R8.* The executables are located in the subdirectory *Bin.*

C4.5 uses a number of files whose names all take the form *filestem.extn*, where *filestem* is a prefix that you choose that is the same for all the files, and *extn* is the file type extension. For example, you might have the input files `Herbert.data` and `Herbert.names` in order to produce a decision tree in the file `Herbert.tree`.

1. Building a decision tree by running C4.5

To use C4.5 build a decision tree from a set of training data, you need to create two input files. The first of these, named *filestem*`.data`, contains your training data. Each line of the file is a set of data representing the attributes ('features' in our terminology) of one case, and the final item on the line is the correct classification of the case. (The `BrownStats.data` file that you produced above is in this exact format.)

The second file, named *filestem*`.names`, contains a description of the data in the first file. The first line gives the names of the classes (which in this assignment will be the genre codes). Each remaining line gives the name of an attribute, followed by a colon and its datatype. Allowable datatypes include `continuous` for numerical value, `ignore` for values to be ignored (such as the file name in this assignment), and a set of discrete values (not needed in this assignment).

Examples of both files are shown in table 6. More examples can be found in the subdirectory *Data.*

The command line to use is:

`c4.5 -f` *filestem*

This builds the decision tree, which is placed in the file *filestem*`.tree`, and tests it on the training data. If the `-u` option is specified and there is a test-data file *filestem*`.test`, then the decision tree is also tried out on this test-data. In addition to the decision-tree file, C4.5 outputs a human-readable form of the tree on *stdout*, along with a summary of the tests. If the `-u` option is specified, it also gives a table of correctness data from which precision, recall, and accuracy can be computed.

You should try C4.5 out on some of the sample data provided with the system. For example, try

```
c4.5 -f  /u/cs401/C4.5/R8/Data/labor-neg  -u
```

Two other C4.5 command-line options that you might want to use are `-m` and `-c`. The `-m` option specifies the minimum number of cases that any branch of the decision may have. The default is 2, but in the case of noisy data (such as in this assignment), a higher value is sometimes desirable to avoid oddly shaped trees; try changing the value to improve your results. The `-c` option specifies the confidence factor used in pruning. The default is 25%, but a smaller value, causing heavier pruning, is advisable if the error rate is much higher on test cases than predicted from the training set.

For more complete documentation on the program, see the attached *man* pages.

2. N-fold cross-validation with *xval.sh*

To perform N-fold cross-validation with C4.5, you need to partition the training data in N different ways into training and test segments, and for each of the N partitions, run C4.5 on the training segment and test it on the test segment. Since it's rather tedious and error-prone to do this by hand, a Unix script *xval.sh* is provided that does the work for you. Unfortunately, this script is not well documented. The command line is:

`xval.sh` *filestem* N [*options*] [+*identifier*]

where N is the order of cross-validation (4 in our case), *options* are options to be passed to C4.5, and +*identifier* is appended to the name of the results summary file, *filestem*.tres+*identifier*.

3. Using the decision tree with *consult* and *consultall*

A decision tree that has been built by C4.5 can be used to classify new cases. There are two programs available to do this do this. The first, *consult*, is an interactive program that requires the user to input the values of individual attributes of the unknown case. This can be helpful in debugging. The command line is:

```
consult -f filestem
```

`consult` expects to find a decision tree in *filestem*`.tree`. It prompts the user for the attribute values of an unknown case and classifies it according to the decision tree. More details are given in the attached `man` page.

However, when there are many unknown cases to be classified, as in part 5 of this assignment, an interactive program is unsuitable, as manually entering many floating-point numbers for each of the cases is a tedious and error-prone task. In these cases, it is better to use the program *consultall* (written by Greg Kondrak for use in this assignment). The command line is:

```
consultall -f filestem
```

`consultall` expects to find a file named *filestem*`.cases` containing cases that are to be classified using the decision tree in *filestem*`.tree`. For each case in *filestem*`.cases`, the program prints the class predicted by the decision tree. The format of the *filestem*`.cases` file is the same as that of the *filestem*`.data` file, except that the name of the class is given as `?`, meaning ‘unknown’. Note that the *filestem*`.names` file must be exactly as it was when the decision tree in *filestem*`.tree` was generated. More information on each case can be obtained by using the *consult* program directly.

Figure 1: Overview of the assignment

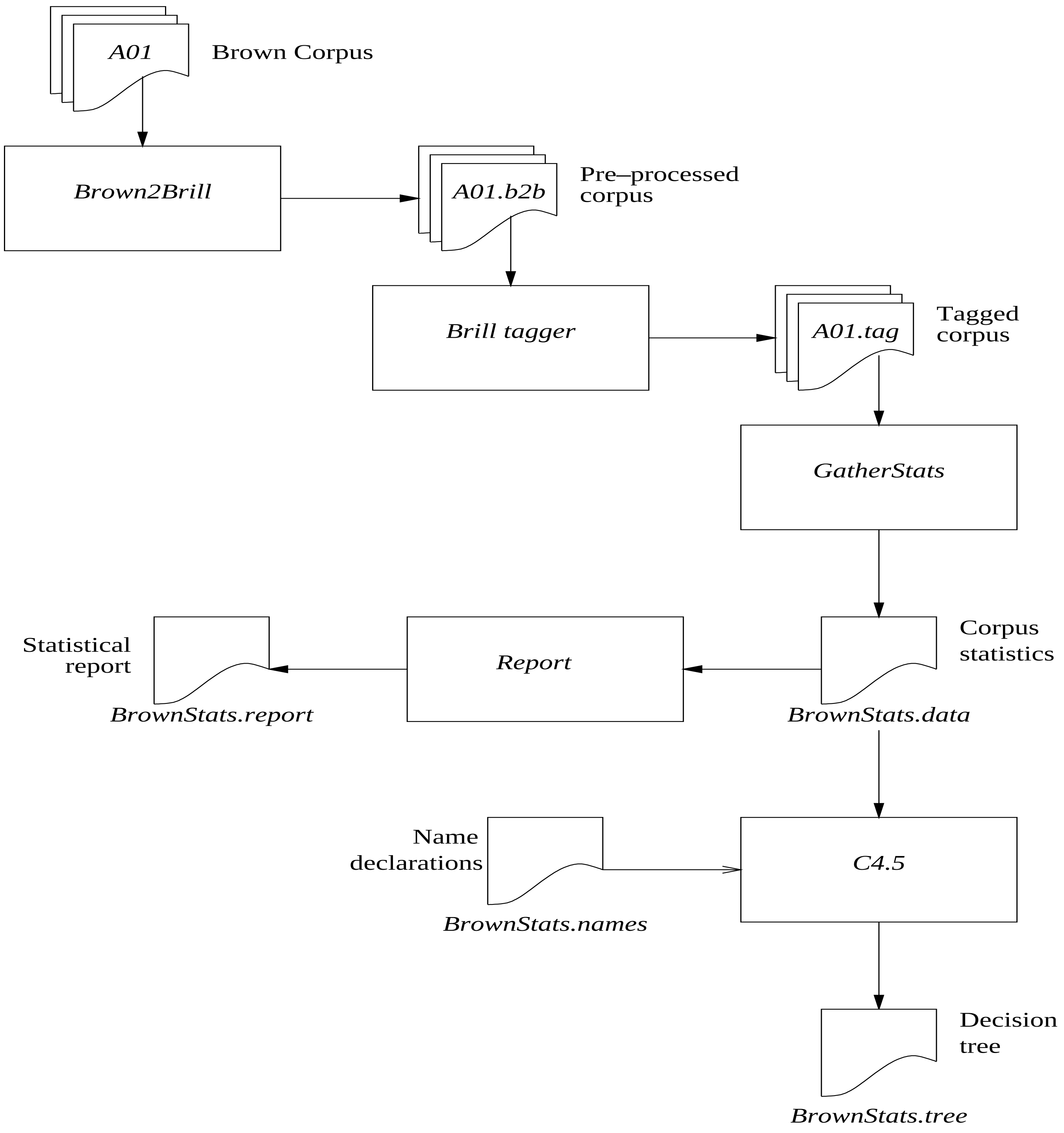

Table 1: Genre and subgenre codes for the Brown Corpus

I. Press
- A. Reportage
- B. Editorial
- C. Reviews

II. Miscellaneous
- D. Religion
- E. Skills and hobbies
- F. Popular lore
- G. Belles-lettres

III. Formal documents
- H. Government and institutional
- J. Learned

IV. Fiction
- K. General
- L. Mystery
- M. Science fiction
- N. Adventure
- P. Romance
- R. Humour

Table 2: Conversion from Brown Corpus to tagged text

Data from Brown Corpus:

```
A01␣0010␣␣1␣␣␣␣␣␣␣The␣Fulton␣County␣Grand␣Jury␣said␣Friday␣an␣investigation
A01␣0020␣␣1␣␣␣␣of␣Atlanta's␣recent␣primary␣election␣produced␣"no␣evidence"
A01␣0020␣␣9␣␣␣␣that␣any␣irregularities␣took␣place.
```

Format for Brill tagger (all on one line of the file!):

```
The␣Fulton␣County␣Grand␣Jury␣said␣Friday␣an␣investigation␣of␣Atlanta␣'s
␣recent␣primary␣election␣produced␣"␣no␣evidence␣"␣that␣any␣irregularities
␣took␣place␣.
```

Output from Brill tagger:

```
The/DT Fulton/NNP County/NNP Grand/NNP Jury/NNP said/VBD Friday/NNP an/DT
investigation/NN of/IN Atlanta/NNP 's/POS recent/JJ primary/JJ election/NN
produced/VBN "/" no/DT evidence/NN "/" that/IN any/DT irregularities/NNS
took/VBD place/NN ./.
```

Table 3a: The Penn part-of-speech tagset—words

Tag	**Name**	**Example**
CC	Coordinating conjunction	*and*
CD	Cardinal number	*three*
DT	Determiner	*the*
EX	Existential *there*	*there [is]*
FW	Foreign word	*d'oeuvre*
IN	Preposition or subordinating conjunction	*in, of, like*
JJ	Adjective	*green, good*
JJR	Adjective, comparative	*greener, better*
JJS	Adjective, superlative	*greenest, best*
LS	List item marker	*(1)*
MD	Modal	*could, will*
NN	Noun, singular or mass	*table*
NNS	Noun, plural	*tables*
NNP	Proper noun, singular	*John*
NNPS	Proper noun, plural	*Vikings*
PDT	Predeterminer	*both [the boys]*
POS	Possessive ending	*'s, '*
PRP	Personal pronoun	*I, he, it*
PRP$	Possessive pronoun	*my, his, its*
RB	Adverb	*however, usually, naturally, here, good*
RBR	Adverb, comparative	*better*
RBS	Adverb, superlative	*best*
RP	Particle	*[give] up*
SYM	Symbol (mathematical or scientific)	+
TO	*to*	*to [go] to [him]*
UH	Interjection	*uh-huh*
VB	Verb, base form	*take*
VBD	Verb, past tense	*took*
VBG	Verb, gerund or present participle	*taking*
VBN	Verb, past participle	*taken*
VBP	Verb, non-3rd-person singular present	*take*
VBZ	Verb, 3rd-person singular present	*takes*
WDT	*wh*-determiner	*which*
WP	*wh*-pronoun	*who, what*
WP$	Possessive *wh*-pronoun	*whose*
WRB	*wh*-adverb	*where, when*

Table 3b: The Penn part-of-speech tagset—punctuation

Tag	**Name**	**Example**
#	Pound sign	*£*
$	Dollar sign	$
.	Sentence-final punctuation	*!, ?*
,	Comma	
:	Colon, semi-colon, ellipsis	
(	Left bracket character	
)	Right bracket character	
"	Straight double quote	
‘	Left open single quote	
“	Left open double quote	
’	Right close single quote	
”	Right close double quote	

Table 4: Statistics to be computed for each text

- Frequency counts (per 1000 tokens):
  — First person
  — Second person
  — Third person
  — Conjunct words
  — Present-tense verbs
  — Past-tense verbs
  — Private verbs
  — Public verbs
  — Verbs of saying
  — Nouns
  — Nouns that start with *q* or *Q*
  — Adverbs
  — *that*
  — Non-existential *there*
  — Split infinitives
  — Hifalutin and technical words
- Average length of sentences (in tokens)
- Average length of tokens, excluding punctuation tokens (in characters)
- Type–token ratio (see definition on page 3)

Table 5: Definitions of word categories

First person:
I, me, my, mine, we, us, our, ours

Second person:
you, your, yours

Third person:
he, him, his, she, her, hers, it, its, they, them, their, theirs

Conjunct words:
alternatively, altogether, consequently, conversely, e.g., else, furthermore, hence, however, i.e., instead, likewise, moreover, namely, nevertheless, nonetheless, notwithstanding, otherwise, rather, similarly, therefore, thus, viz.

Verbs of saying:
address, affirm, allege, apostrophize, articulate, assert, asserverate, aver, bark, bawl, bellow, bespeak, blare, blat, blubber, boom, bray, breathe, buzz, cackle, chant, chatter, chirp, comment, converse, coo, crow, declaim, declare, drawl, enunciate, exclaim, express, flute, gab, gasp, growl, grunt, hiss, interject, keen, lilt, mention, mouth, mumble, murmur, mutter, note, nuncupate, observe, pant, patter, phonate, pipe, proclaim, pronounce, quote, recite, relate, remark, roar, rumble, say, scream, screech, shriek, sibilate, sigh, sing, snap, snarl, snort, sob, sound, speak, squall, squawk, squeak, squeal, state, talk, thunder, trumpet, twang, utter, vocalize, voice, wail, warble, whine, whisper, yak, yap, yawp, yell, yelp

Private verbs:
anticipate, assume, believe, conclude, decide, demonstrate, determine, discover, doubt, estimate, fear, feel, find, forget, guess, hear, hope, imagine, imply, indicate, infer, know, learn, mean, notice, prove, realize, recognize, remember, reveal, see, show, suppose, think, understand

Public verbs:
acknowledge, admit, agree, assert, claim, complain, declare, deny, explain, hint, insist, mention, proclaim, promise, protest, remark, reply, report, say, suggest, swear, whisper, write

Split infinitives:
TO + {RB | RBR | RBS} + VB

Hifalutin and technical terms:
Any noun or verb with at least 10 characters, except for
VBG for which the minimum is 12 characters.

Table 6: Example `labor-neg.names` and `labor-neg.data` files for C4.5.

Names:

```
|  Title: Final settlements in labor negotiations in Canadian industry

|  Classes
|  -------

good, bad.

|  Attributes
|  ----------

duration:                         continuous
wage increase first year:         continuous
wage increase second year:        continuous
wage increase third year:         continuous
cost of living adjustment:        none, tcf, tc
working hours:                    continuous
pension:                          none, ret_allw, empl_contr
standby pay:                      continuous
shift differential:               continuous
education allowance:              yes, no
statutory holidays:               continuous
vacation:                         below average, average, generous
longterm disability assistance: yes, no
contribution to dental plan:      none, half, full
bereavement assistance:           yes, no
contribution to health plan:      none, half, full
```

Data:

```
1,5.0,?,?,?,40,?,?,2,?,11,average,?,?,yes,?,good
2,4.5,5.8,?,?,35,ret_allw,?,?,yes,11,below average,?,full,?,full,good
?,?,?,?,?,38,empl_contr,?,5,?,11,generous,yes,half,yes,half,good
3,4.5,4.5,5.0,?,40,?,?,?,?,12,average,?,half,yes,half,good
3,4.0,5.0,5.0,tc,?,empl_contr,?,?,?,12,generous,yes,none,yes,half,good
3,6.9,4.8,2.3,?,40,?,?,3,?,12,below average,?,?,?,?,good
2,3.0,7.0,?,?,38,?,12,25,yes,11,below average,yes,half,yes,?,good
1,5.7,?,?,none,40,empl_contr,?,4,?,11,generous,yes,full,?,?,good
3,3.5,4.0,4.6,none,36,?,?,3,?,13,generous,?,?,yes,full,good
2,6.4,6.4,?,?,38,?,?,4,?,15,?,?,full,?,?,good
2,3.5,4.0,?,none,40,?,?,2,no,10,below average,no,half,?,half,bad
3,3.5,4.0,5.1,tcf,37,?,?,4,?,13,generous,?,full,yes,full,good
1,3.0,?,?,none,36,?,?,10,no,11,generous,?,?,?,?,good
2,4.5,4.0,?,none,37,empl_contr,?,?,?,11,average,?,full,yes,?,good
1,2.8,?,?,?,35,?,?,2,?,12,below average,?,?,?,?,good
```