Nota Bene: This is just a design, not working code. But something like this (code cleanup) is something we probably need to do anyway to make it possible to move forward in future work without tons of almost-but-not-quite-identical cargo-culted C code in the classifiers.

Anyway- any comments? Broken-nesses?

Bill Yerazunis

Design for a callable CRM114 library

This is the proposed design for a callable library to do these things:

PRIMARY REASONS

- Recode to be 64-bit safe
- Recode to factor commonly used functionality
- Recode to make all classifiers double-sided and polyvalent

Why aren't these things there *now*? Because when I started the first experiments on what would become CRM114 back in 1998, I had no clue that 64-bit was ever going to happen on commodity PCs, nor that there was ever going to be more than one classifier, nor that anybody would ever really need or want a classifier that needed the complexity of training that double-sided classifiers need, let alone have polyvalent classifiers.

But now it's a new millennium, the initial code is a decade old, and we now know a lot more about the problem. So maybe it's time for a rewrite of the classifiers (maybe the language could also use a rewrite, but let's not get into that).

DETAILS AND OPEN QUESTIONS

1. Replace the current set of classifiers in CRM114 with equivalent classifiers but recoded as needed to be 64-bit safe.

2. Refactor classifiers to all use common methods for choosing statistics files, tokenizing input (if it's a tokenizing classifier like OSB, not like bit entropy), and comparing / displaying match results.

    2.1. Each statistics file will be self-identifying as to type, so master "learn" and "classify" routines can do the right thing by checking the file type.

        2.1.1. Each classifier will return results in a common format (a pR, a probability, and a line of human-readable text.) The master classify dispatcher will assemble these into the final output.

        2.1.2. Each classifier will have an additional call to "get information" and "groom" (doing like what cssutil does now), however these will be part of the library, with a common calling sequence.

3. Allow plain old C programs to call CRM114 classifiers. You won't have to learn Latin action verbs (although the current language with it's declensional syntax and overlapped-string memory model will be continued; the new classifiers will be forward-compatible)

    3.1. This will be a C library (not C++, although it will be callable from C++). Reason: C++ name munging makes C++ libraries callable only from C++, whereas (at last in much of the computing world) an ANSI C API means you can be called by any programming language.

4. Architecture Change:

    4.1. all classifiers will be doublesided; that is, you will *always* train both in-class and out-of-class samples.

        Yes, it's true that some of the algorithms don't need doublesided input. That's OK;classifiers are free to disregard that input BUT note that for a polyvalent operation each classifer file needs to be able to stand alone with in-class and out-of-class examples and so some classifiers will need to change their behavior.

4.1.1. This means that the REFUTE keyword (which currently means *both* "remove from class" and "learn as a negative example" will need to be recast. Probably REFUTE will mean "not in class" and another keyword will mean "erase this".

4.1.1.1. How about ERASE or FORGET as the erase keyword? There is value in Keeping It Super Simple.

Or a cool Latin keyword? Note to self: what's latin for "erase" or "forget"?
("delende/deletus" = destroy, obliterate; "oblitus" = forget. Not cool enough...)

4.2. All statistics files will be self-identifying as to method and version.

4.2.1. Short Cut: use the multisection format in the new neural network code as the "new standard".

4.2.2. You specify the classifier type for the first LEARN, and after that, type information in the LEARN and CLASSIFY statements is *disregarded*; the file knows what it is and won't be fooled. (flags that are compatible like MICROGROOM (or not) and UNIQUE will still be honored; whether things like changing the tokenize regex or VT pipeline vectors may or may not be carried over and may or may not be overrideable;

4.2.2.1. What's the wisdom on this? Allow changes or not to things like the parse regex and VT pipeline parameters? Going from OSB-features with unigrams to UNIGRAMs would work. On the other hand, if you specify a custom pipeline during LEARN, then you can definitely use smaller pipelines depending on what you want to do with it.

Tentative answer: allow it but make the default of "no flags" (a null parameter pointer) use the values originally used in the creating LEARN. This is a case of "You make a mess, you clean it up."

5. Open Question: what's the penalty for making statistics files 32/64-bit portable? How much of a performance hit is that? If it's not horrible, do it. Or go to 32-bits-everywhere statistics, with a compile-time option to go to 64-bits-everywhere via a proper magic typecast.

6. General Guidance on types to use:

6.1. Use a typedef on hashes so that we can modulate this as needed. The type "crmhash_t" is specifically meant for this. This will default to 32 bits everywhere.

6.2. Make sure crmhash_t is at least size_t for all hashes and indices, especially indices derived from hashes. ...ESPECIALLY indices derived from hashes.

6.3. Use "int" everywhere else (int is 32 bits on gcc on X86-32 - I wrote a test example just to prove it). Don't use "unsigned" without explaining why; don't use "long" or "long long" at all! (use (u)int64_t instead.

6.4. Yes, this is creepy for all of us embedded systems programmers where an int may well be just 8 bits.

Get over it.

6.5. Unless you can guarantee there will never be an embedded NULL character, all strings must be counted-length.

7. Open Question: what will the accuracy impact of going to 32-bit tokens for everything? Long ago, it was like 2% in full Markovian, but times change and algorithms improve.

8. Make ALL classifiers polyvalent - that is, you will now be able to train your business mail with a Markovian and your personal mail with an SVM and your spam with a neural network and you can use all of them in the same CLASSIFY statement.

8.1. This works because each file is examined in the top level CLASSIFYer and and the proper actional routine (_learn, _classify, _getinfo, and _groom) based on the actual type inside the file.

8.2. This makes "none of the above" classification will be much easier to do; some classifiers currently renormalize between different statistics files which makes NONE_OF_THE_ABOVE results difficult to determine. (i.e. right now, if you run a CLASSIFY with OSB but with only one statistics file, it always comes back with a perfect match (100% probability, pR > 300ish). That's a garbage answer, and with doublesided training, we can make it not happen any more.

9. While we're doing this, we might want to consider thread safety; that is, if you have multiple threads running, they should not step on each other. (note that this means that the fixed-size buffers inbuf, outbuf, and tmpbuf

cannot be used as they currently would be shared among all of the threads.)

9.1. Modify *all* classifiers to NOT use inbuf, outbuf, or tmpbuf. Such fixed buffers are problematic in the context of a callable library, and are not thread safe.

9.2. Use malloc/free instead.

9.3. If we really want thread safety (and we would need it for some applications, like running in a browser that does threads) then we can't have anything not in a per-thread structure. Therefore, provide a base structure to carry things like the regex engine pointers, the flags, temp data, all that.

10. Regexes: we'll use routines very much like the current crmregex calls but we need to make them redirectable at runtime. If we want thread safety, then we also need to pass the routine pointers along in the control structure.

11. File name parsing: We need to provide a call to break the filenames string into an array of filenames (NULL termination is OK because POSIX filenames are always null terminated and there is no "counted string" filename access call.

11.1. Prototypical prototypes:

```
crm_filename_split ( char *in_filename, int *start_offsets, int *lens);
```

11.2. Note that this does not address the problem of people who put spaces in their filenames. But, by putting the whole parsing issue in one centralized place, if we ever come up with a way to handle such filenames cleanly, we can then put the fix in one place as well.

12. In/Out parsing: We will provide a call to find which filename is "|" (needed for the in-class / out-of-class parsing)

13. Statistics file headers: Provide a call that accepts the info needed for a statistics file, and creates the file, and another call that accepts a filename and maps that statistics file (with it's subfields properly parsed out and the pointers returned). Recommendation: use the one currently in the new neural network classifier code.

PROPOSED API

This is the proposed API for calling the libcrm114 classifiers.

INITIALIZATION AND CLEANING UP

To initialize the library (currently there's nothing known that needs initialization, but we'll leave open the possibility) and create a control structure. The control structure contains all of the flags and things needed to control the classifiers:

```
crm114_controlstruct mystruct = libcrm114_controlstruct (void *foo)
```

`void *foo` is a struct that may in the future contain something. Maybe. For now, you can pass a NULL pointer.

This returned control struct is where all of the longer-term storage resides during execution. Thus, please do an orderly cleanup by calling

```
int libcrm114_freecontrolstruct(crm114_controlstruct *my_controlstruct);
```

which frees all of the memory and does the other appropriate releases.

Note that this is thread-safe; you can have many control structs at the same time.

LEARNING

To actually LEARN a text, use this:

```
int libcrm114_learntext(char *text,
    uint32_t textlen,
    char *filename,
    long classnumber, // 0 for "in", 1 for "out"
    crm114_controlstruct *my_controlstruct);
```

Future Expansion Issue: classnumber may someday expand to beyond 0 and 1. However, that's a "reserved" thing and for now, classnumber > 0 will be treated as classnumber = 1.

CLASSIFYING

To actually CLASSIFY a text, use this:

```
int libcrm114_learntext(char *text,
    uint32_t textlen,
    char *filename,
    long classnumber, // 0 for "in", 1 for "out"
    crm114_controlstruct *my_controlstruct);
```

Future Expansion Issue: classnumber may someday expand to beyond 0 and 1. However, that's a "reserved: thing and for now, classnumber > 0 will be treated as classnumber = 1.

GETTING INFO

To get info about a particular classifier file, use:

```
int libcrm114_getinfo(char *filename,
    char *out_text,
    int outtextlen);
```

which creates a human-readable (but intentionally regex-parseable) description about a particular statistics file.

RESIZING

To resize a file (almost always unnecessary; microgrooming is preferred)

However, if you need to compress a file (say, for embedding onto an teeney little embedded system) or grow a file you can do this:

```
int libcrm114_resize(char *filename,
    float size_ratio);
```

where size_ratio is how big the new file should be compared to the old file. Not every file type can be resized; a negative return value will indicate that the

operation is not supported. Note that some classifiers will auto-grow, most others have microgrooming, and most classifiers work just fine out of the box, so most users will never need to use this.

NONSTANDARD INITIALIZATIONS

Normally, the routine libcrm114_controlstruct() will create a perfectly acceptable default match control structure.

Setting the desired regex engine:

The control struct contains slots for the regex engine. The default (TRE) is set up by `libcrm114_controlstruct`.

In case you want to use other than the standard regex engines, you can. This function allows you to set the five regex functions available (the actual calling sequences will be the same as the current functions in CRM114; this just allows users to pick any regex package they want without needing to be locked to the TRE regex library. We should possibly supply two wrappers that set this for TRE and FSF regex libraries at the minimum, and default to TRE).

```
int libcrm114_setregexpkg(crm114_controlstruct int regcomp(), int
regexec(), int regfree(), size_t regerr(), char *regversion() )
```

Setting File Cacheing

You can turn on or off the file cacheing; use

```
int libcrm114_setfilecacheing(int filecacheing_enable);
```

with a 0 to turn it off, or a 1 to turn it on.