MeTA: ModErn Text Analysis (/) A Modern C++ Data Sciences Toolkit

Classification

Beyond its search and information retrieval capabilities, MeTA also provides functionality for performing document classification on your corpora. We will start this tutorial by discussing how you can perform document classification experiments on your data **without writing a single line of code**, and then discuss more complicated examples later on.

The first step in setting up your classification task is, of course, selecting your corpus. MeTA's built-in `classify` program supports the following corpus inputs:

- anything from which an `inverted_index` can be generated
- pre-processed, libsvm-formatted corpora

MeTA's internal representation for its `forward_index` uses libsvm-formatted data, so if you already have your data in this format you are encouraged to use it. Otherwise, you can generate libsvm-formatted data automatically from any existing `inverted_index` .

Creating a Forward Index From Scratch

To create a `forward_index` directly from your corpus input, your configuration file would look something like this:

```
corpus-type = "line-corpus"
dataset = "20newsgroups"
forward-index = "20news-fwd"
inverted-index = "20news-inv"

[[analyzers]]
method = "ngram-word"
ngram = 1
filter = "default-chain"
```

The process looks something like this: first, an `inverted_index` will be created (or loaded if it already exists) over your corpus' data, and then it will be un-inverted to create the `forward_index` . The process of un-inverting is a one-time cost, and is necessary to provide efficient access to the document vectors for the classifiers. Once you have generated your `forward_index` , you *never need to generate it again* unless you want to change your document representation. Once you have a `forward_index` , you can use it with *any* of the classifiers provided by MeTA.

Creating a Forward Index from LIBSVM Data

In many cases, you may already have pre-processed corpora to perform classification tasks on, and MeTA gracefully handles this. The most common input for these classification tasks is the LIBSVM file format, and MeTA supports this format directly as an input corpus.

To create a `forward_index` from data that is already in LIBSVM format, your configuration file would look something like this:

```
corpus-type = "line-corpus"
dataset = "rcv1"
forward-index = "rcv1-fwd"
inverted-index = "rcv1-inv"

[[analyzers]]
method = "libsvm"
```

The `forward_index` will recognize that this is a LIBSVM formatted corpus and will simply generate a few metadata structures to ensure efficient random access to document vectors. An `inverted_index` is not created through this method, and so you will not be able to use classifiers such as `knn` that require an `inverted_index` .

Selecting a Classifier

To actually run the `classify` executable, you will need to decide on a classifier to use. You may see a list of these in the API documentation for the `classifier` class (doxygen/classmeta_1_1classify_1_1classifier.html) (they are listed as subclasses). The public static id member of each class is the identifier you would use in the configuration file.

A recommended default configuration is given below, which learns an SVM via stochastic gradient descent:

```
[classifier]
method = "one-vs-all"
    [classifier.base]
    method = "sgd"
    loss = "hinge"
    prefix = "sgd-model"
```

Here is an example configuration that uses Naive Bayes:

```
[classifier]
method = "naive-bayes"
```

Here is an example that uses *k*-nearest neighbor with *k* = 10 and Okapi BM25 as the ranking function:

```
[classifier]
method = "knn"
k = 10
    [classifier.ranker]
    method = "bm25"
```

Running `./classify config.toml` from your build directory will now create a `forward_index` (if necessary) and run 5-fold cross validation on your data using the prescribed classifier. Here is some sample output:

```
          chinese   english   japanese
        -----------------------------
 chinese | 0.802     0.011     0.187
 english | 0.0069    0.807     0.186
japanese | 0.0052    0.0039    0.991

------------------------------------------------
Class       F1 Score    Precision   Recall
------------------------------------------------
chinese     0.864       0.802       0.936
english     0.88        0.807       0.967
japanese    0.968       0.991       0.945
------------------------------------------------
Total       0.904       0.867       0.949
------------------------------------------------
1005 predictions attempted, overall accuracy: 0.947
```

Online Learning

If your dataset cannot be loaded into memory, you should look at the online learning tutorial (online-learning.html) for more information about how to handle this case.

Manual Classification

If you want to customize the classification process (such as providing your own test/training split, or changing the number of cross-validation folds), you should interact with the classifiers directly by writing some code. Refer to `classify.cpp` and the API documentation for `classifier` (doxygen/classmeta_1_1classify_1_1classifier.html).

Here’s a simple example that changes the number of folds to 10:

```
auto f_idx = meta::index::make_index<index::memory_forward_index>(argv[1]);
auto config = cpptoml::parse_file(argv[1]);

auto class_config = config.get_table("classifier");
auto classifier = meta::classify::make_classifier(*class_config, f_idx);

auto confusion_mtrx = classifier->cross_validate(f_idx->docs(), 10);
confusion_mtrx.print();
confusion_mtrx.print_stats();
```

And here’s a simple example that uses your own training/test split:

```
auto f_idx = meta::index::make_index<index::memory_forward_index>(argv[1]);
auto config = cpptoml::parse_file(argv[1]);

auto class_config = config.get_table("classifier");
auto classifier = meta::classify::make_classifier(*class_config, f_idx);

auto train = /* filter f_idx->docs() for your training set */;
auto test = /* filter f_idx->docs() for your test set */;

classifier->train(train);
auto confusion_mtrx = classifier->test(test);
confusion_mtrx.print();
confusion_mtrx.print_stats();
```

Writing Your Own Classifiers

Any classifiers you write should subclass `classify::classifier` and implement its virtual methods. You should be clear as to whether your classifier directly supports multi-class classification (subclass from `classify::classifier` directly) or only binary classification (sublcass from `classify::binary_classifier`).

If you would like to be able to create your classifier by specifying it in a configuration file, you will need to provide a public static id member that specifies the text that identifies your classifier class, and register it with the toolkit somewhere in `main()` like this:

```
// if you have a multi-class classifier
meta::classify::register_classifier<my_classifier>();

// if you have a multi-class classifier that requires an
// inverted_index
meta::classify::register_multi_index_classifier<my_classifier>();

// if you have a binary classifier
meta::classify::register_binary_classifier<my_binary_classifier>();
```

If you need to read parameters from the configuration group given for your classifier, you should specialize the `make_classifier()` function like so:

```
// if you have a multi-class classifier
namespace meta
{
namespace classify
{
template <>
std::unique_ptr<classifier>
    make_classifier<my_classifier>(
        const cpptoml::table& config,
        std::shared_ptr<index::forward_index> idx);
}
}

// if you have a multi-class classifier that requires an
// inverted_index
namespace meta
{
namespace classify
{
template <>
std::unique_ptr<classifier>
    make_multi_indexclassifier<my_classifier>(
        const cpptoml::table& config,
        std::shared_ptr<index::forward_index> idx,
        std::shared_ptr<index::inverted_index> inv_idx);
}
}

// if you have a binary classifier
namespace meta
{
namespace classify
{
template <>
std::unique_ptr<classifier>
    make_binary_classifier<my_binary_classifier>(
        const cpptoml::table& config,
        std::shared_ptr<index::forward_index> idx,
        class_label positive_label,
        class_label negative_label);
}
}
```

Documentation for MeTA: ModErn Text Analysis (https://github.com/meta-toolkit/meta)