BaysianMimeTypeSelector

Describe BaysianMimeTypeSelector here.

TIKA-1517 [MIME type selection with probability]

The motivation is that the current implemenation within MimeTypes for detecting mime types in Tika is a bit stiff and less flexible(at the time the article is being written, the current version of MimeTypes which has 3 detection approaches to identify mime types is implemented with a fall-back strategy), the detection highly depends on the magic byte detection.

The last two approaches (i.e. extension and metatdatahint matching) are subsidiary and auxiliary in the final detection decsion. In other words, the decision that comes from the last two approach will probablly be considered when there is a tie to break in the magic bytes detection as there might be multiple mime types estimated by magic bytes method, in this situation file extension and metadatahint will be used.

It is also possible that in some situations the type given by the file extension and metadata hint matching are more specialized than magic bytes method, then the most specialized or specific type gets returned. This implementation seems to exhibt a bit inflexibilities in some situations where users prefer a particular type of detection e.g. they might only trust or prefer their file extensions.

Perhaps, in the future we might have more probablistic mime detection algorithms being considered for deploying into Tika, probably from this perspective, the current implementation also seems to give less space for expanding with more detection methods in Tika.

Therefore, it would be great to have a feature like Tika-1517 where user can add weights or preference on the detection method they want to use for detecitng mime types.

More information with this intuition of the feature - probabilistic mime type selection can be found in Tika 1517. The algorithm behind this feature is actually a simple or naive baysian rule, we eventually compute 3 scores each of which corresponds to a detection method, the one that gives the maximum score or posterior probability will be returned as final detected type.

The idea can be also simply illustrated with the samlam bayesian network toolkit which is available for download on http://reasoning.cs.ucla.edu/samiam/index.php

The following screenshot with some examples illustrates the intuition and idea with the toolkit, please note under the hood, the same baysian network is implemented in this feature in Tika.

The important thing to be noted is that each detection method needs to be assigned with a conditional probability and prior, which could be all regarded as a value of trust.

Initially

The prior for the type detected by each the detection method is 0.5, this forces our prior estimation to be stochastic; The virtue of the baysian rule is to incorperate the notion of condition probability, which in this case needs to be specified by users in terms of their own preference or the domain knowledege where they know for certain which method they want to trust most and how much they want to trust; In order to estimate or find the exact and intuitive value, this mentioned toolkit can be used to explore their conditional probability settings.

The following shows the initial conditional probability values used in the default setting for the feature in Tika.

```
/* conditional probability: probability of the type estimated by the Magic test given that the type is the type predicted by the magic test.  */ private static final float DEFAULT_MAGIC_TRUST = 0.9f;
```

```
/* conditional probability: probability of the type estimated by the metadata hint test given that the type is the type predicted by the metadata hint test.*/ private static final float DEFAULT_META_TRUST = 0.8f;
```

```
/* conditional probability: probability of the type estimated by the extension test given that the type is the type predicted by the extension test.*/ private static final float DEFAULT_EXTENSION_TRUST = 0.8f;
```

```
/* conditional probability: probability of the type Not estimated by the Magic test given that the type is not the type predicted by the magic test.*/ 0.9f
```

```
/* conditional probability: probability of the type Not estimated by the meadata test given that the type is not the type predicted by the metadata test.*/ 0.8f
```

```
/* conditional probability: probability of the type Not estimated by the extension test given that the type is not the type predicted by the extension test.*/ 0.8f
```

https://lh4.googleusercontent.com/wUrj6keimRl2xjZjxutax0jXyhIbHBkOtNZTwLoHmU2q_VFq_fVxLBP1YRv29teUZdbdvXunj7xiTtC3GLtCErsg_yz2NaftfVO9YF9rmdD18QRgaHoYJ21k-kEb2WrOw6NST8U|height="381px;",width="563px;"

The above tells that the file extension and contentMetadata matching method fail to detect a type (i.e. they both return byte-stream as the type), and only the magic test returns a non-byte-stream type, then the type estimated by magic test has a higher posterior probability 90%;

https://lh5.googleusercontent.com/FO5VfTBjcTuogGvGz0XecZ_3oSABCNdswjFum-ILO1_37VPK6WiSD8vXMggPNPvGRagyXvrymeQU9iM9Ta69vVJyLT7guSE-6srV5efQMHKHSDdO0H_ZU4tjHaga2wlY_c_QBEE|height="364px;",width="562px;"

The above screenshot shows magic Test and file extension method do not agree on the same type, the posterior probability of the type estimated by magic test is 69.23%

https://lh4.googleusercontent.com/3Q68oP4-IFoAP5woGiqiaSLN_JwN11ZBylS2l9NlmPQg16ZxGezBK6vxBxl4sEz5xOgy_A-2pfAMgz2bmy4gEPu829tYAnnWG03rQaV-tLKs_j_B-WZ5kwGwF_oOSJeGDwF6SwU|height="368px;",width="595px;"

This one shows the posterior probability of the type estimated by the glob test(file extension matching), and the magic test(magic byte matching) does not agree with it on the same type, so the type predicted by the glob test is lower and it is 30.77%; (note the content test(content metadata machine) above fails to detect a type, so it is not participating into compuation of the final posterior probability for the type estimated by glob test);

Kindly note, usually the metadata hint is present in the http response header, if we bluntly or directly give a file to the feature in Tika, the meta data hint usually is absent.

How to use this feature in Tika (proposed use)

If a user wanted to use this feature, the following code would be needed.

```
tika = new Tika(new TikaConfig() {
                @Override
                protected Detector getDefaultDetector(MimeTypes types,
                        ServiceLoader loader) {
                    /*
                     * here is an example with the use of the builder to
                     * instantiate the object.
                     */
                    Builder builder = new ProbabilisticMimeDetectionSelector.Builder();
                    ProbabilisticMimeDetectionSelector proDetector = new ProbabilisticMimeDetectionSelector(
                            types, builder.priorMagicFileType(0.5f)
                                    .priorExtensionFileType(0.5f)
                                    .priorMetaFileType(0.5f));
                    return new DefaultProbDetector(proDetector, loader);
                }
            });
```

- The idea is simple that we overwrite the getDefaultDetector() by providing the DefaultProbDetector which extends the CompositeDetector, a CompositeDetector is one (whose supertype is Detector) that takes a list of detectors, and when its detect() method gets called, each detector in the list is called sequentially one after another. The original implemenation of getDefaultDetector() in TikaConfig returns an instance of “DefaultDetector” that also extends the CompositeDetector by providing a list of detectors that includes “MimeTypes” which is the native implemenation with 3 detectors (i.e. magic bytes, extension and metadatahint). However, DefaultProbDetector replaces this MimeTypes with ProbabilisticMimeDetectionSelector.

In order to set the preferential weights, an instance of ProbabilisticMimeDetectionSelector can be created in the above example snippet.

Alternatively, if we dont want to go with the default settings with ProbabilisticMimeDetectionSelector, it is ok to just ignore the arguments by calling only “return new DefaultProbDetector();”.

- Alternatively, if we dont want to write some extra code, the following can also be used.

```
/*
             * an xml file needs to be given to tell where the detector is
             * located, customers can build their own detectors by including or
             * excluding this feature or any detectors in the composite list at
             * their will.
             */
            tika = new Tika(new TikaConfig(
                    new File("TIKA-detector-sample.xml")));
```

TIKA-detector-sample.xml

<?xml version="1.0" encoding="UTF-8"?> <!--

--> <properties>

- <detectors>
    - <detector class="org.apache.tika.detect.DefaultProbDetector"/>
      </detectors>

</properties>