TUDIIR Toolkit Overview

David Urbansky, Klemens Muthmann

TU Dresden, Department of Systems Engineering, Chair Computer Networks, IIR Group, Germany

September 29, 2010

Contents

Chapter 1

Introduction

Internet Information Retrieval (IIR) is a research domain in computer science that is concerned with the retrieval, extraction, classification, and presentation of information from the Internet. This toolkit provides functionality which is often needed to perform IIR tasks such as crawling, classification, and extraction of various information types.

1.1   What is TOOLKIT?

1.2   What is TOOLKIT NOT?

1.3   License

The complete source code is licensed under the Apache License 2.0. All source files should include the following license snippet at the very top.

```
Copyright 2010 David Urbansky, Klemens Muthmann
Licensed under the Apache License, Version 2.0 (the "License"); you may not
use this file except in compliance with the License. You may obtain a copy of
the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

1.4   Alternative and Complimentary Toolkits

Some functionalities of this toolkit are covered in other libraries. Before you start using TOOLKIT you might want to take a look at these alternatives. The objective of TOOLKIT is to create new functionalities or improve existing ones, we do not intend to reinvent the wheel. For example, TOOLKIT has no functionality for part-of-speech tagging (PoS tagging) since there are many toolkits that do this task pretty accurate already. In case you do not find the functionality you are looking for in this toolkit, you probably will in one of the following toolkits.

1. **AlchemyAPI** [1] is a commercial web-service that can be used via several programming languages. The service offers named entity recognition, text classification, language identification,

concept tagging, keyword extraction, content scraping, and web page cleaning. The service comes in 4 variants: free, basic, professional, and metered.

2. **Apache Mahout** [38] is a Java-based machine learning library. Its main features are collaborative filtering, user and item based recommenders, (fuzzy k-means clustering, mean shift clustering, latent dirichlet process allocation, singular value decomposition, parallel frequent pattern mining, complementary naive bayes classifier, and a random forest decision tree based classifier. The library is licensed under the Apache Software license.

3. **Balie** [5] is a Java-based information extraction library. Its main features are language identification, tokenization, sentence boundary detection, and named entity recognition (using dictionaries). The library is licensed under the GNU GPL and supports English, German, French, Romanian, and Spanish as input languages.

4. **ContentAnalyst** [9] is a commercial platform for text analytics. The platform's main features are concept search, dynamic clustering, near-duplicate document identification, automatic summarization, text classification, and latent semantic indexing.

5. The **Dragon Toolkit** [49] is a Java-based development package for information retrieval and text mining. Its main features are text classification, text clustering, text summarization, and topic modeling.

6. **FreeLing** [4] is a natural language processing library written in C++. Its main features are Text tokenization, sentence splitting, morphological analysis, sSuffix treatment, retokenization of clitic pronouns, flexible multiword recognition, contraction splitting, probabilistic prediction of unkown word categories, named entity detection, recognition of dates, numbers, ratios, currency, and physical magnitudes, PoS tagging, chart-based shallow parsing, named entity classification, WordNet based sense annotation and disambiguation, Rule-based dependency parsing, and nominal correference resolution. It is licensed under GPL and supports Spanish, Catalan, Galician, Italian, English, Welsh, Portuguese, and Asturian as languages. An online demo is available under `http://garraf.epsevg.upc.es/freeling/demo.php`.

7. **GATE** [10] is a Java-based text mining and processing framework. The framework itself comes with few text processing features but many plugins can be used and chained into a text engineering pipeline. The framework is licensed under the GNU Lesser General Public License.

8. The **Illinois Cognitive Computation Group** [16] has a list of ready to use programs for semantic role labeling, text chunking, named entity tagging, named entity discovery, PoS tagging, unsupervised rank aggregation, and named entity similarity metrics.

9. **Julie NLP** [44] is a Java-based toolkit of UIMA based text processing components. The toolkit can be used for semantic search, information extraction, and text mining. The Toolkit is licensed under the Common Public License.

10. **Language Computer** [21] provides commercial products for sentence splitting, tokenization, PoS tagging, named entity recognition, co-reference resolution, attribute extraction, relationship extraction, event extraction, question answering, and text summarization.

11. **Lingo3G** [22] is a text clustering engine that organizes text collections into hierarchical clusters. The software is commercial but [41] offers an open source alternative for text clustering algorithms written in Java. The algorithms integrate with other programming or scripting languages such as PHP, Ruby, and C# too.

12. **LingPipe** [2] is a text processing toolkit using computational linguistics. LingPipe is written in Java. Its main features are topic classification, named entity recognition, clustering, PoS tagging, sentence detection, spelling correction, database text mining, string comparisons, interesting phrase detection, character language modeling, chinese word segmentation, hyphenation and syllabification, sentiment analysis, language identification, singular value decomposition, logistic regression, expectation maximization, and word sense disambiguation. LingPipe is available under a free license for academic use and several commercial licenses.

13. **Mallet** [28] is a Java-based toolkit for statistical natural language processing. Its main features are text classification, sequence tagging (PoS tagging), topic modeling, and numerical optimization. The toolkit is licensed under the Common Public License.

14. **MinorThird** [8] is a Java-based toolkit for text processing. Its main features are annotating text, named entity recognition, and text classification. The toolkit is licensed under the BSD license.

15. **MontyLingua** [23] is a Python and Java-based toolkit for natural language processing (English only). Its main features are tokenization, PoS tagging, lemmatization, and natural language summarization. The toolkit is free for non-commercial use and licensed under the MontyLingua version 2.0 License.

16. **MorphAdorner** [30] is a Java-based command line program for text processing. Its main features are language recognition, lemmatization, name recognition, PoS tagging, noun pluralization, sentence splitting, spelling standardization, text segmentation, verb conjugation, and word tokenization. The program is licensed under a NCSA style license.

17. **NaCTeM Software Tools** [31] are programs for natural language processing and text mining that are made available by the National Centre for Text Mining. The programs include functionality for PoS tagging, syntactic parsing, named entity recognition, sentence splitting, text classification, and sentiment analysis.

18. **NLTK** [25] is a Python-based natural language processing toolkit. Its main features are tokenization, stemming, PoS tagging, text classification, and syntactic parsing. The toolkit is licensed under the Apache 2.0 license.

19. **OpenCalais** [34] is a web service that performs named entity recognition, fact and event extraction. The web service is free for commercial and non-commercial use but limited to 50,000 transactions a day. A professional plan is available too including more transactions and an service license agreement.

20. **OpenNLP** [35] is a toolkit of various open source natural language processing packages. The goal of this toolkit is to integrate several stand-alone projects to increase the interoperability. Algorithms in this toolkit include but are not limited to tokenization and named entity recognition.

21. The **RASP System** [6] is a C and Lisp-based toolkit for natural language processing (English only). Its main features are tokenization, PoS tagging, lemmatization, morphological analysis, and grammar-based parsing. The toolkit is free for non-commercial use and licensed under the RASP System License.

22. The **Rosette Linguistic Platform** [37] is a software suite that can perform name translation, name matching, named entity recognition, morphological analysis, and language identification. The suite works for 55 European, Asian, and Arabic languages. The software is a commercial product.

23. **Stanford NLP** [40] is a set of Java-based natural language processing libraries. Their main features are PoS tagging, named entity recognition, Chinese word segmentation, and classification. The software distributions are licensed under the GNU Public License.

24. **SRILM - The SRI Language Modeling Toolkit** [42] is a C++-based toolkit for language modeling. Its main features are speech recognition, statistical tagging and segmentation, and machine translation. The toolkit is free for non-commercial use and licensed under the SRILM Research Community License.

25. **TextAnalyst** [43] is a commercial text processing software offering text summarization, semantic information retrieval, meaning extraction, and text clustering.

26. **VisualText** [46] is a natural language processing software that addresses named entity recognition, text indexing, text filtering, text classification, text grading, and text summarization.

27. **WEKA** [15] is a Java-based machine learning and data mining library. The library contains a large set of machine learning algorithms such as Support Vector Machines, Neural Networks, Naive Bayes, k-nearest neighbor for but not limited to (text) clustering, (text) classification, and regression. The library is licensed under the GNU General Public License.

Chapter 2

Installation and making it work

The TUDIIR toolkit is managed using Subversion (SVN) (see 3). It is build and tested automatically on a weekly basis using Apache Maven and Hudson CI. Bugs might be reported using the Mantis bugtracker. The project encoding must be set to UTF-8. To support unavailable libraries we manage our own Maven repository using Sonatype Nexus. How to use these components is explained in the next sections.

At the moment each of the systems manages its own user base so you need to register with each one individually. When you start working with the toolkit you might consider sending an E-Mail to the administrator to get access to Mantis, Nexus and Hudson. Provide your name, the name of your advisor and the reason why you need to work with the toolkit and you will get your logins.

2.1 Building the toolkit using Apache Maven

Apache **Maven needs to be installed** before building the toolkit. How to do this is explained here: Maven installation. There is one necessary manual step before you can start building the toolkit. You need to add our Nexus repository to your local maven settings. Do this by **locating or creating the settings.xml file in your local home folder:** `%YOUR_HOME_FOLDER%/.m2/settings.xml` and adding the following content:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
<mirrors>
    <mirror>
      <!--This sends everything else to /public -->
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>http://www.effingo.de/nexus/content/groups/public</url>
    </mirror>
  </mirrors>
  <profiles>
    <profile>
      <id>nexus</id>
      <!--Enable snapshots for the built in central repo to direct -->
      <!--all requests to nexus via the mirror -->
      <repositories>
        <repository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
```

```
          <snapshots><enabled>true</enabled></snapshots>
        </repository>
      </repositories>
     <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
    <!--make the profile active all the time -->
    <activeProfile>nexus</activeProfile>
  </activeProfiles>
  <servers>
    <server>
      <id>nexus</id>
      <username>your-username</username>
      <password>your-password</password>
      <filePermissions>664</filePermissions>
      <directoryPermissions>775</directoryPermissions>
      <configuration></configuration>
    </server>
  </servers>
</settings>
```

You need to set your username and your password in the server section. This can be obtained by sending an E-Mail to klemens.muthmann@tu-dresden.de stating who is your advisor and why you need to work with the toolkit. After completing the installation perform the following steps to build the toolkit using Maven:

1. Check out the code from SVN!

2. Open your favorite command line.

3. Change to the toolkits root folder.

4. Type `mvn clean install` to start the build process.

There also is an Eclipse plugin, that allows you to issue maven build from within Eclipse. It is quite beta so be careful with it.

2.2   Regularly builds and tests using Hudson CI

Currently the toolkit is build automatically every week using Hudson CI. Be careful to check in only working code or Hudson will send you and your advisor an E-Mail about broken code. If this happens try to fix your code as fast as possible and check in again. You can get details about the problem by logging into Hudson. You can get a login by sending an E-Mail stating your advisor and why you nood to work with the toolkit to klemens.muthmann@tu-dresden.de

2.2.1   The Continuous Integration Game

Hudson supports a game that rewards people commiting code that improves the toolkit and punishing people breaking it. The leader (the person having the most points) is highly valued by the toolkit commiters community. The rules for the game are explained in detail in the next section.

Rules The rules of the game are:

- -10 points for breaking a build

- 0 points for breaking a build that already was broken

- +1 points for doing a build with no failures (unstable builds gives no points)

- -1 points for each new test failures

- +1 points for each new test that passes

- Adding/removing a HIGH priority PMD warning = -5/+5. Adding/removing a MEDIUM priority PMD warning = -3/+3. Adding/removing a LOW priority PMD warning = -1/+1.

- Adding/removing a violation = -1/+1. Adding/removing a duplication violation = +5/-5.

- Adding/removing a HIGH priority findbugs warning = -5/+5. Adding/removing a MEDIUM priority findbugs warning = -3/+3. Adding/removing a LOW priority findbugs warning = -1/+1

- Adding/removing a compiler warning = -1/+1.

- Checkstyle Plugin. Adding/removing a checkstyle warning = -1/+1.

2.3 Hello Toolkit - Your first application using the TUDIIR Toolkit

Project creation: Create a new Maven Project using the New Project wizard of Eclipse (See Fig. 2.1, 2.2 and 2.3).

Adding the toolkit dependency to the project: Right click on the new project. In the context menu that appears choose *Maven → Add Dependency* (See Fig. 2.4 and 2.5). A search interface appears (See 2.6). If you followed the steps in Section 2.1 and installed the toolkit to your local Maven repository, you can type *toolkit* in the search interface (See Fig. 2.7 and add the dependency with a double click on the *de.tud.inf.rn.iir toolkit* entry.

Note: If you need to add further dependencies in the future you can use the same steps. The Maven plugin adds the dependency to your pom.xml file, which should look like Fig. 2.8.

Configuring your project Since Maven by default still uses Java 1.4 (very conservative), but the toolkit depends on Java 1.6 (very visionary) you need to configure the Maven Java compiler plugin to use Java 1.6. This is shown in Fig. 2.9. The same step is necessary to tell Eclipse that it should use Java 1.6 instead of 1.4. For this purpose open the project properties via the context menu for example. In the tree to the left choose *Java Compiler* and change all three entries to 1.6. This is shown in Fig. 2.10 and Fig. 2.11.

Writing your first toolkit code Now you can start to write your first code. Your project will already contain the default Maven directory structure. Do not change this structure since Maven depends on it[1]. As usual we will start with a very simple "Hello World" application. Create a new package `de.tud.inf.rn.iir.toolkit` and a new class `HelloToolkit` containing a `main` method. In this main method you can add actual toolkit code. We used the first example as described in Section 5.3.1 and search Bing for the term "Hello Toolkit". The example is also shown in Fig. 2.12. For the code to work you need three additional files that are already in the config folder in the toolkit project. These are "crawler.conf", "apikeys.conf" and "feeds.conf". You need to copy them to src/main/resources/config. The folder src/main/resources usually contains all resources that are not Java files but are required by your code. All files are shown in Fig. 2.13, 2.14 and 2.15. Fig. 2.16 shows the final directory and file structure of your project. With the structure from Fig. 2.16 you
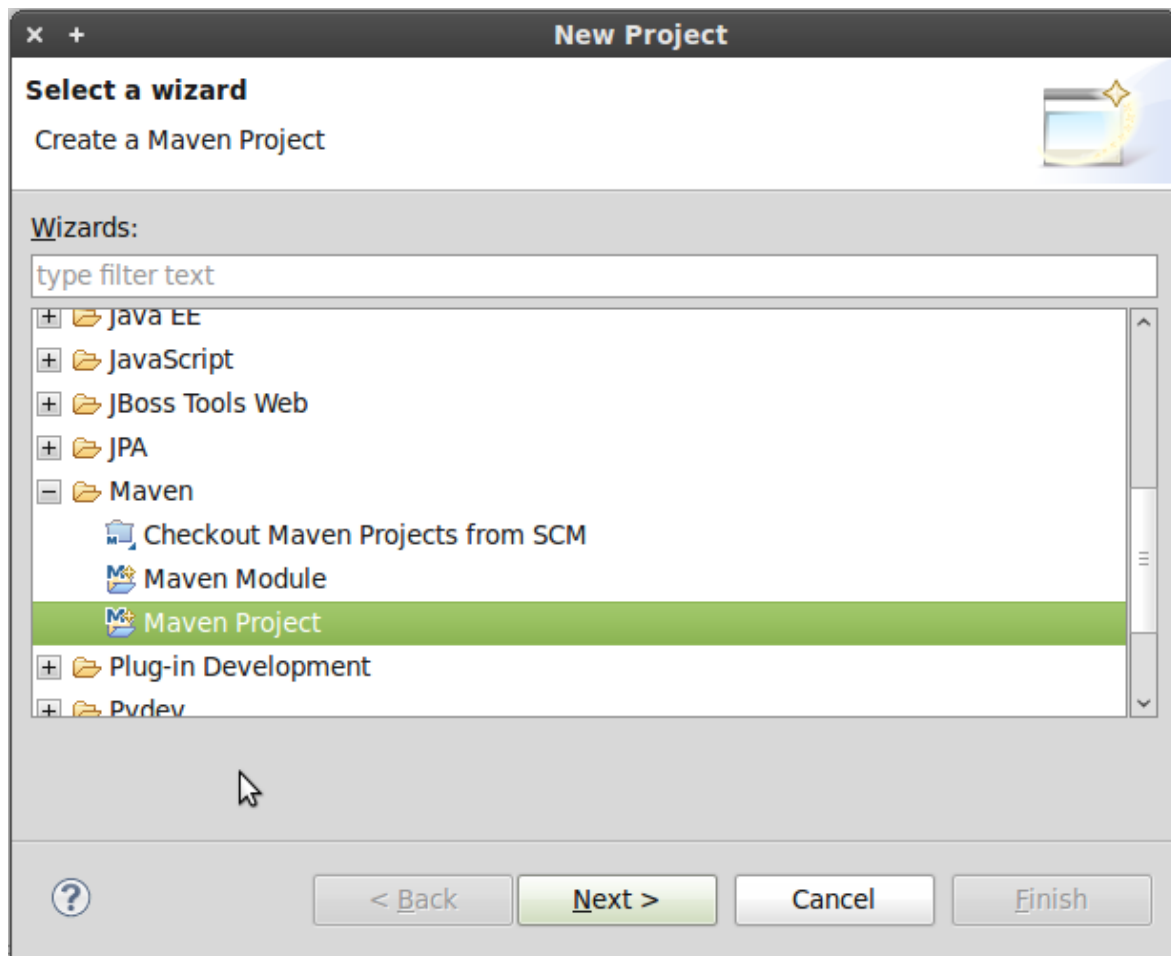
Figure 2.1: Create a new Maven Project.
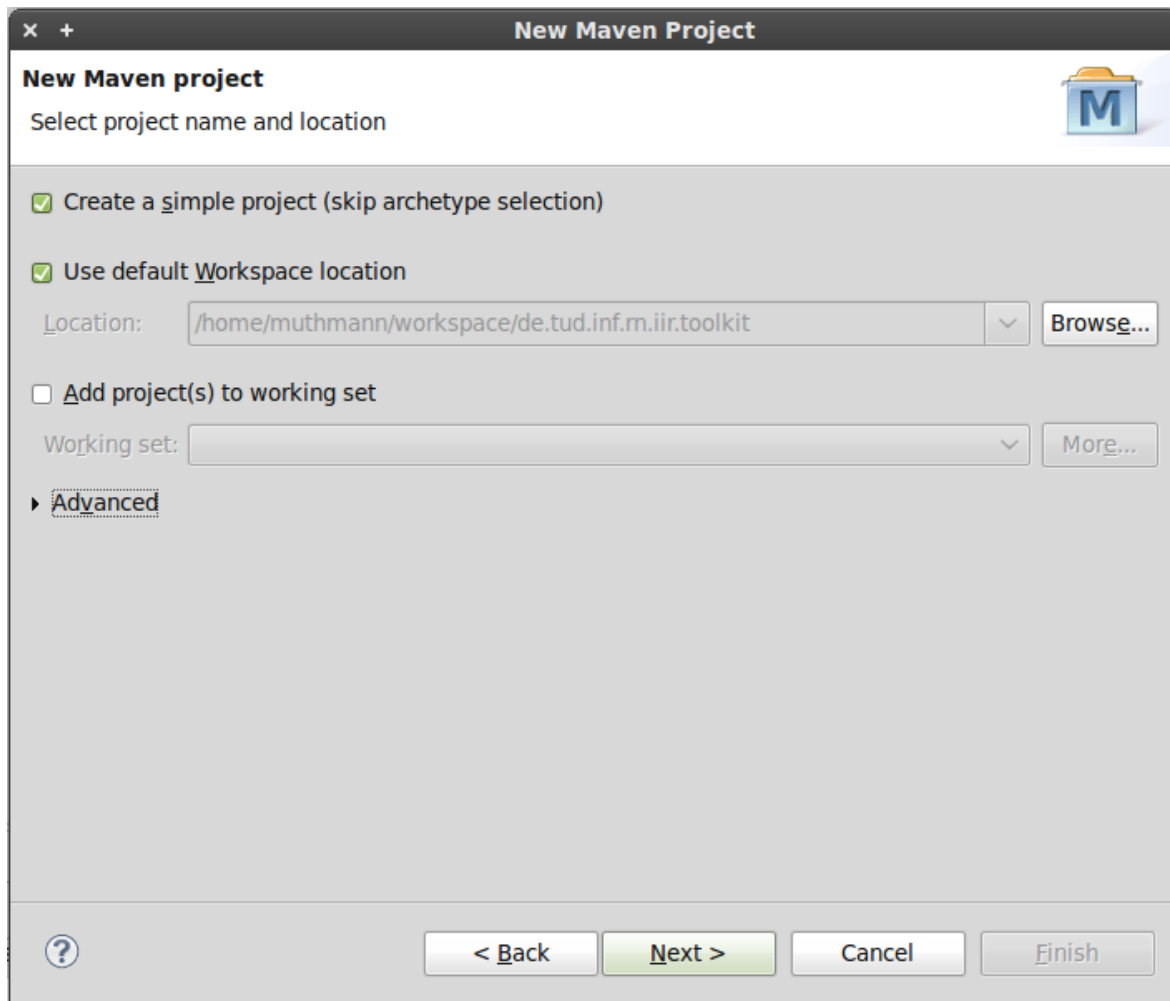
Figure 2.2: Choose to create a simple Maven Project.

Figure 2.3: Enter detail information about your new Maven Project
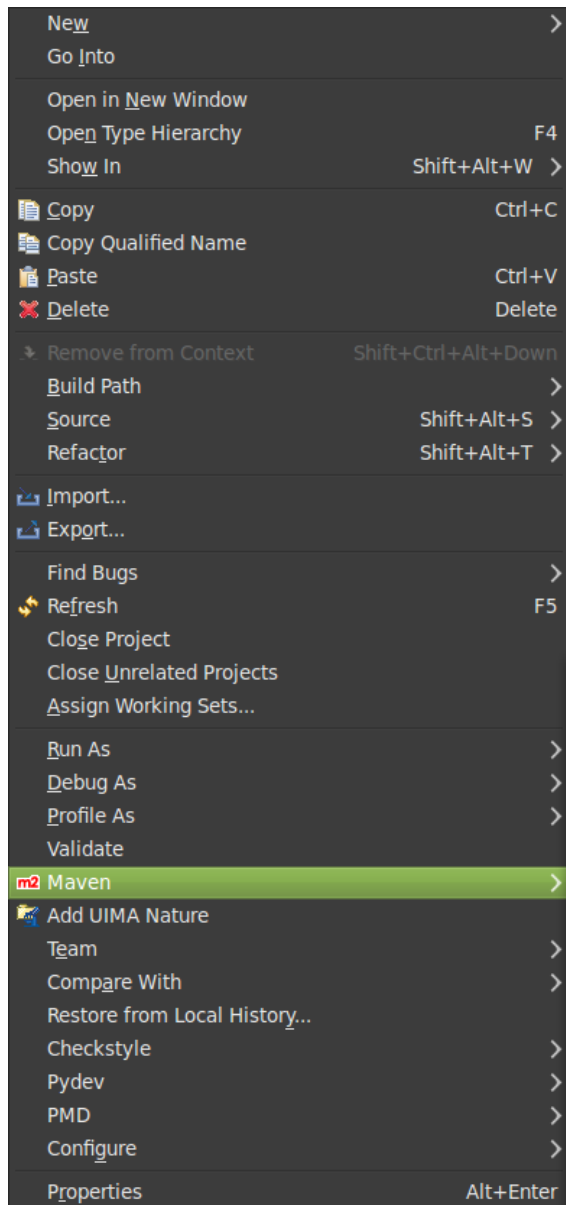
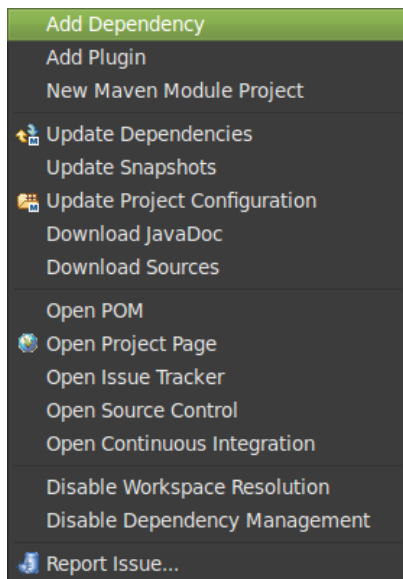Figure 2.4: Maven Project Context Menu. Choose *Maven*
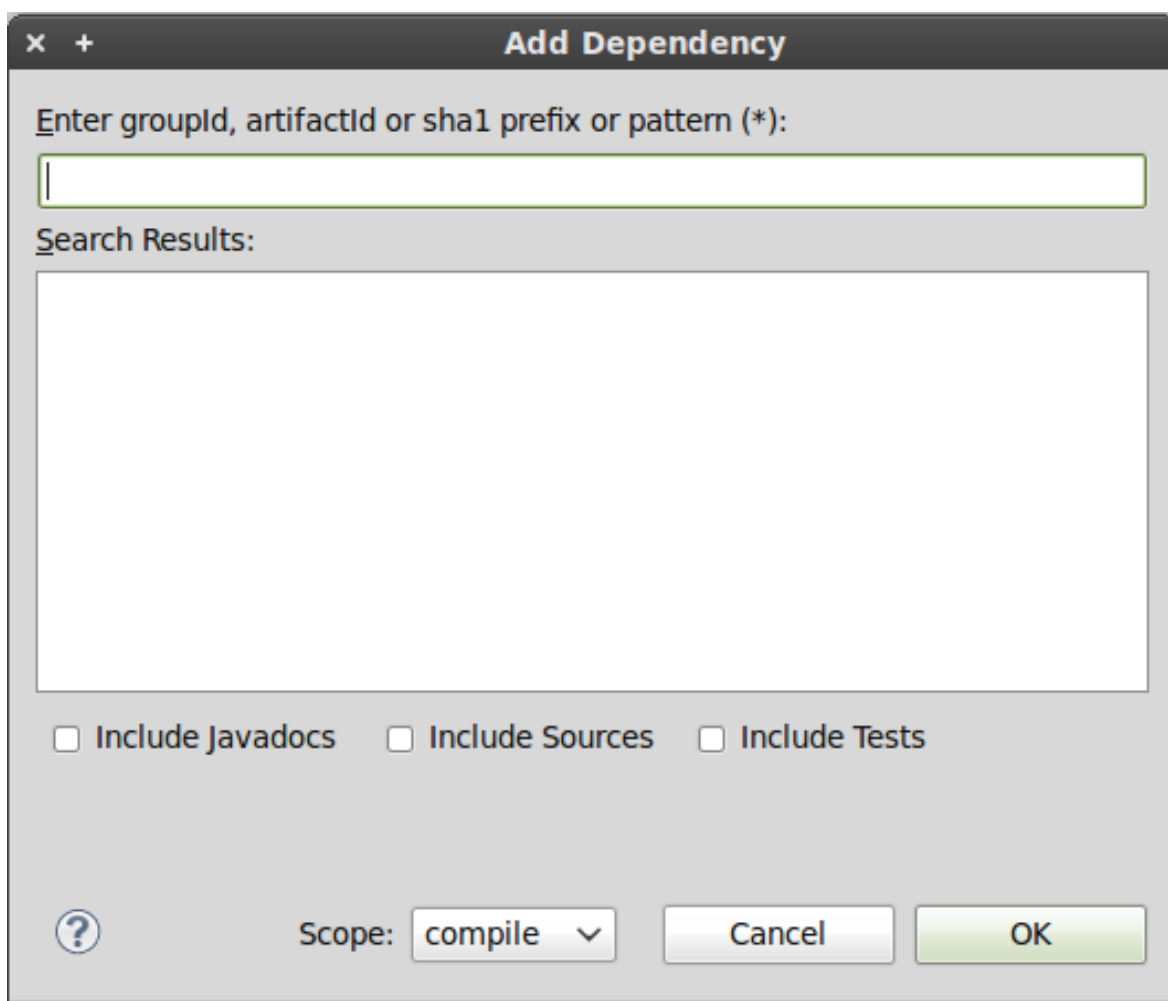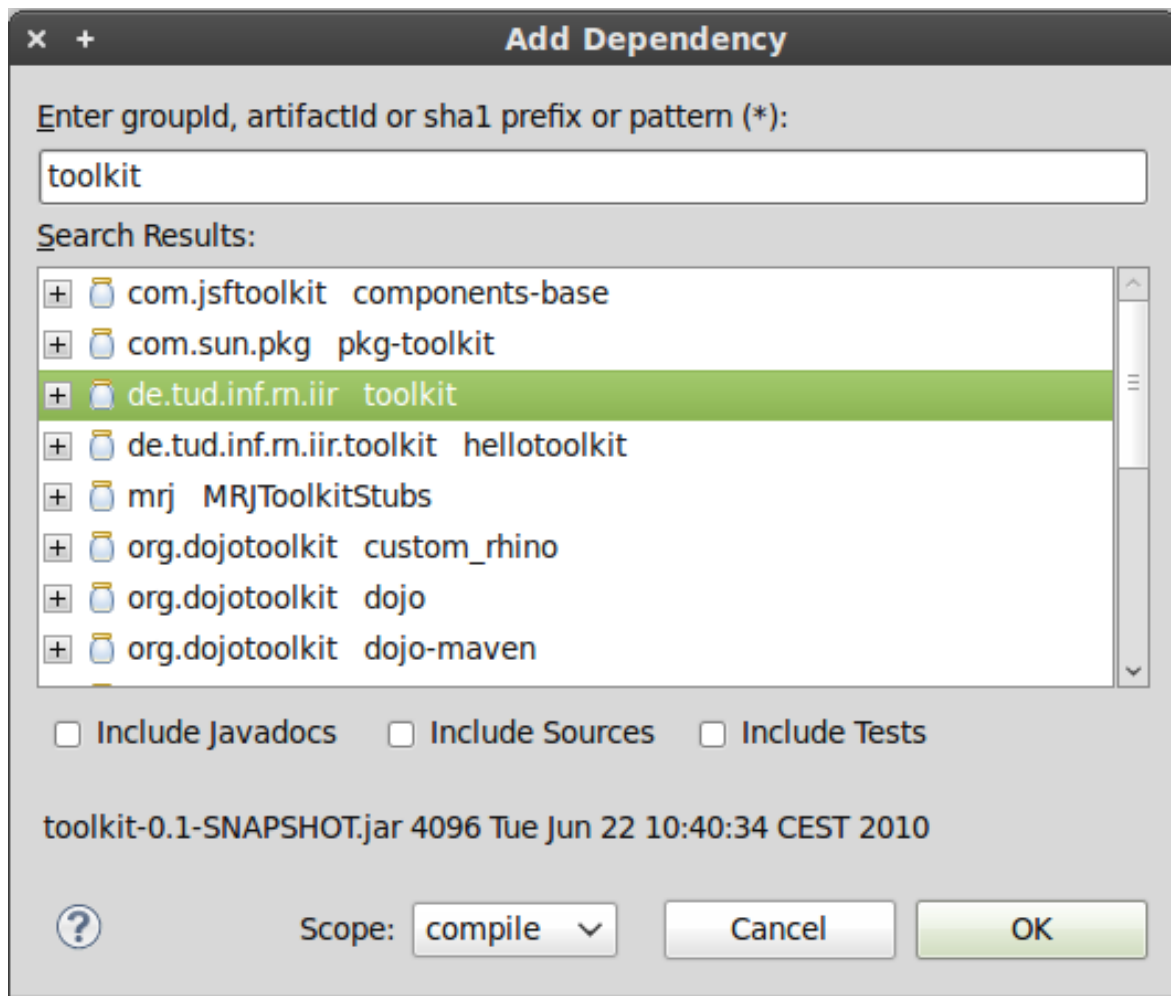
Figure 2.5: Create a new Maven project

Figure 2.6: Search interface for Maven dependencies.
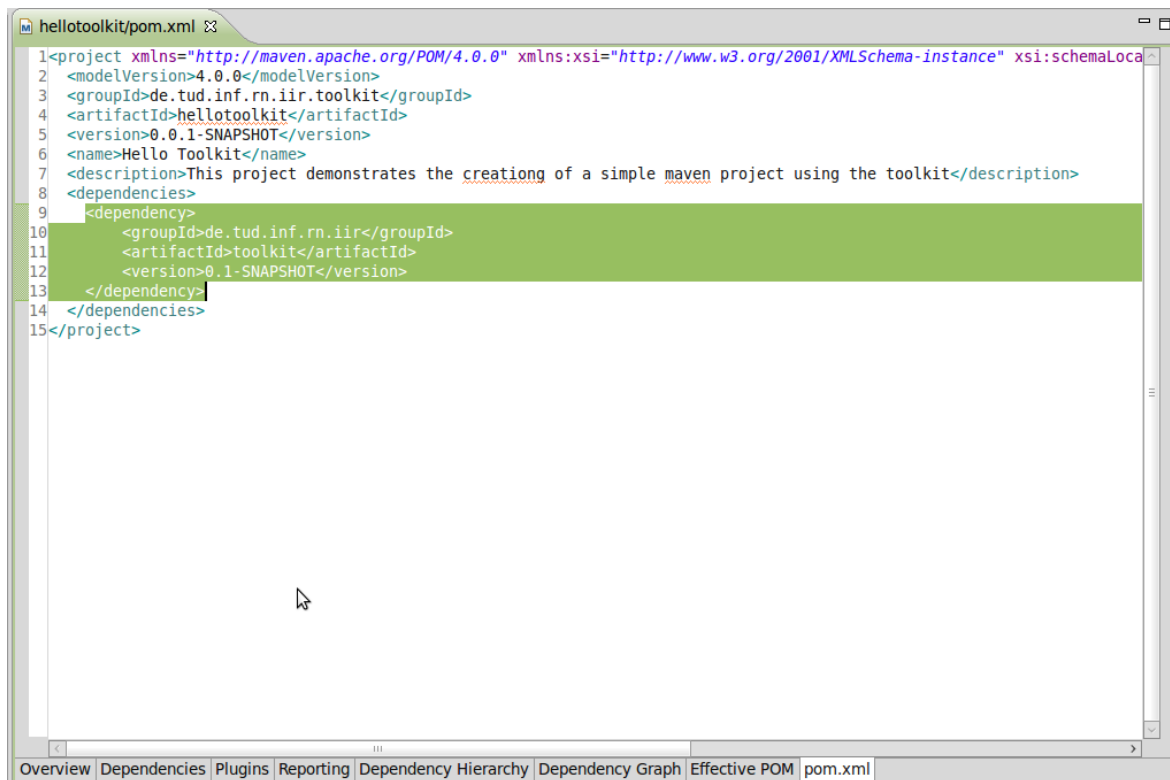
Figure 2.7: Search results for *toolkit*

```
hellotoolkit/pom.xml ✕
 1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLoca
 2   <modelVersion>4.0.0</modelVersion>
 3   <groupId>de.tud.inf.rn.iir.toolkit</groupId>
 4   <artifactId>hellotoolkit</artifactId>
 5   <version>0.0.1-SNAPSHOT</version>
 6   <name>Hello Toolkit</name>
 7   <description>This project demonstrates the creationg of a simple maven project using the toolkit</description>
 8   <dependencies>
 9     <dependency>
10       <groupId>de.tud.inf.rn.iir</groupId>
11       <artifactId>toolkit</artifactId>
12       <version>0.1-SNAPSHOT</version>
13     </dependency>
14   </dependencies>
15 </project>
```

Overview | Dependencies | Plugins | Reporting | Dependency Hierarchy | Dependency Graph | Effective POM | pom.xml

Figure 2.8: POM with added dependency on the TUD IIR Toolkit.

```
hellotoolkit/pom.xml ✕     *HelloToolkit.java     de.tud.inf.rn.iir.toolkit/pom.xml
 1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
 3   <modelVersion>4.0.0</modelVersion>
 4   <groupId>de.tud.inf.rn.iir.toolkit</groupId>
 5   <artifactId>hellotoolkit</artifactId>
 6   <version>0.0.1-SNAPSHOT</version>
 7   <name>Hello Toolkit</name>
 8   <description>This project demonstrates the creationg of a simple maven project using the toolkit</description>
 9   <build>
10     <plugins>
11       <plugin>
12         <groupId>org.apache.maven.plugins</groupId>
13         <artifactId>maven-compiler-plugin</artifactId>
14         <version>2.3.1</version>
15         <configuration>
16           <compilerArgument>-Xlint:all</compilerArgument>
17           <showWarnings>true</showWarnings>
18           <source>1.6</source>
19           <target>1.6</target>
20           <compilerArguments>
21             <encoding>UTF-8</encoding>
22           </compilerArguments>
23           <showDeprecation>true</showDeprecation>
24           <verbose>true</verbose>
25           <encoding>UTF-8</encoding>
26         </configuration>
27       </plugin>
28     </plugins>
29   </build>
30   <dependencies>
31     <dependency>
32       <groupId>de.tud.inf.rn.iir</groupId>
33       <artifactId>toolkit</artifactId>
34       <version>0.1-SNAPSHOT</version>
35     </dependency>
```

Overview | Dependencies | Plugins | Reporting | Dependency Hierarchy | Dependency Graph | Effective POM | pom.xml
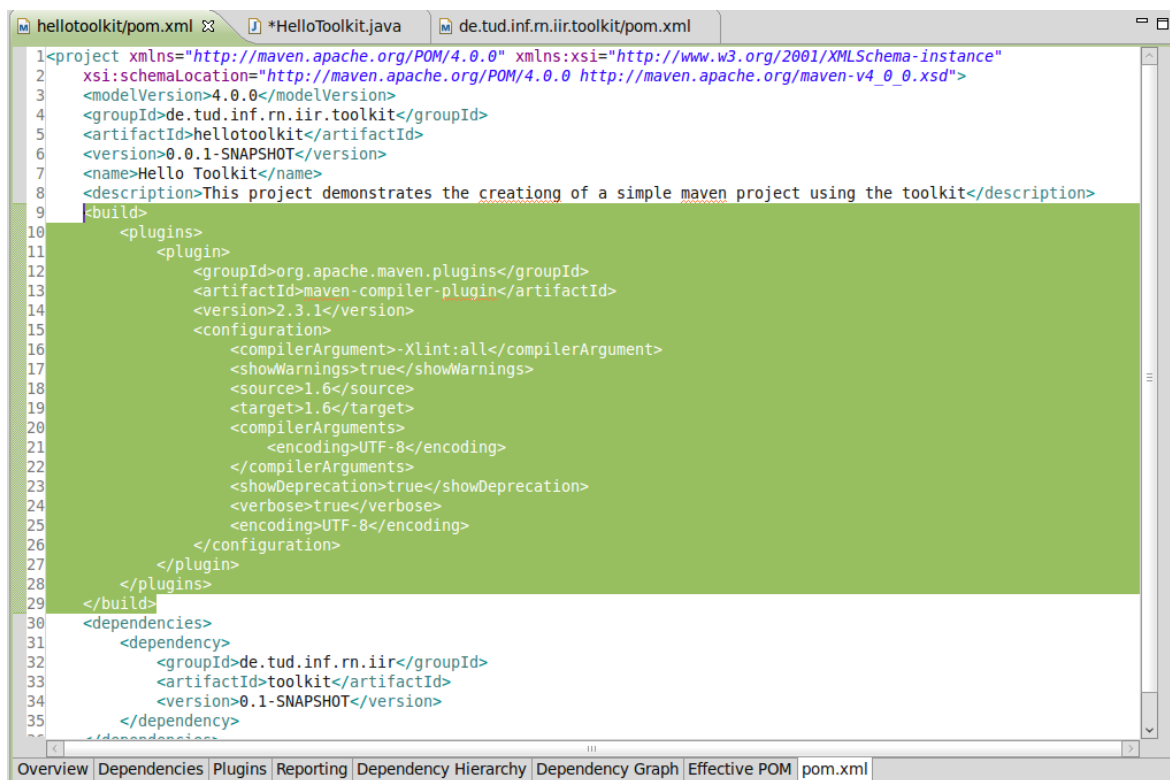
Figure 2.9: Configure the Maven Java compiler to use Java 1.6 instead of 1.4
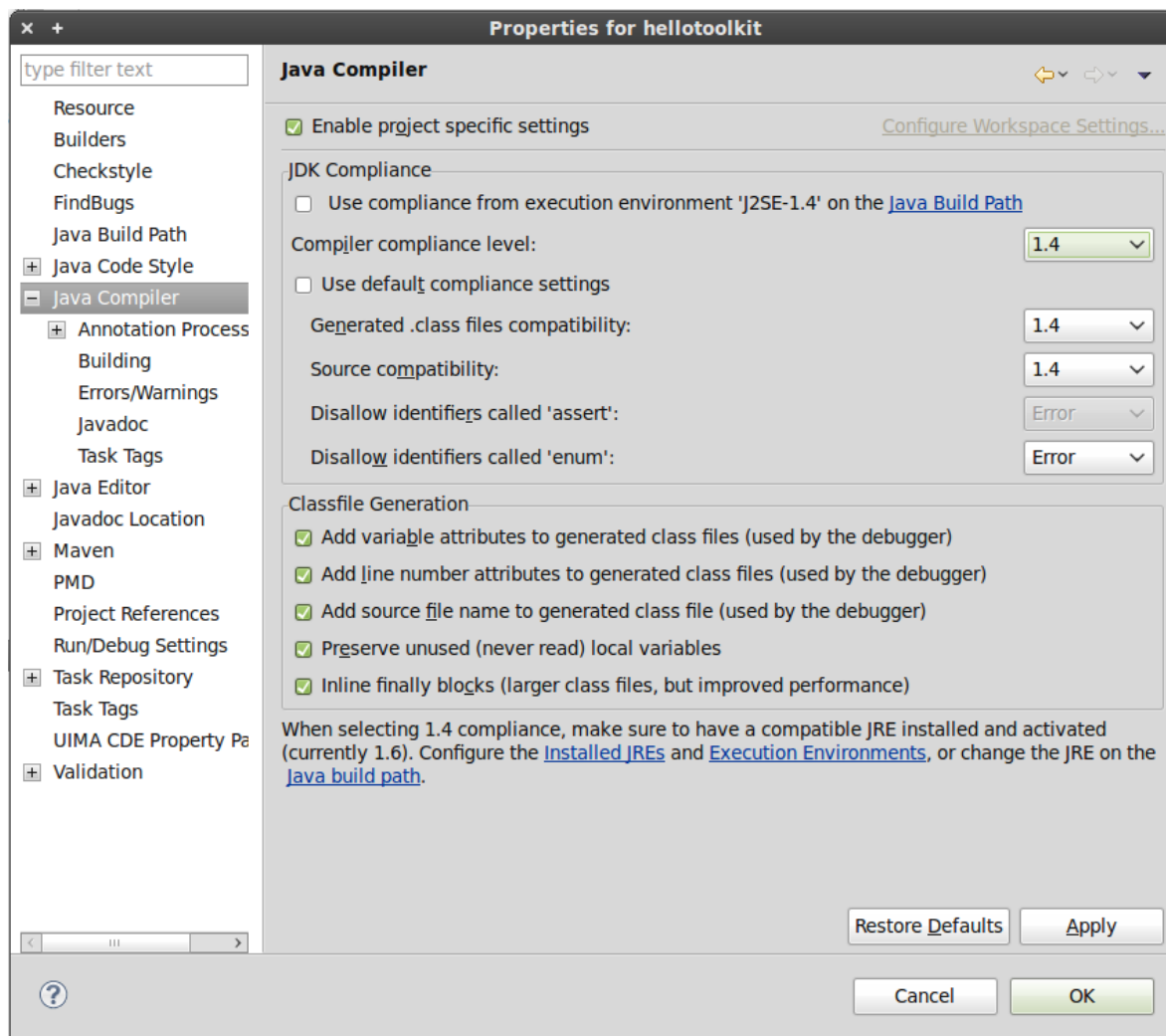
Figure 2.10: Eclipse uses Java 1.4 by default for Maven projects

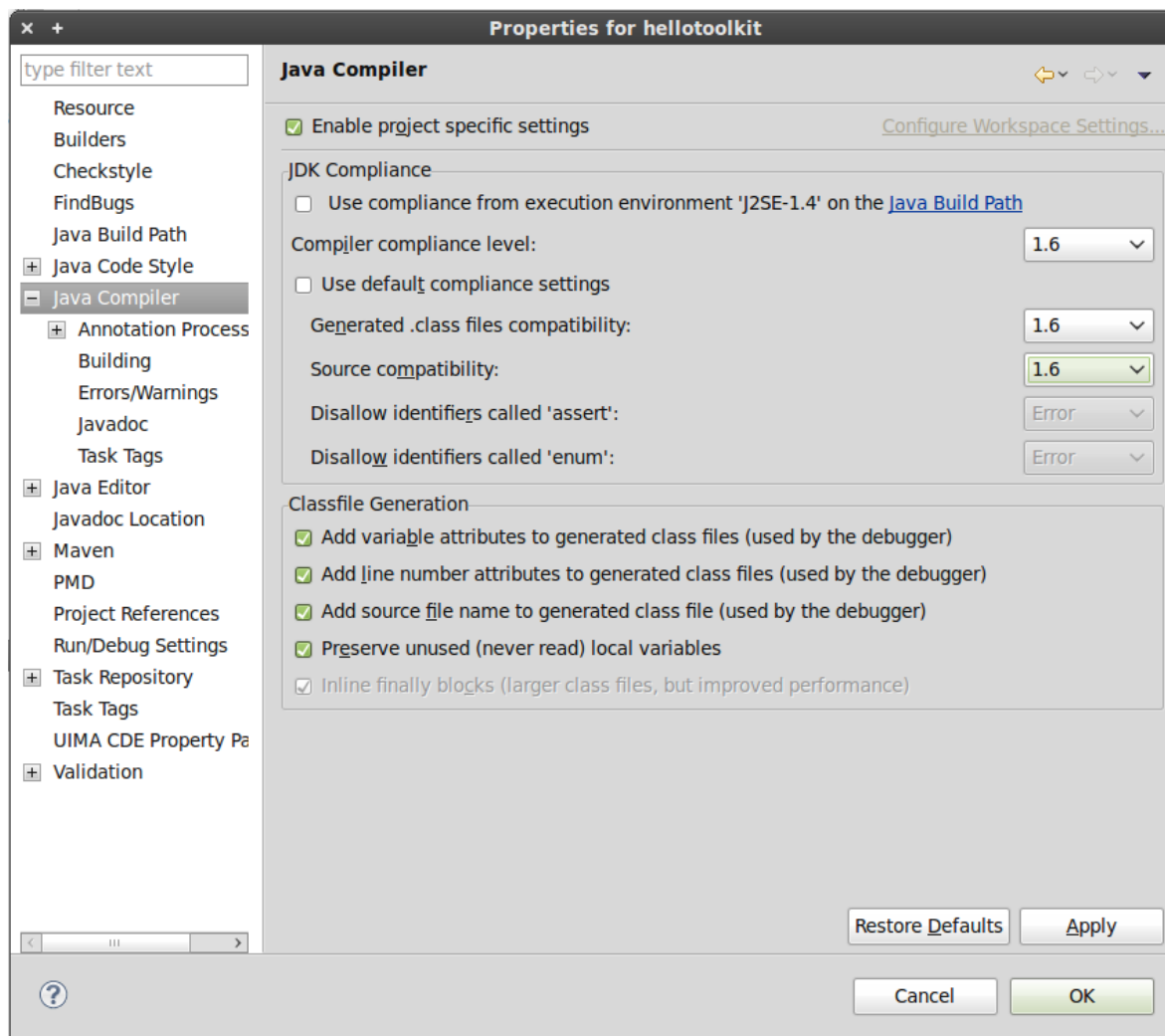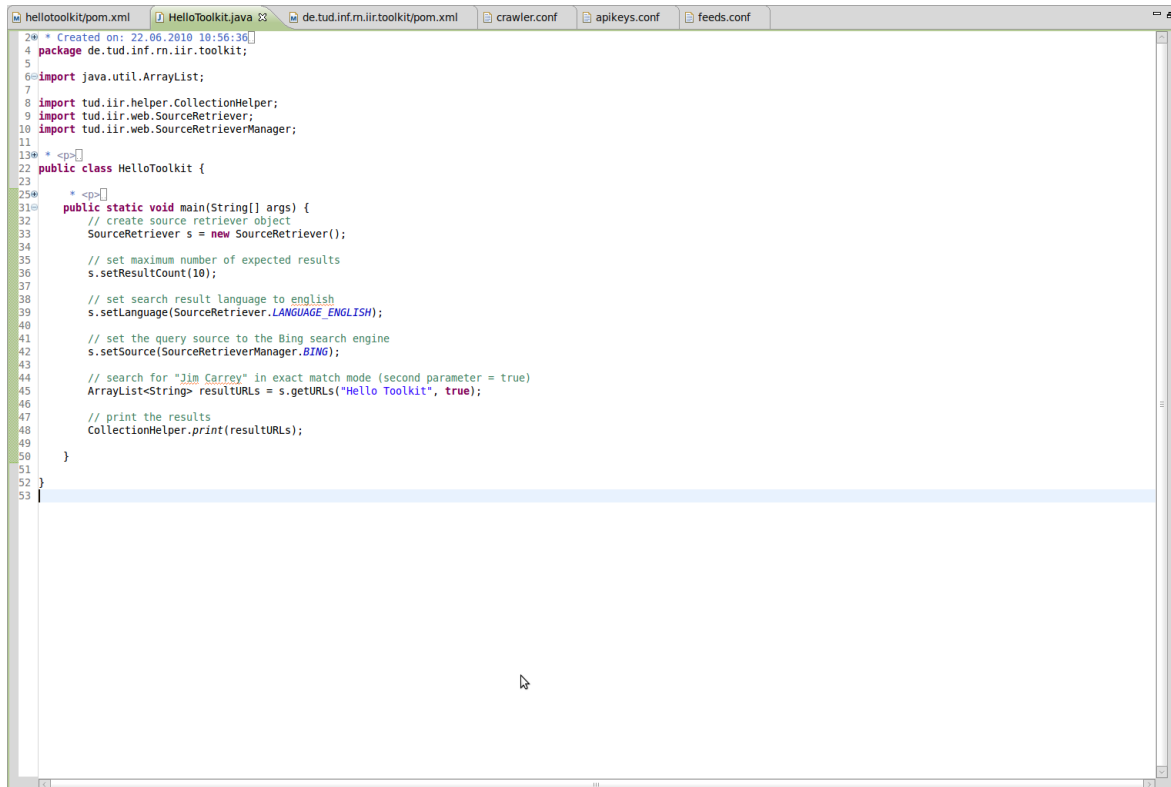Figure 2.11: configure Eclipse to use Java 1.6

Figure 2.12: "Hello Toolkit" Code
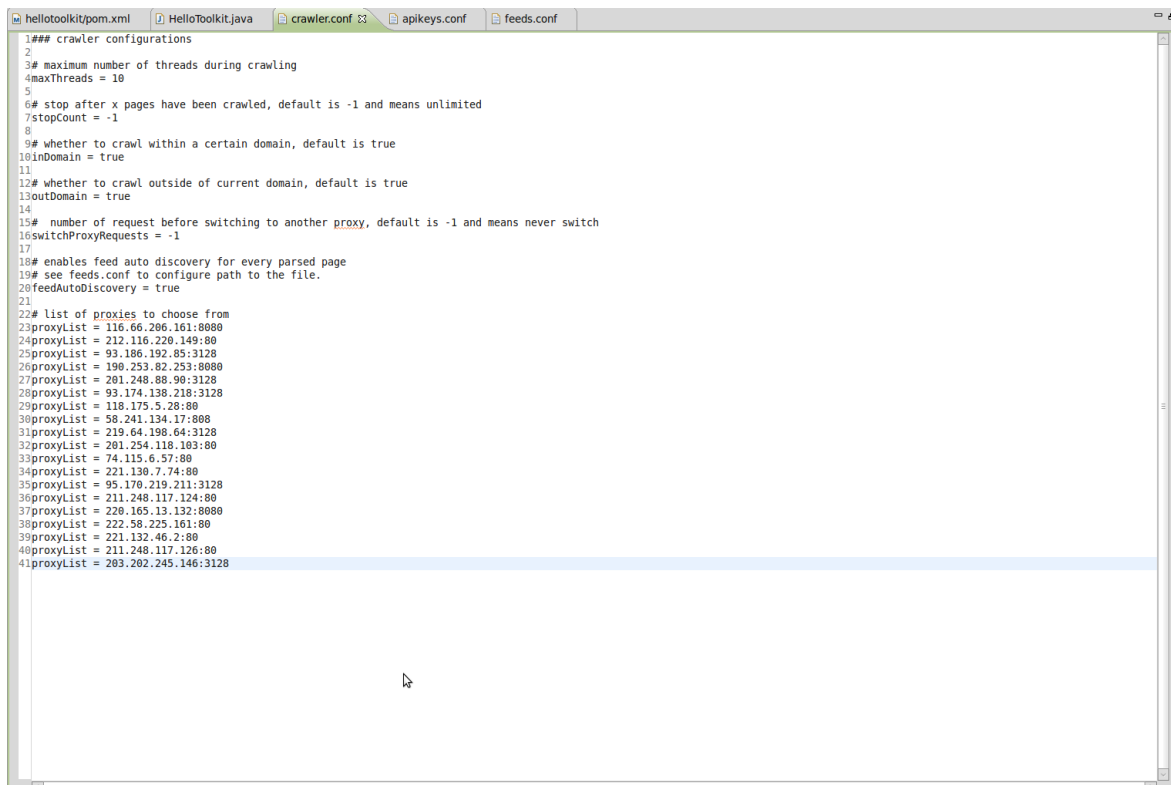

Figure 2.13: Create a new Maven project
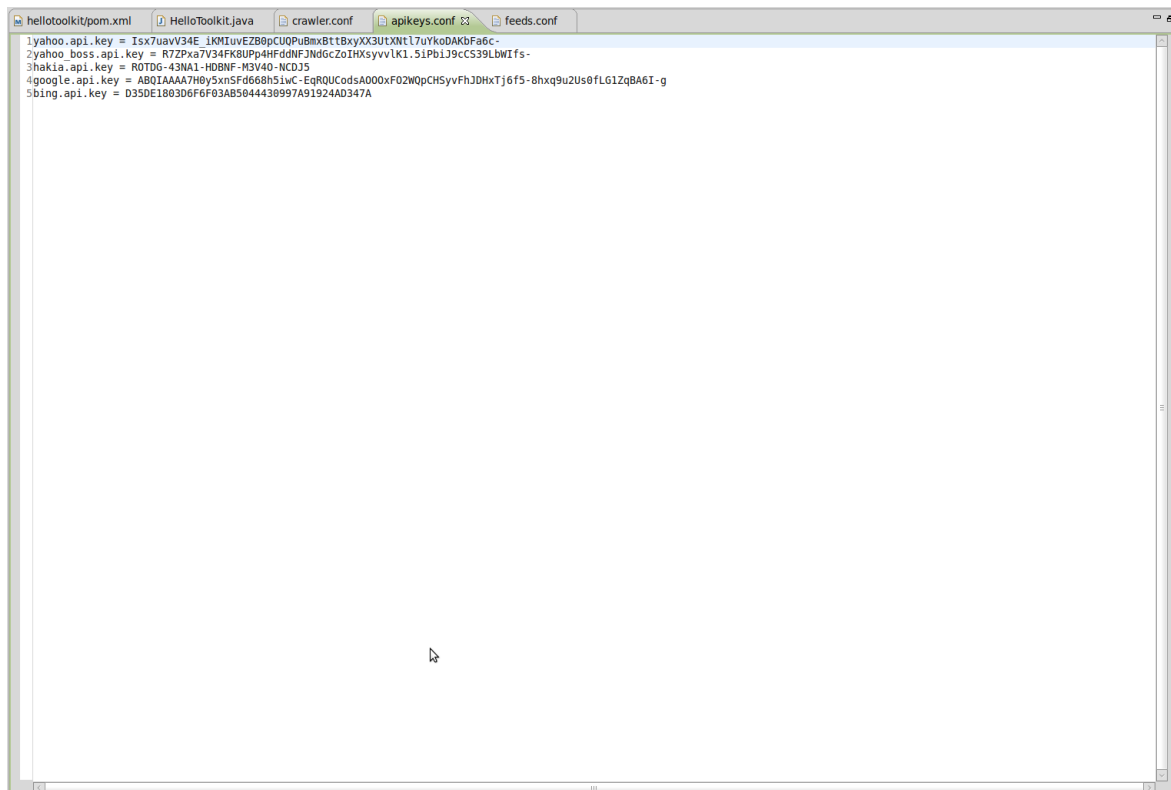
Figure 2.14: Create a new Maven project
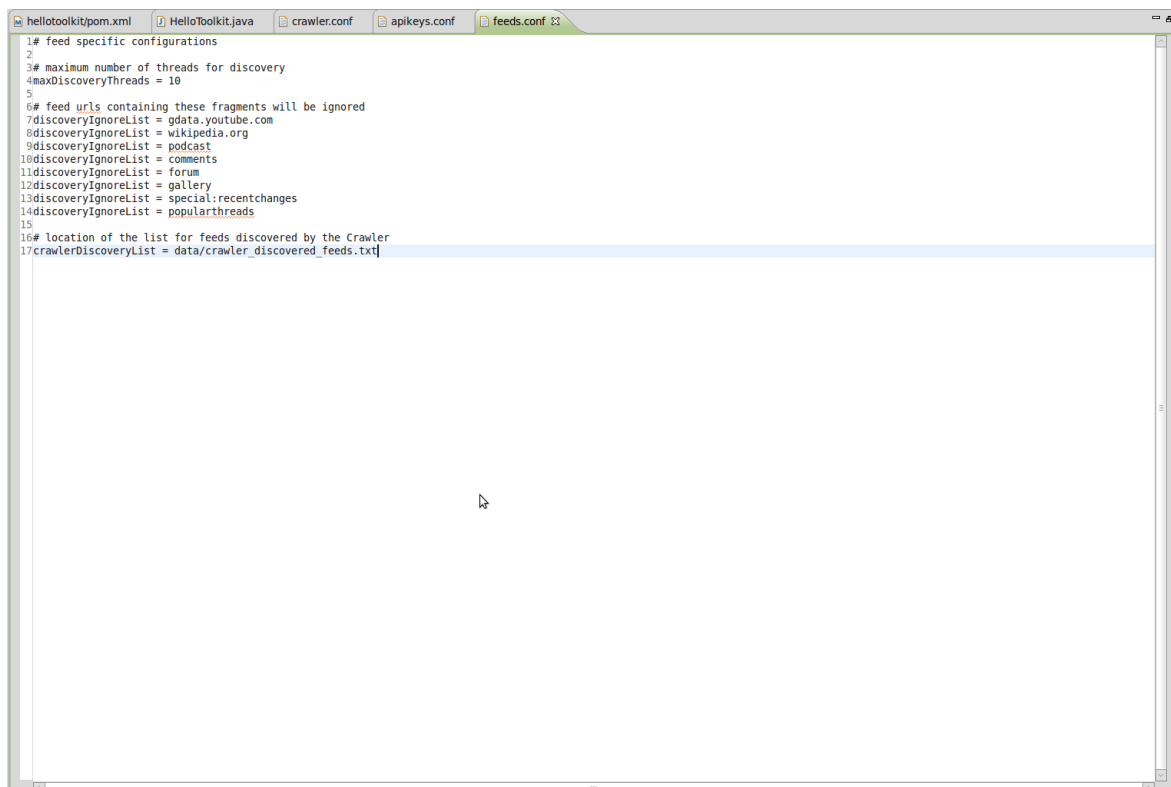
Figure 2.15: Create a new Maven project
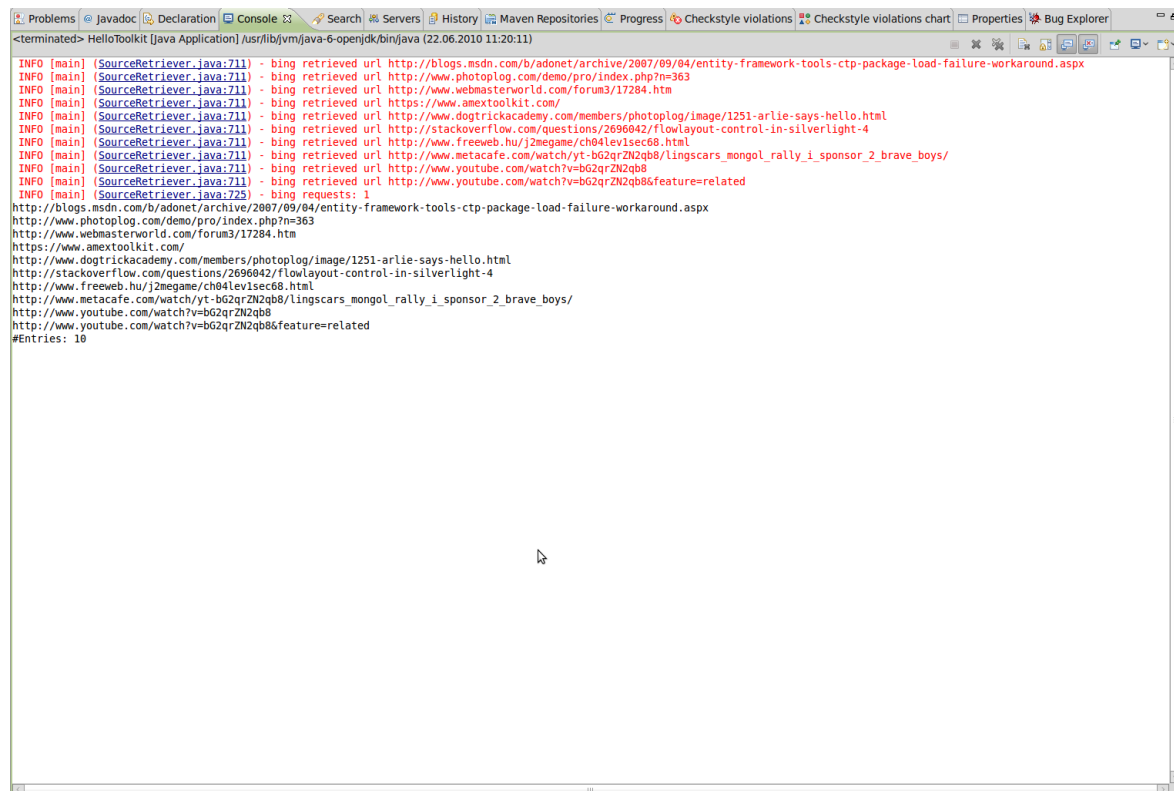
Figure 2.16: Create a new Maven project

can run your project. Just open your projects context menu again, choose *Run As → Maven Clean*, then open context menu again choose *Run As → Maven Install* and finally choose *Run As → Java Application*.

2.4 Reporting issues using Mantis

To report issues with the toolkit or to view issues assigned to you, you need to register with Mantis. To do this send an E-Mail to the administrator at klemens.muthmann@tu-dresden.de and provide your name, your advisor and the reason why you are working on the toolkit (i.e. thesis topic or research fellow). You will receive an E-Mail (not automatically so it might take some time) providing a link were you can set your password. If you successfully set a password for your account you can login on: Mantis.

---

[1]Of course you can change Mavens behaviour via the pom.xml but this requires additional configuration not covered by this document. Refer to the Maven documentation under `http://maven.apache.org/` for further information.

Chapter 3

Toolkit Structure

The TUDIIR Toolkit is managed using subversion. The top level folder structure follows the usual subversion layout using trunk for the main development, branches for parallel development and tags to mark specific versions. The trunk is located in our SVN[1]. The folder structure looks as follows.

```
toolkit
 |- config
 |- data
    |- datasets
    |- knowledgeBase
    |- models
    |- test
 |- documentation
    |- handout
    |- javadoc
 |- exe
 |- src
    |- main
       |- java
    |- test
       |- java
 |- dev
```

3.1   Config Folder

The config folder contains configuration files for several components of the toolkit. The files are explained in the following sections.

3.1.1   apikeys.conf

The api keys that are used by the toolkit components are specified here. You may need to apply for API keys at the provider's page.

```
yahoo.api.key =
yahoo_boss.api.key =
hakia.api.key =
google.api.key =
bing.api.key =
```

---

[1]https://141.76.40.86/svn-students/iircommon/toolkit/trunk

3.1.2   classification.conf

Classification settings for the tud.iir.classification.page.ClassifierManager.

```
# percentage of the training/testing file to use as training data
page.trainingPercentage = 80

# create dictionary on the fly (lowers memory consumption but is slower)
page.createDictionaryIteratively = false

# alternative algorithm for n-gram finding (lowers memory consumption but is slower)
page.createDictionaryNGramSearchMode = true

# index type of the classifier:
# 1: use database with single table (fast but not normalized and more disk space needed)
# 2: use database with 3 tables (normalized and less disk space needed but slightly slower)
# 3: use lucene index on disk (slow)
page.dictionaryClassifierIndexType = 1
```

3.1.3   crawler.conf

Crawler settings for the tud.iir.web.Crawler. All settings can be set in Java code as well.

```
# maximum number of threads during crawling
maxThreads = 10

# stop after x pages have been crawled, default is -1 and means unlimited
stopCount = -1

# whether to crawl within a certain domain, default is true
inDomain = true

# whether to crawl outside of current domain, default is true
outDomain = true

#  number of request before switching to another proxy, default is -1 and means never switch
switchProxyRequests = -1

# list of proxies to choose from
proxyList = 83.244.106.73:8080
proxyList = 83.244.106.73:80
proxyList = 67.159.31.22:8080
```

3.1.4   db.conf

Database settings for the tud.iir.persistence.DatabaseManager.

```
db.type = mysql
db.driver = com.mysql.jdbc.Driver
db.host = localhost
db.port = 3306
db.name = toolkitdb
db.username = root
db.password = rootpass
```

3.1.5 general.conf

General settings used by the tud.iir.control.Controller.

3.2 Data Folder

The data folder contains files that are used during runtime of several components.

3.2.1 knowledgeBase

The knowledge base folder contains the OWL ontology files used for the extraction tasks in the tud.iir.extraction package.

3.2.2 models

The models folder contains learned models that can be reused.

3.2.3 Temp Folder

The temp folder is not part of the repository. Some functions may however create this folder and write temporary data.

3.2.4 test

The test folder contains data that is used for running jUnit tests.

3.3 Documentation Folder

The documentation folder contains help files to understand the toolkit. This very document is located in the handout folder and a Javadoc can be found there too.

3.4 Exe Folder

The exe folder contains all runnable jar files in separate folders including a sample script to run the program and a readme.txt that explains the run options.

3.5 Libs Folder

The libs folder contains all referenced libs used by the toolkit.

3.6 Src Folder

The src folder contains all source files of the toolkit. You may need to put the log4j.properties file here in order to use custom logging settings. Alternatively you include the config folder in the class path.

Chapter 4

Conventions

4.1 Coding Standards

To keep the code readable and easy to understand for other developers, we use the following coding guidelines.

1. All text is written in (American) English (color instead of colour).

2. Variables and method names should be camel cased and not abbreviated (*computeAverage* instead of *comp_avg*).

3. There must be a space after a comma.

4. There must be a line break after opening { braces.

5. Static fields must be all uppercase. There should be an _ for longer names ($STATIC\_FIELD$).

6. Each class must have a comment including the author name.

7. Methods with very simple, short code do not need to have comments (getters and setters) all other methods should have an explaining comment with @param explanation and @return.

8. Avoid assignments (=) inside if and while conditions.

9. Statements after conditions should always be in braces ({})

The following listing shows an example class with applied coding standards. Please also have a look at [14] for a quick overview of best practices.

```java
1  /**
2   * This is just an example class.
3   * It is here to show the coding guidelines.
4   *
5   * @author Forename Name
6   */
7  public class ExampleClass implements Example {
8
9          // this field holds all kinds of brackets
10         private static final char[] BRACKET_LIST = {'(', ')'};
11
12         /**
13          * This is just and example method.
14          *
15          * @param timeString A string with a time.
16          * @return True if no error occurred, false otherwise.
```

```
17              */
18          public boolean computeAverageTime(String timeString) {
19                  if (hours < 24 && minutes < 60 && seconds < 60) {
20                          return true;
21                  } else {
22                          return false;
23                  }
24          }
25 }
```

4.2 Eclipse Plugins for better Coding

It is a very good practice to install the following eclipse plugins to check ones own code before committing:

1. CodeFormatter is a simple file that can be loaded into Eclipse to format the source code correctly. Go to Eclipse ▶ Window ▶ Preferences ▶ Java ▶ Code Style ▶ Formatter and import the "tudiir_eclipse_formatter.xml" from the dev folder. Pressing Control+F formats the source code of a selected class.

2. Checkstyle [1] checks styles of the code, whether JavaDoc comments are set etc. After installing you should go to the preferences section of Checkstyle in Enclipse and load the "checkstyle_config.xml" from the dev folder.

3. PMD[2] tells you what is wrong with your code in terms of common violations, forgotten initializations and much more. After installing you should go into the preferences of PMD and load the ruleset file "pmd_ruleset" from the dev folder.

4. FindBugs[3] is similar to PMD but focuses on severe errors only. After installing you don't need to configure anything.

Checkstyle, PMD, and FindBugs can be initiated by right clicking a package or class file and selecting the plugin. The violations will be shown so that you can eliminate them.

Using these plugins raises the chances to win in the continuous integration game as described in Section 2.2.1.

4.2.1 Tests

To guarantee that all components work as expected, we use jUnit tests. Before major check-ins to the repository, all jUnit Tests must run successfully. Run the tud.iir.control.AllTests.java to make sure all components work correctly. After finishing a new component, new testing code must be written.

---

[1] http://eclipse-cs.sourceforge.net/downloads.html
[2] http://pmd.sourceforge.net/eclipse/
[3] http://findbugs.cs.umd.edu/eclipse/

Chapter 5

Toolkit Functionality

5.1 Classification

5.1.1 Text Classification

Text classification is the process of assigning one or more categories to a given text document. There are several types of text classification which are shown in Figure 5.1. The TOOLKIT can classify text in a single category, multiple categories, or in a hierarchical manner.

The text classification components are built from scratch and do not rely on external libraries such as Weka. In this section we will explain which features can be used for the classification, the basic theory of the classifiers, and how the performance of a classifier can be evaluated. In each section, we will describe theory and how the classification components can be used programmatically.

Classification Type

The TOOLKIT supports simple single category classification, tagging, and hierarchical classification. Each classifier needs to know in which type it has to perform classification. This information is stored in the *tud.iir.classification.page.evaluation.ClassificationTypeSetting* object which is then passed to the classifier. Please read the Javadoc of that class for more detailed information.

Features

Features are the input for a classifier. In text classification we have a long string as an input from which we can derive several features. All of the TOOLKIT classifiers work with n-grams. N-grams are sets of tokens of the length n. The TOOLKIT can preprocess text with character or word-level n-grams. See Section 5.4.3 for more details.

Sometimes you do not want to have all n-grams to be features for the classifier. You can simply disallow certain features by putting them in a stop word list. These n-grams will then be ignored for the classification.

All these settings are stored in a *tud.iir.classification.page.evaluation.FeatureSetting* object which is passed to the classifier. Please read the Javadoc of that class for more detailed information.

Text Classifiers

The text classifiers perform the actual classification task by calculating the most relevant category (or categories) for the input document. Two text classifiers are implemented, a dictionary based classifier and a k-nearest neighbor classifier. Both are explained in more detail in the following paragraphs.

K-Nearest Neighbor Text Classifier   The KNN classifier uses the n-grams of the training documents to place them in a high dimensional vector space. The dimensions of the space equal the total
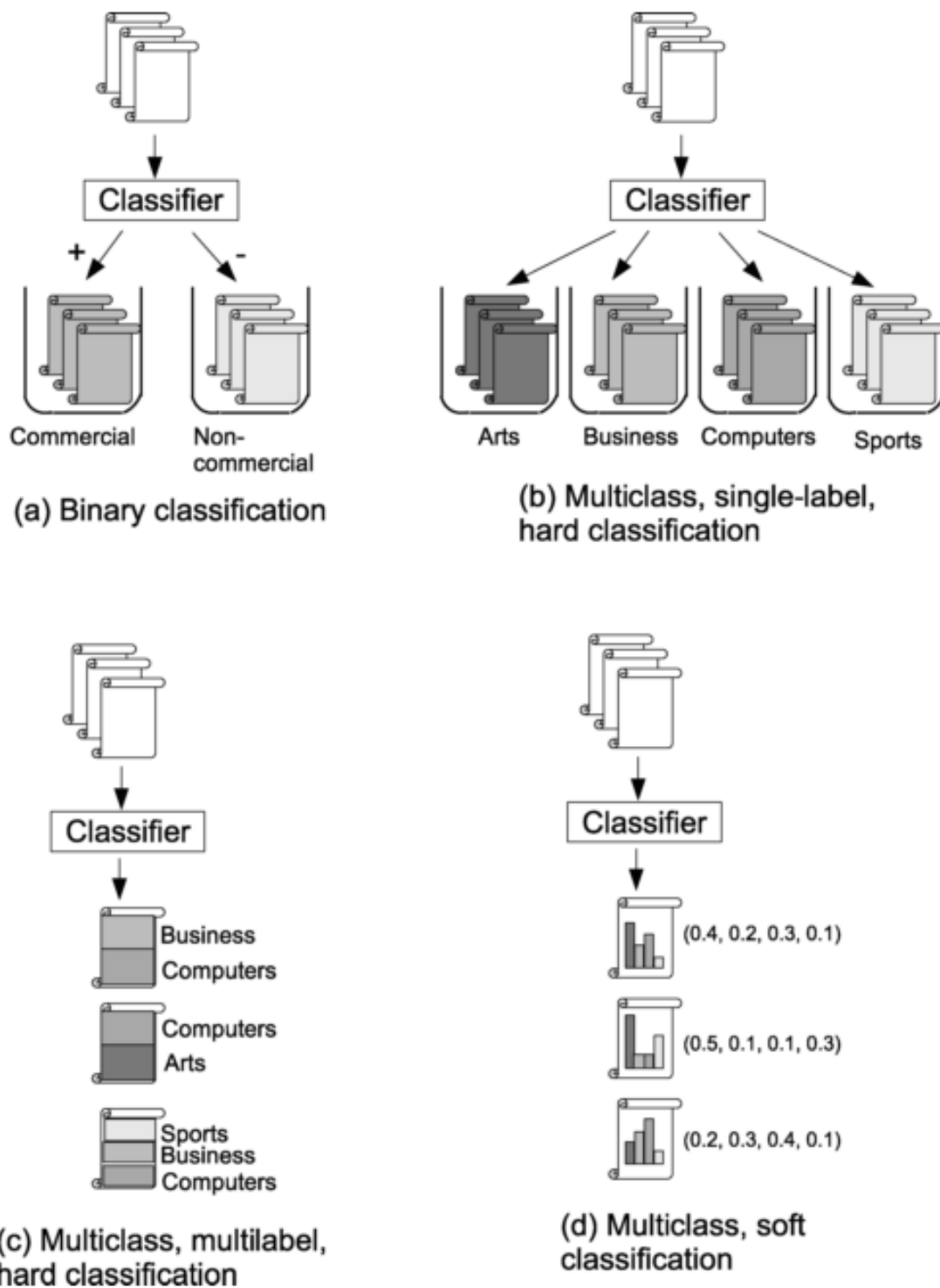
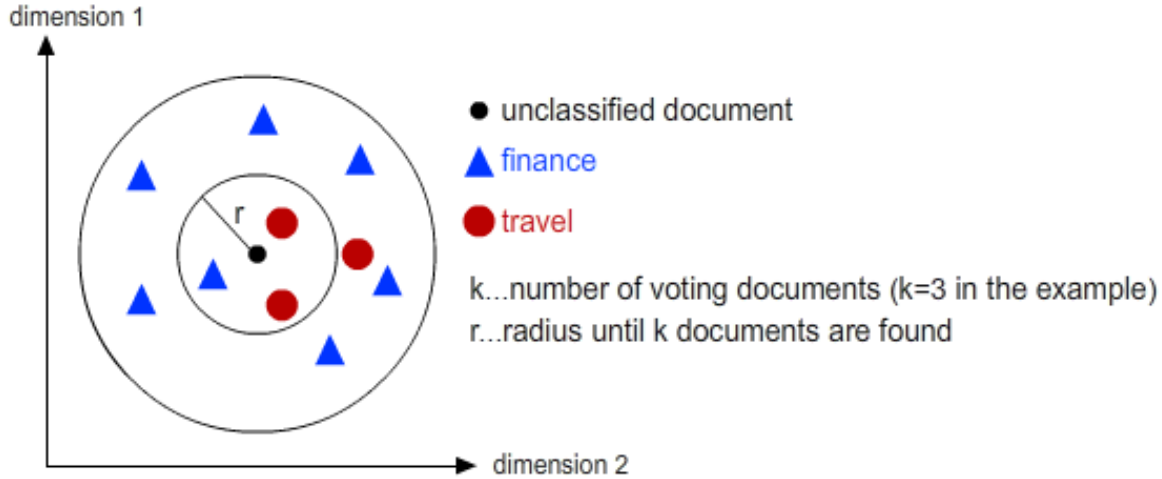Figure 5.1: Types of classification[36].

Figure 5.2: A simple KNN example.

number of available n-grams. Each training document is therefore a vector in that highly dimensional space. A new, unclassified document is now put into that vector space and by using distance function the k nearest neighbors are found for that document. Each of these neighbors votes with its own class, the more votes for one class the more likely that the new document belongs to that class.

Figure 5.2 shows a simple example of how the KNN classifier works. We limited the dimension to two for easier understanding. In this scenario we want to classify the given document (black dot in the middle) into one of the two categories "finance" (blue triangles) or "travel" (red dots). We calculate the distance between the new document and all training documents and consider the votes of the nearest three. In the example, two of these three document vote for "travel" which would let us classify our input document into that class.

The distance between two documents is calculated as shown in Equation 5.1 where $d1$ and $d2$ are the two documents. The shorter the distance, the more similar the documents.

$$distance(d1, d2) = \frac{1}{numberOfMatchingNGrams} \tag{5.1}$$

Dictionary-Based Classifier   The dictionary-based classifier[1] learns how probable each n-gram is for each given category and assigns the most probable category (or categories) to the input document.

A dictionary is built at training stage by counting and normalizing the co-occurrences of one n-gram and a category. The dictionary might then look as shown in Table 5.1 where each column is a category (finance, travel, and science) and each row is an n-gram. In each cell we now have the learned relevance for each n-gram and category $relevance(ngram, category)$. The sum of the relevances in each row must add up to one.

Table 5.1 shows an example dictionary matrix. The n-gram "money" is more likely to get the category "finance" ($relevance(money, finance) = 0.6$) than "science" ($relevance(money, science) = 0.25$) while the n-gram "beach" is most likely to appear in the category "travel" ($relevance(beach, travel) = 0.85$).

To classify a new document, we again create all n-grams, look up the relevance scores in the dictionary and assign the categories with the highest probability. The probability for each category and given document is calculated as shown in Equation 5.2 where $N_{Document}$ is the set of n-grams for the given document.

---

[1]This classifier won the first Research Garden (`http://www.research-garden.de`) competition where the goal was to classify product descriptions into 8 different categories. See press release at `http://www.research-garden.de/c/document_library/get_file?uuid=e60fa8da-4f76-4e64-a692-f74d5ffcf475&groupId=10137`

Table 5.1: N-Gram dictionary with relevances for categories.

n-gram	finance	travel	science
money	0.6	0.15	0.25
beach	0.1	0.85	0.05
paper	0.3	0.2	0.5

$$CategoryProbability(category, document) = \sum_{n\,\epsilon\,Nurl} relevance(n, category) \qquad (5.2)$$

The dictionary can be stored in memory, in an embedded H2 database, or in a client/server MySQL database. These settings can be made in the classification.conf file in the conf folder (see Section 3.1).

Evaluation

In order to find out which classifier works best with which feature settings, you can evaluate these combinations. The *tud.iir.classification.page.ClassifierManager* has the *learnBestClassifier* method to run the evaluation on all given classifiers with the given evaluation setting object *tud.iir.classification.page.Evaluation*. See Section 5.1.1 for more details of how to perform the evaluation programmatically.

The output of the evaluation will be three csv files that hold information about the combinations of classifier, dataset, training percentage, and the final performance for the combination. The performance is measured in precision, recall, and F1. The three files are stored in the data/temp folder and hold the following information.

averagePerformancesDatasetTrainingFolds.csv   This file holds the performance measures for each classifier, averaged over all given datasets, training percentages, and folds in the cross validation.

averagePerformancesTrainingFolds.csv   This file holds the performance measures for each classifier and dataset combination, averaged over all given training percentages and folds in the cross validation.

averagePerformancesFolds.csv   This file holds the performance measures for each classifier, dataset, and training percentage combination, averaged over all given folds in the cross validation.

Best Practices

This section describes how to prepare training and testing data to learn a model, evaluate, and use a text classifier.

Preparing the Training/Testing Data   The data can be specified in a simple text file. There are three classification options, namely, one-category classification, hierarchical classification and multi-category classification. They all require a similar structure of the data.

One-Category Classification   We write one URL and one category separated with a single space on each line. For example:

```
http://www.google.com search
http://www.fifa.com sport
http://www.oscars.com entertainment
```

Hierarchical Classification We write one URL and multiple categories separated with a single space on each line. The categories must be in the correct order, so the first category is the main one, all following are subcategories of each other. For example:

```
http://www.google.com search search_engine
http://www.fifa.com sport team_sports soccer
http://www.oscars.com entertainment movies awards usa
```

Multi-Category Classification We write one URL and multiple categories separated with a single space on each line. The order of the categories (tags) does not matter. For example:

```
http://www.google.com search image_search video_search
http://www.fifa.com soccer sport free_time fun ball_game results to_read
http://www.oscars.com entertainment movies films awards watch video stars
```

Training a classifier Before we can use a classifier, we need to learn a model. The model is an internal representation of the learned data. After learning a model, a classifier can applied to unseen data. We now have prepared the training and testing data so we can now learn the models. The classifier is saved as a lucene index or a database under the name of the classifier with a "Dictionary" suffix. The result of the learning are three files: the classifier ("CLASSIFIER.ser"), the dictionary object ("CLASSIFIERDictionary.ser"), and the actual dictionary as the index or database (e.g. "CLASSI-FIERDictionary.h2.db"). More settings can be configured in the config/classification.conf file. See 3.1.2 for more information.

Listing 5.1.1 shows an example for how to train and save a classifier. In this case we train a classifier that can classify the language of given documents. As training data we use a list of web pages of different languages from Wikipedia.

```
1  // create a classifier mananger object
2  ClassifierManager classifierManager = new ClassifierManager();
3
4  // specify the dataset that should be used as training data
5  Dataset dataset = new Dataset();
6
7  // set the path to the dataset
8  dataset.setPath("data/datasets/classification/language/languageDocumentIndex.txt");
9
10 // tell the preprocessor that the first field in the file is a link to the actual docu
11 dataset.setFirstFieldLink(true);
12
13 // create a text classifier by giving a name and a path where it should be saved to
14 TextClassifier classifier = new DictionaryClassifier("LanguageClassifier", "data/model
15
16 // specify the settings for the classification
17 ClassificationTypeSetting classificationTypeSetting = new ClassificationTypeSetting();
18
19 // we use only a single category per document
20 classificationTypeSetting.setClassificationType(ClassificationTypeSetting.SINGLE);
21
22 // we want the classifier to be serialized in the end
23 classificationTypeSetting.setSerializeClassifier(true);
24
25 // specify feature settings that should be used by the classifier
26 FeatureSetting featureSetting = new FeatureSetting();
27
28 // we want to create character-level n-grams
29 featureSetting.setTextFeatureType(FeatureSetting.CHAR_NGRAMS);
```

```
30
31 // the minimum length of our n−grams should be 3
32 featureSetting.setMinNGramLength(3);
33
34 // the maximum length of our n−grams should be 5
35 featureSetting.setMaxNGramLength(5);
36
37 // we assign the settings to our classifier
38 classifier.setClassificationTypeSetting(classificationTypeSetting);
39 classifier.setFeatureSetting(featureSetting);
40
41 // now we can train the classifier using the given dataset
42 classifierManager.trainClassifier(dataset, classifier);
```

Using a classifier   After we trained a model for a classifier we can apply it to unseen data. Let's use the model we just trained to classify the language of a new document.

Listing 5.1.1 shows how to use a trained classifier.

```
1 // the path to the classifier we want to use
2 String classifierPath = "data/models/languageClassifier/LanguageClassifier.ser";
3
4 // load the language classifier
5 TextClassifier classifier = ClassifierManager.load(classifierPath);
6
7 // create a classification document that holds the result
8 ClassificationDocument classifiedDocument = null;
9
10 // classify the little text (if classifier works it would say Spanish)
11 classifiedDocument = classifier.classify("Yo solo s que no s nada.");
12
13 // print the classified document
14 System.out.println(classifiedDocument);
```

You can also try the language classifier online at `http://www.webknox.com/wi#LanguageDetection`.

Evaluating a Classifier   To get an idea of how good a trained classifier works, we can evaluate it using test data which is structured the same way as the training data. Listing 5.1.1 shows how to evaluate a trained classifier, you will see that is very similar to training a classifier. Make sure that you evaluate the classifier using disjunct data, otherwise the evaluation results are invalid .

```
1 // create a classifier mananger object
2 ClassifierManager classifierManager = new ClassifierManager();
3
4 // the path to the classifier we want to use
5 String classifierPath = "data/models/languageClassifier/LanguageClassifier.ser";
6
7 // specify the dataset that should be used as testing data
8 Dataset dataset = new Dataset();
9
10 // set the path to the dataset (should NOT overlap with the training set)
11 dataset.setPath("data/datasets/classification/language/languageDocumentIndex.txt");
12
13 // tell the preprocessor that the first field in the file is a link to the actual docu
14 dataset.setFirstFieldLink(true);
```

```
15
16  // load the language classifier
17  TextClassifier classifier = ClassifierManager.load(classifierPath);
18
19  // now we can test the classifier using the given dataset (output is written to the co
20  ClassifierPerformance classifierPerformance = null;
21  classifierPerformance = classifierManager.testClassifier(dataset, classifier);
```

Testing parameter combinations As you have seen, you can train the classifier using different parameters. So how can you be sure that you set the parameters correctly? Do they work well on different datasets? Is the chosen classifier always better than others? In order to answer these questions with hard data you can automatically run different combinations of classifiers, settings, and datasets as shown in Figure 5.3. The green line shows the combination that was found to perform best. In the end you will get one evaluation csv with information about how the combination performed. You can then manually pick the best performing settings.
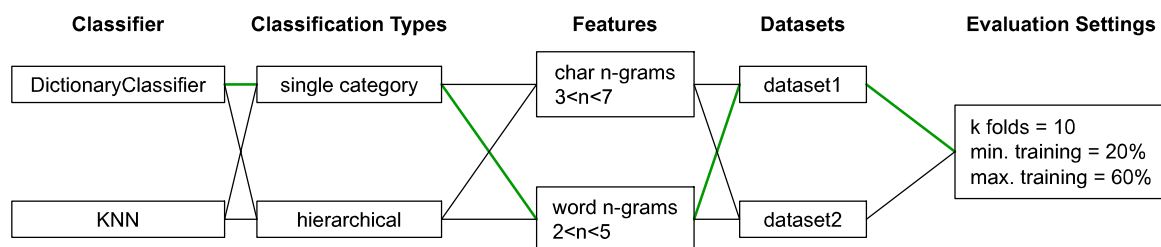
Figure 5.3: Combinations for training a solid classifier.

Listing 5.1.1 shows you how to do just that.

```
1   ClassifierManager classifierManager = new ClassifierManager();
2
3   // build a set of classification type settings to evaluate
4   List<ClassificationTypeSetting> classificationTypeSettings = new ArrayList<Classificati
5   ClassificationTypeSetting cts = new ClassificationTypeSetting();
6   cts.setClassificationType(ClassificationTypeSetting.SINGLE);
7   cts.setSerializeClassifier(false);
8   classificationTypeSettings.add(cts);
9
10  // build a set of classifiers to evaluate
11  List<TextClassifier> classifiers = new ArrayList<TextClassifier>();
12  TextClassifier classifier = null;
13  classifier = new DictionaryClassifier();
14  classifiers.add(classifier);
15  classifier = new KNNClassifier();
16  classifiers.add(classifier);
17
18  // build a set of feature settings for evaluation
19  List<FeatureSetting> featureSettings = new ArrayList<FeatureSetting>();
20  FeatureSetting fs = null;
21  fs = new FeatureSetting();
22  fs.setTextFeatureType(FeatureSetting.CHAR_NGRAMS);
23  fs.setMinNGramLength(4);
24  fs.setMaxNGramLength(7);
25  featureSettings.add(fs);
26
27  fs = new FeatureSetting();
```

```
28  fs.setTextFeatureType(FeatureSetting.CHAR_NGRAMS);
29  fs.setMinNGramLength(3);
30  fs.setMaxNGramLength(8);
31  featureSettings.add(fs);
32
33  fs = new FeatureSetting();
34  fs.setTextFeatureType(FeatureSetting.WORD_NGRAMS);
35  fs.setMinNGramLength(1);
36  fs.setMaxNGramLength(3);
37  featureSettings.add(fs);
38
39  // build a set of datasets that should be used for evaluation
40  Set<Dataset> datasets = new HashSet<Dataset>();
41  Dataset dataset = new Dataset();
42  dataset.setPath("dataset1.txt");
43  datasets.add(dataset);
44  dataset = new Dataset();
45  dataset.setPath("dataset2.txt");
46  dataset.setSeparationString("#");
47  datasets.add(dataset);
48
49  // set evaluation settings
50  EvaluationSetting evaluationSetting = new EvaluationSetting();
51  evaluationSetting.setTrainingPercentageMin(20);
52  evaluationSetting.setTrainingPercentageMax(80);
53  evaluationSetting.setkFolds(5);
54  evaluationSetting.addDataset(dataset);
55
56  // let's take the time
57  StopWatch stopWatch = new StopWatch();
58
59  // train and test all classifiers in all combinations
60  classifierManager.learnBestClassifier(classificationTypeSettings, classifiers, feature
61  evaluationSetting);
62
63  System.out.println("finished training and testing classifier combinations in " + stopW
```

5.2 Extraction

5.2.1 Keyword Extraction / Controlled Tagging

It is often interesting which words describe a given text most. These words are often called "keywords" or "tags". TOOLKIT is able to use a weighted, controlled vocabulary and perform tagging of a text using TF-IDF and tag correlations.

The ControlledTaggerIndex is the basis for the tagging. This index contains information about the controlled tag vocabulary, tag frequencies, a stem-map, and a correlation table for all tags. The tag index is built using training data of single text files that have been assigned with weighted tags. This index is serialized as a model file for later usage.

To tag a new text with the most relevant keywords, the ControlledTagger tokenizes the input text and stems the tokens using the SnowballStemmer. Tokens are now in their stemmed form, that is, tokens such as "blogging" and "blogs" became "blog".

All tokens that match a tag from the controlled vocabulary in the index are held in a bag-of-words which also stores the frequencies of the tokens in the input text. Using this data, we can calculate the TF-IDF values for each tag candidate. The TF-IDF scores are used to rank the tags initially.

Another ranking step is performed by using the tag correlation table from the index.

The ControlledTagger can be configured using the ControlledTaggerSettings class. You can configure the following behaviors:

1. TF-IDF threshold under which the tags should be discarded OR alternatively, the number of fixed tags per document.

2. The correlation re-ranking modus.

3. The stemmer to use

4. The stop words list

5. A list of regular expression that the tags need to match in order to be assigned

```
1 ControlledTagger tagger = new ControlledTagger();
2 System.out.println(tagger);
```

The keyword extraction is state-of-the art and beats the comparable system "Maui"[29] by 18.9% in the F1-Score. More information about the controlled tagging can be found in [20].

5.2.2 Web Page Content Extraction

Content oriented web pages such as blogs or news articles contain not only the text of interest but also clutter such as the navigation, footer, header, and ads. In order to automatically process the text without the clutter, we need to extract the text content. Figure 5.4 shows an example web page where the main article is in the green box. Everything else is just clutter and should not be extracted when looking for the unique article.

To perform this kind of extraction we could use wrapper approaches which are explained in more detail in [7]. However, these approaches require manual wrapper construction or semi-supervised learning which is too cumbersome. Another approach would be to detect the template of the web page and get only the contents of the main block (the green box). Template detection usually works by comparing one page with several other pages of the same domain to find the fix and changeable contents. This however requires several http requests and therefore more time and bandwidth. In TOOLKIT we use an approach that is based on a hierarchical analyzis of the web page's DOM tree. Each DOM element is analyzed and ranked based on the length of the contained text, the link density, and the frequency of other elements in relation to the text length. Longer text fragments usually indicate a relevant part of the article, while a high link density is more likely to indicate that the analyzed element is part of the navigation for example. Also the attributes "id" and "class" are analyzed. If the attribute values contain keywords such as "entry", "content", or "text", the element is more likely to contain relevant content as if the attribute values contain keywords such as "header", "footer", or "sidebar". The implementation of the web page content extraction adopted great parts of the Firefox extension "Readability"[2]. The used heuristics in Readability have shown to be quite accurate for a wide range of web pages.

5.2.3 Named Entity Recognition

Named Entity Recognition (NER) is the task of recognizing and disambiguating known and unknown entities in documents. In order to understand the task we need to understand what an *entity* is. An entity is a collection of rigidly designated chunks of text that refer to exactly one or multiple identical, real or abstract concept instances. These instances can have several aliases and one name can refer to different instance [48]. So for example "Iron Man 2" and "Iron-Man 2" are two different chunks of text which however refer to the same movie. Common types of entities are people (e.g. "Jim Carrey"), locations (e.g. "Los Angeles"), and organizations (e.g. "Google Inc.").

---

[2]`http://lab.arc90.com/experiments/readability/`

Figure 5.4: Article content of a web page marked with a green box [20].

A named entity recognizer is therefore a system or technique that tries to detect entities in a given natural language text. For example, an NER system that is used to detect people, locations, and organizations should be able to find the bold entities in the following text:

Bill Gates founded **M**icrosoft with **P**aul Allen in 1975 in **A**lbuquerque, New Mexico. The headquarter is located in **R**edmond, Washington and the company is led by CEO **S**teve Ballmer.

Taggers

The TOOLKIT wraps 7 Named Entity Recognizers using a single interface making it easier to test and substitute them in real world applications.

Alchemy   The Alchemy API is a web-based service that also offers Named Entity Recognition. Using this NER requires the application to have a developer key and Internet access. Alchemy covers a wide

range of concepts. For more details see `http://www.alchemyapi.com/api/entity/types.html` or the JavaDoc.

Illinois Learning-based Java   Students from the University of Illinois developed this Recognizer [16]. The recognizer comes with a large set of lists that help the recognizer to tag concepts such as corporations, countries, jobs, movies, people, songs, and more. More information can also be found on `http://l2r.cs.uiuc.edu/~cogcomp/asoftware.php?skey=FLBJNE`.

LingPipe   The LingPipe library [2] also offers an NER. Unfortunately the tagger comes with no models. Models for each desired concept need to be trained manually More information can be found on `http://alias-i.com/lingpipe/demos/tutorial/ne/read-me.html`.

OpenCalais   The Open Calais API [34] is a web-based service that also offers Named Entity Recognition. The service requires a developer key and access to the Internet. Open Calais covers many concept as covered on the documentation `http://www.opencalais.com/documentation/calais-web-service-api/api-metadata/entity-index-and-definitions` and in the JavaDoc.

OpenNLP   The OpenNLP toolkit [35] provides a maximum entropy based approach for named entity recognition. The toolkit comes with a small set of pre-trained models covering concepts such as locations, organizations, and persons. More details can be found in the JavaDoc.

Stanford   As part of the Stanford NLP library [40] [12] developed a named entity recognizer that is based on conditional random fields (CRF). The tagger was trained on the CoNLL 2003 data and comes with a small set of classifiers that can recognize persons, locations, and organizations. For more details see `http://www-nlp.stanford.edu/software/crf-faq.shtml` and the JavaDoc.

TUD   The TUD named entity recognizer was developed at the Dresden University of Technology. The recognizer is based on a list lookup and dictionary classification approach. So far no trained models exist for this tagger.

Tagging a Text

To use an NER you need two things: A text that you want to tag and a model for the tagger that has been trained to recognize the entities you want to find in the given text. The tagging works the same for all NERs and is shown in Listing 5.2.3.

```java
NamedEntityRecognizer ner = new TUDNER();
String inputText = "The iphone 4 is the successor of the iphone 3gs.";
String modelPath = "data/models/tudner/phone.model";
String taggedText = ner.tag(inputText, modelPath);
System.out.println(taggedText);
// desired output
// The <PHONE>iphone 4</PHONE> is the successor of the <PHONE>iphone 3gs</PHONE>.
```

Training a Tagger

Some of the available taggers come with trained models that can be used to detect entities. However, the concepts they were trained for might not be sufficient for the application you have in mind, therefore, you need to create your own training set with entities from your domain and learn a tagger using that data. This section explains which tagging formats are available and how models for the available taggers can be trained.

First we start with the tagging formats which can be used to create a training set.

Tagging Formats   Many tagging types have been developed which makes interoperability between different systems harder. TOOLKIT can handle many of these formats and is able to transform a tagged text within these formats.

The different tagging are explained in this section using the same example text: "Bill Gates founded Microsoft in 1975.".

Slash   The slash notation tags a given text with a slash and the tag right after the word. The sample text in the slash notation would look as show below.

```
Bill/PERSON Gates/PERSON founded Microsoft/ORGANIZATION in 1975/DATE.
```

Bracket   The bracket notation wraps the word in brackets. The sample tags would look as shown below.

```
[Bill Gates PERSON] founded [Microsoft ORGANIZATION] in [1975 DATE].
```

Column (BIO)   The column notation is one of the most widely used. For example the CoNLL NER dataset uses this format. The text is tokenized and on each line of the tagged text there is only one token (word) with its related tag separated by a space or tab. The BIO format means **B**eginning, cont**I**nue, and **O**utside. The BIO format can help to distinguish entities of the same type that are written close together.

The sample text in the column format would look as shown below. The "O" tag stands for "outside", that is no matching tag has been found.

```
Bill PERSON
Gates PERSON
founded O
Microsoft ORGANIZATION
in O
1975 DATE
. O
```

The sample text in the column BIO format would look as shown below.

```
Bill B-PERSON
Gates I-PERSON
founded O
Microsoft B-ORGANIZATION
in O
1975 B-DATE
. O
```

If we had another person name right after the first one, our tagger might think that it was only one person if we use the column notation. In the column BIO notation, both persons could be distinguished because the second one would start with B-PERSON.

XML   The XML notation is very common too and is the favored one in TOOLKIT. Similarly to the brackets notation, the XML notation wraps the word in an XML tag. The sample tags would look as shown below.

```
<PERSON>Bill Gates</PERSON> founded <ORGANIZATION>Microsoft</ORGANIZATION> in <DATE>1975</DATE>.
```

List The list notation is a separate file which contains references to the marked entities in the text. For each marked entity it contains its name, its start and stop index, and its tag. The sample list file for the sample text would look as shown below.

```
Bill Gates, 0-10, PERSON
Microsoft, 19-28, ORGANIZATION
1975, 32-36, DATE
```

The FileFormatParser can transform the formats back and forth as shown in Figure 5.5. As you can see, we can transform every format to the XML notation which will be used to create an annotation list for the tagged text.

Column BIO
word B-tag
word2 I-tag

Slash
word/tag

Column
word tag
word2 tag

Brackets
[word tag]

XML
<tag>word</tag>

List
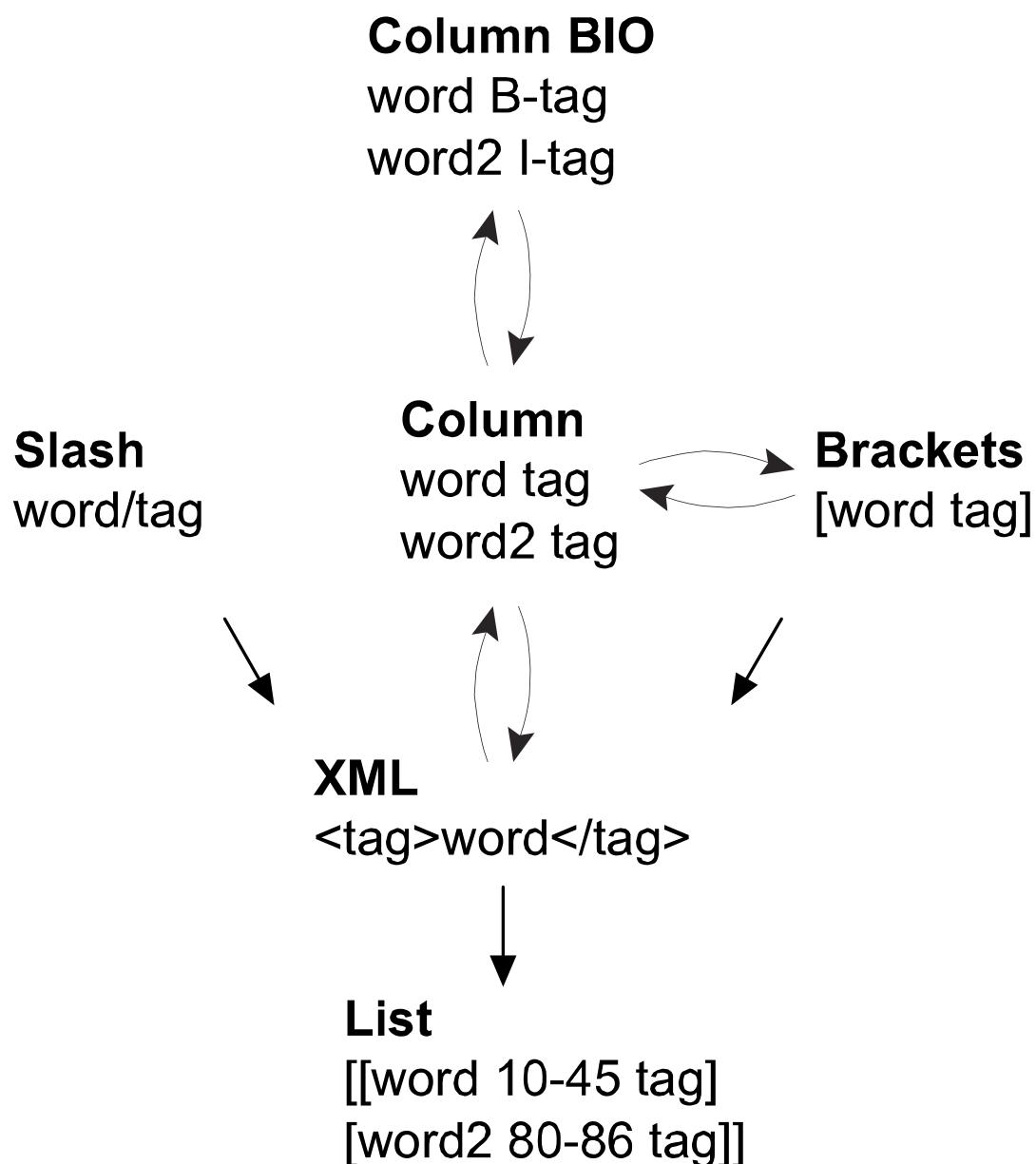[[word 10-45 tag]
[word2 80-86 tag]]

Figure 5.5: Possible transformation of tagging formats.

Using the training set   Once you have created your training set, you should use the FileFormat-Parser to transform it into a tab separated column tagging format (if it isn't in this format already).

It is not possible to train a model for the web-based taggers Alchemy and OpenCalais. All other taggers can be trained using the train method, that requires the training file and a model configuration parameter which is different for each tagger.

Let us assume we want to train a model for each trainable tagger to recognize mobile phone names. You should have tagged training data in column separated form as shown below. Usually, the more training data you have, the more stable the trained model. This training data is stored in a tsv file called "phoneTraining.tsv".

```
The O
iphone PHONE
4 PHONE
sells O
very O
well O
. O
```

Each tagger can be trained as shown in Listing 5.2.3. The only parameter which is used differently for each tagger is the "modelConfig". Table 5.2 shows what that parameter means for each trainable recognizer.

Table 5.2: Configuration parameters to train named entity recognizers.

NER	Parameter meaning
IllinoisLbjNER	Path to a configuration file, in this file you can specify which features and parameters should be used
LingPipeNER	Output path for the trained model
OpenNLPNER	Output path for the trained model, the filename must follow the format "openNLP_TAG.bin.gz" sin
StanfordNER	Path to a configuration file, in this file you can specify which features and parameters should be used
TUDNER	Output path for the trained model

```
1  NamedEntityRecognizer ner = new StanfordNER();
2  boolean successful = ner.train("phoneTraining.tsv", modelConfig);
3  String successfulOutput = "successful";
4  if (!successful) {successfulOutput = "not successful";}
5  System.out.println("Training of " + ner.getName() + " was " + successfulOutput);
```

Evaluating a Tagger

To see how good a trained tagger performs, we need test data which is tagged in the same manner as the training data. There are different versions of how to evaluate the performance. Let's go through them using an example. For example, let us assume that a human expert created the following markup [32]:

```
Unlike <PERSON>Robert</PERSON>, <PERSON>John Briggs Jr</PERSON> contacted <ORGANIZATION>Wonderful St
```

Furthermore, let us assume that a NER system created the following markup [32] for the same text:

```
<LOCATION>Unlike</LOCATION> Robert, <ORGANIZATION>John Briggs Jr</ORGANIZATION> contacted Wonderful
```

The only correct match between the correct solution and the NER system output is $<ORGANIZATION>$ $Acme</ORGANIZATION>$, all other markups are some kind of errors.

Error Types In classification tasks it is often possible to determine the true positives, false positives etc. but in NER it can sometimes help to be more precise about the classes. For example, two false positives are not necessarily equally wrong. Consider a system that had to tag person names in text, and it tagged "A good start" and "Jim Carrey was" as persons. While the first occurrence is obviously totally wrong, the second has to be considered wrong too but the system only did not find the right hand boundary correctly and tagged the word "was" too. In the previous example we can see five different errors an NER system can make [27]. The errors are shown and explained in the Table 5.3 [32].

Correct Solution	System Output
Unlike	<LOCATION>Unlike</LOCATION>
<PERSON>Robert</PERSON>	Robert
<PERSON>John Briggs Jr</PERSON>	<ORGANIZATION>John Briggs Jr</ORGANI
<ORGANIZATION>Wonderful Stockbrockers Inc</ORGANIZATION>	<ORGANIZATION>Stockbrockers</ORGANIZ
<LOCATION>New York</LOCATION>	<DATE>in New York</DATE>

Table 5.3: NER features [32].

Due to the variety of combinations to use the error types, three main evaluation methods have evolved during the years.

Exact-Match Evaluation The exact-match evaluation is the most simple one and does not take the different error types into account. A correct assignment must have the boundaries and the classification correct. The final score for the NER system is a micro-averaged f-measure (MAF). The NER system from the example would get the following scores.

- Precision = Correct / Total Assigned = 1 / 5 = 20%

- Recall = Correct / Total Possible = 1 / 5 = 20%

- MAF = 20%

MUC Evaluation The MUC evaluation method takes all five errors from Table 5.3 into account and scores a system along two axis, the TYPE and the TEXT axis. If an entity was classified correctly (regardless of the boundaries) the TYPE is assigned correct. If an entity was found with the correct boundaries (regardless of its type) the TEXT is assigned correct. For both axis, three measures are kept: the number of possible entities (POS), the number of actual assigned entities by the system (ACT) and the number of correct answers by the system (COR). MUC als uses the MAF as the final score for the NER system. As the usual f-measure, also the micro-averaged f-measure is the harmonic mean between precision and recall. For the example we can calculate the MUC score for the system as follows.

- COR = 4 (2 times TYPE correct, 2 times TEXT correct)

- ACT = 10 (5 times TYPE assigned, 5 times TEXT assigned)

- POS = 10 (5 times TYPE, 5 times TEXT)

- Precision = COR / ACT = 4 / 10 = 40%

- Recall = COR / POS = 4 / 10 = 40%

- MAF = 40%

Listing 5.2.3 shows how we can programmatically evaluate a tagger.

```
1  // initialize the tagger
2  NamedEntityRecognizer ner = new TUDNER();
3
4  // the path to the test data
5  String testFilePath = "testData.xml";
6
7  // the path to the model that we want to evaluate
8  String modelPath = "data/models/tudner/phone.model";
9
10 // store the evaluation results in an object
11 EvaluationResult eResult = null;
12
13 // evaluate the model using the test data
14 eResult = ner.evaluate(testFilePath, modelPath, TaggingFormat.XML);
15
16 // print out the evaluation result
17 System.out.println(eResult);
18
19 // get interesting values in exact match and MUC mode
20 System.out.println("F1 Exact Match: " + eResult.getF1(EvaluationResult.EXACT_MATCH));
21 System.out.println("F1 MUC: " + eResult.getF1(EvaluationResult.MUC));
```

5.2.4 Web Page Age Detection

The age of a web page can often be a useful indicator about the freshness of the information. It might also be used in classification or ranking of web documents. TOOLKIT is able to detect the age of many web pages by using four main techniques:

1. Reading the **H**TTP header of a web page and look for the date that the page has changed. Although this information is often absent of incorrect it sometimes is the only date we get.

2. Recognize a date in the **U**RL of the web page. Especially blogs use the URL style of "http://domain.tld/posts/YYYY or similar. In these cases the date in the domain is a very good indicator of the web page's age.

3. Recognizing and ranking dates in the **s**tructure and content of the web page. Especially news pages start or end their news posts with the location and the date that the event they are reporting about happened. Since many dates might be found in the content it is necessary to rank them among each other. This is done using indicators such as the nearness to certain keywords such as "published" for example.

4. Searching for **i**nbound links from archives. Some web pages can be found in archives with a date. If none of the other techniques returns valuable results it is possible to look up the URL in an archive, although the chances to find it are quite low.

Listing 5.2.4 shows how to detect the age of a web page. For more information on the algorithms used, see [13]. You can also check out the functionality of the web page age detection online at `http://www.webknox.com/wi#AgeDetection`.

```
1  // the URL of the page we want to know the age from
2  String url = "http://www.bbc.co.uk/news/world-europe-11432849";
3
4  // get the highest ranked date for this web page
5  ExtractedDate date = DateGetter.getBestDate(url);
6
7  // print the extracted date (should be 29.09.2010)
8  System.out.println(date);
```

5.2.5 FAQ Extractor

*F*AQs are frequently asked questions that are often found on web pages in a structured form. Basically there are three types how the FAQ can be structured:

1. **I**nternal Link In the beginning of the page, there is a list of questions that each link to the corresponding answer in on the same page.

2. **E**xternal Link In the beginning of the page, there is a list of questions that each link to the the web page that contains the answer.

3. **N**o Link Each question is immediately followed by its answer.

TOOLKIT can handle many FAQs that follow the first or third structure. First, the algorithm detects the XPaths to the questions and then it tries to extract all content between the current and the next question as the answer for the current question.

Listing 5.2.5 shows how the FAQExtractor can be used programatically. You can also check out the functionality of the FAQ Extractor online at `http://www.webknox.com/wi#FAQExtraction`.

```
1  // create a list of question answer pairs
2  ArrayList<QA> qas = null;
3
4  // the URL that contains the FAQ
5  String url = "http://blog.pandora.com/faq/";
6
7  // start extracting question and answers from the URL
8  qas = QAExtractor.getInstance().extractFAQ(url);
9
10 // print the extracted questions and answers
11 CollectionHelper.print(qas);
```

5.2.6 Fact Extraction

The FactExtractor can be used to detect facts in tables on web pages given a URL and optionally a small set of seed attribute names that help the extractor. The FactExtractor constructs all XPaths to the seed attributes and compares them. If it recognizes a similarity it concludes that the XPaths that are similar for the seed attributes might also point to other attributes on the page. In case of attributes that are listed in simple tables this approach works pretty well, if the attributes are presented in a more complicated way, the fact extractor might not find anything.

You can also check out the functionality of the FactExtractor online at `http://www.webknox.com/wi#FactExtraction`.

```
1  // the URL of the facts
2  String url = "http://en.wikipedia.org/wiki/Nokia_N95";
3
4  // the concept of the attributes
5  Concept concept = new Concept("Mobile_Phone");
6
7  // a small list of seed attributes
8  HashSet<Attribute> seedAttributes = new HashSet<Attribute>();
9  int attributeType = Attribute.VALUE_STRING;
10 seedAttributes.add(new Attribute("Second_camera", attributeType, concept));
11 seedAttributes.add(new Attribute("Memory_card", attributeType, concept));
12 seedAttributes.add(new Attribute("Form_factor", attributeType, concept));
13
14 // detect the facts using the seeds from the URL
```

```
15  ArrayList<Fact> detectedFacts = null;
16  detectedFacts = FactExtractor.extractFacts(url, seedAttributes);
17
18  // print the extracted facts
19  CollectionHelper.print(detectedFacts);
```

5.3 Retrieval

5.3.1 Source Retriever

The source retriever is a module that can query a number of sources such as search engines and web pages with terms and retrieve matching results.

Basic Features

Basic features are:

- Query the Google search engine (unlimited queries, top 64 results only).

- Query Yahoo search engine (5000 queries per IP and day, top 1000 results).

- Query Bing search engine (unlimited)

- Query Hakia search engine.

- Query Twitter.

- Query Google Blog search.

- Query Textrunner web page.

Some of the search APIs require API keys which must be specified in the config/apikeys.conf file. See 3.1.1 for more information.

How To

The following code snippet shows how to initialize the source retriever and get a list of (English) URLs from the Bing search engine for the exact search "Jim Carrey".

```
1   // create source retriever object
2   SourceRetriever s = new SourceRetriever();
3
4   // set maximum number of expected results
5   s.setResultCount(10);
6
7   // set search result language to english
8   s.setLanguage(SourceRetrieve.LANGUAGE_ENGLISH);
9
10  // set the query source to the Bing search engine
11  s.setSource(SourceRetrieverManager.BING);
12
13  // search for "Jim Carrey" in exact match mode (second parameter = true)
14  ArrayList<String> resultURLs = s.getURLs("Jim_Carrey", true);
15
16  // print the results
17  CollectionHelper.print(resultURLs);
```

5.3.2 Web Crawler

The web crawler can be used to crawl domains or just retrieve the cleansed HTML document of a single web page.

Basic Features

Basic functionalities include:

- Download and save contents of a web page.

- Automatically crawl in- and/or outbound links from web pages.

- Use URL rules for the crawling process.

- Extract title, description, keywords and body content of a web page.

- Remove HTML, SCRIPT and CSS tags.

- Find a sibling page of a given URL.

- Switch proxies after a certain number of requests to avoid being blocked.

How To

The following code shows how to instantiate a simple crawler that starts at http://www.dmoz.org and follows all in- and outbound links. The URL of each crawled page is printed to the screen. The crawler will use 10 threads, changes the proxy after every third request and stops after having crawled 1000 pages. Instead of setting the parameters using the code, we can also specify them in the config/crawler.conf file. See 3.1.2 for more information.

```java
// create the crawler object
Crawler c = new Crawler();

// create a callback that is triggered for every crawled page
CrawlerCallback crawlerCallback = new CrawlerCallback() {
        @Override
        public void crawlerCallback(Document document) {
                // TODO do something with the page
                System.out.println(document.getDocumentURI());
        }
};
c.setCrawlerCallback(crawlerCallback);

// stop after 1000 pages have been crawled (default is unlimited)
c.setStopCount(1000);

// set the maximum number of threads to 10
c.setMaxThreads(10);

// the crawler should automatically use different proxies
// after every 3rd request (default is no proxy switching)
c.setSwitchProxyRequests(3);

// set a list of proxies to choose from
List<String> proxyList = new ArrayList<String>();
proxyList.add("83.244.106.73:8080");
proxyList.add("83.244.106.73:80");
proxyList.add("67.159.31.22:8080");
```

```
29  c.setProxyList(proxyList);
30
31  // start the crawling process from a certain page,
32  // true = follow links within the start domain
33  // true = follow outgoing links
34  c.startCrawl("http://www.dmoz.org/", true, true);
```

5.4   Preprocessing

5.4.1   Tokenization

A *Token* is a sequence of characters that can be categorized according to the tokenization rules.
Tokenization is the process of transforming a text into a sequence of tokens. Table 5.4 shows example
token types. The text "Today I lost $1000 dollar playing poker." could consists of 8 tokens for
example. What token types exist depend on the application. The dollar sign could be a single token
for example. TOOLKIT uses regular expressions to perform the tokenization.

Table 5.4: Example types of tokens.

Character sequence	Token type
many	word
$1000	amount of money
26.09.2010	date
.	punctuation

One can also see each sentence of a text as a token, in this case the tokenization process is called
sentence splitting as explained int he next section.

5.4.2   Sentence Splitting

TOOLKIT has a rudimentary implementation for the common need for sentence splitting. TOOLKIT's
implementation works with hand-crafted rules and thus does not require a model. Sentences try to
be splitted on periods, question marks, and exclamation marks but there are also rules that try to
prevent splitting sentences at ellipses. For example, the following is online one sentence although it
contains several periods: "Sometimes sentenes contain many periods...really!".

The following code shows how the sentence splitting can be used:

```
1  String inputText = "This is a sentence. This is another one!";
2  List<String> sentences = Tokenizer.getSentences(inputText);
3  CollectionHelper.print(sentences);
4  // prints:
5  // This is a sentence
6  // This is another one!
```

You can also get a specific sentence by providing a phrase that is part of the sentence using the
getSentence method.

5.4.3   Creating N-Grams

N-grams are sets of tokens of the length n. We can distinguish two main types of n-grams:

1. **Character level n-grams** use each character of the string as a token. For example, from the
   string "It is sunny" we can create the following set of 3-grams: $it, ti, is, is, ss, su, sun, unn, nny$.
   The number of n-grams in a set can be calculated as $ngrams = numberOfTokens - n + 1$. In
   our example that means $9 = 11 - 3 + 1$.

2. **Word level n-grams** use each word (separated with white space) of the string as a token. For example, from the string "It is so nice and sunny today" we can create the following set of 3-grams: *Itisso, issonice, soniceand, niceandsunny, andsunnytoday*. The number of n-grams in a set can be calculated as $ngrams = numberOfTokens - n + 1$. In our example that means $5 = 7 - 3 + 1$. If you want to have single words as features for the text classifier you can simply use unigrams or bigrams which are n-grams with $n = 1$ or $n = 2$ respectively.

The document preprocessor allows you to create a set of n-grams with different length too. For example, you can create all 2-grams, 3-grams, and 4-grams for the given input text.

Listing 5.4.3 shows how you can create n-grams from a given text.

```
1  // the input text that we want to separate into n-grams
2  String inputText = "a cat runs funnily";
3
4  // store a set of n-grams
5  Set<String> ngrams = null;
6
7  // calculate all character n-grams of length 3
8  ngrams = Tokenizer.calculateCharNGrams(inputText,3);
9
10 // calculate all word n-grams of length 3
11 ngrams = Tokenizer.calculateWordNGrams(inputText,3);
12
13 // calculate all character level n-grams of length 3 to 5
14 ngrams = Tokenizer.calculateAllCharNGrams(inputText,3,5);
15
16 // calculate all character word n-grams of length 1 to 3
17 ngrams = Tokenizer.calculateAllWordNGrams(inputText,1,3);
```

5.4.4 Noun Pluralization and Singularization

TOOLKIT is able to transform most English singular nouns to their plural and back. For example, "city" becomes "cities" and "index" becomes "indices".

Listing 5.4.4 shows the simple usage of the singularization and pluralization using the WordTransformer class.

```
1  String singular = "city";
2  String plural = "";
3  plural = WordTransformer.wordToPlural(singular);
4  singular = WordTransformer.wordToSingular(plural);
5  System.out.println(singular);
6  System.out.println(plural);
7  // prints:
8  // cities
9  // city
```

5.5 Miscellaneous

The toolkit contains many helper functionalities for reoccurring tasks in the tud.iir.helper package. The following code snippet shows several sample usages of some of the functions.

```
1  // sort a map by its value in ascending order (2nd parameter = true)
2  Map m = CollectionHelper.sortByValue(map, true);
```

```java
3
4  // reverse a list
5  List l = CollectionHelper.reverse(list);
6
7  // print the contents of a collection
8  CollectionHelper.print(collection);
9
10 // get the runtime of an algorithm and print it (2nd parameter = true)
11 long startTime = System.currentTimeMillis();
12 for (int i = 0; i < 10000; i++) {
13         int c = i * 2;
14 }
15 DateHelper.getRuntime(t1, true);
16
17 // (de) serialization of objects
18 FileHelper.serialize(obj, "obj.ser");
19 Object obj = FileHelper.deserialize("obj.ser");
20
21 // rename, copy, move and delete files
22 FileHelper.rename(new File("a.txt"), "b.txt");
23 FileHelper.copyFile("src.txt", "dest.txt");
24 FileHelper.move(new File("src.txt"), "dest.txt");
25 FileHelper.delete("src.txt");
26
27 // get files from a folder
28 File[] files = FileHelper.getFiles("folder");
29
30 // zip and unzip a text
31 FileHelper.zip("text", "zipFile.zip");
32 String t = FileHelper.unzipFileToString("zipFile.zip");
33
34 // perform some action on every line of an ASCII file
35 final Object[] obj = new Object[1];
36 obj[0] = 1;
37
38 LineAction la = new LineAction(obj) {
39
40     @Override
41     public void performAction(String line, int lineNumber) {
42         System.out.println(lineNumber + ":_" + line + "_" + obj[0]);
43     }
44 }
45 FileHelper.performActionOnEveryLine(filePath, la);
46
47 // round a number with a number of digits
48 double r = MathHelper.round(2.3333, 2);
49
50 // remove HTML tags
51 String r = HTMLHelper.removeHTMLTags("<a>abc</a>",
52                                      true, true
53                                      true, true);
54
55 // trim a string
56 String t = StringHelper.trim("__to_trim+++");
57
58 // reverse a string
59 String r = StringHelper.reverse("abc");
```

```
60
61 // encode and decode base64
62 String e = StringHelper.encodeBase64("abc");
63 String d = StringHelper.decodeBase64(e);
```

Chapter 6

Where to go from here?

6.1 Referenced Libraries

The TOOLKIT makes excessive use of third party libraries. We do not intend to re-implement code but rather to built on it and create something superior. Here an incomplete list of libraries the toolkit uses:

- Apache Commons [3] for many standard tasks in string and number manipulation and more.
- Fathom [11] to measure readability of English text.
- iText [17] for creating PDF documents.
- Jena [18] for reading and writing ontology files.
- jYaml [19] to read and write YAML files.
- Log4j [24] for logging.
- Lucene [26] for indexing and making learned models persistent.
- NekoHTML [33] to clean up the HTML of web pages in order to process them correctly.
- SimMetrics [39] to calculate similarities of strings.
- Twitter4j [45] to query the Twitter API.
- Weka [15] for machine learning.

6.2 History

The foundation of the toolkit code came out of the WebKnox project[47] that was started in 2008. The code is in development by students of the Dresden University of Technology. Contributors are:

- Christopher Friedrich
- Martin Gregor
- Philipp Katz
- Klemens Muthmann
- Sandro Reichert
- David Urbansky
- Robert Willner
- Martin Werner

Bibliography

[1] `http://www.alchemyapi.com/`, accessed July 11, 2010.

[2] `http://alias-i.com/lingpipe`, accessed July 9, 2010.

[3] `http://commons.apache.org/`.

[4] J. Atserias, B. Casas, E. Comelles, M. González, L. Padró, and M. Padró. FreeLing 1.3: Syntactic and semantic services in an open-source NLP library. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC06)*, pages 48–55, 2006. `http://www.lsi.upc.edu/~nlp/papers/atserias06.pdf`.

[5] `http://balie.sourceforge.net/`.

[6] T. Briscoe, J. Carroll, and R. Watson. The second release of the RASP system. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 77–80. Association for Computational Linguistics, 2006.

[7] C.-H. Chang, M. Kayed, M. Girgis, and K. Shaalan. A Survey of Web Information Extraction Systems. 2006.

[8] W. Cohen. MinorThird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data. 1, 2004. `cohen2004minorthird`, accessed July 11, 2010.

[9] `http://contentanalyst.com/html/tech/technologies.html`, accessed July 11, 2010.

[10] D. Cunningham, D. Maynard, D. Bontcheva, and M. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. 2002.

[11] `http://www.representqueens.com/fathom/`.

[12] J. R. Finkel, T. Grenager, and C. Manning. Incorporating Non-local Information into Information Extraction Systems. In *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pages 363–370, 2005. `http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf`.

[13] M. Gregor. Altersbestimmung von Webseiten. Master's thesis, Dresden University of Technology, 2010.

[14] B. Guzel, 12 2009. `http://net.tutsplus.com/tutorials/html-css-techniques/top-15-best-practices-for-writing-super-readable-code/`.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. `http://www.cs.waikato.ac.nz/~ml/index.html`.

[16] `http://l2r.cs.uiuc.edu/~cogcomp/software.php`.

[17] `http://itextpdf.com/`.

[18] `http://jena.sourceforge.net/`.

[19] `http://jyaml.sourceforge.net`.

[20] P. Katz. NewsSeecr Clustering und Ranking von Nachrichten zu Named Entities aus Newsfeeds. Master's thesis, Dresden University of Technology, 2010.

[21] `http://www.languagecomputer.com/technology/`, accessed July 11, 2010.

[22] `http://carrotsearch.com/lingo3g-overview.html`, accessed July 11, 2010.

[23] H. Liu. MontyLingua: An end-to-end natural language processor with common sense, 2004. `http://web.media.mit.edu/~hugo/montylingua/`, accessed July 11, 2010.

[24] `http://logging.apache.org/log4j/`.

[25] E. Loper and S. Bird. NLTK: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics*, volume 1, page 70. Association for Computational Linguistics, 2002. `http://www.nltk.org/`, accessed July 11, 2010.

[26] `http://lucene.apache.org`.

[27] C. Manning. Doing named entity recognition? don't optimize for f1. Website, Blog, August 2006.

[28] A. McCallum. Mallet: A MAchine Learning for LanguagE Toolkit, 2002. `http://mallet.cs.umass.edu/index.php`, accessed July 11, 2010.

[29] O. Medelyan, E. Frank, and I. H. Witten. Human-competitive tagging using automatic keyphrase extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1318–1327. Association for Computational Linguistics, 2009.

[30] `http://morphadorner.northwestern.edu/`.

[31] `http://www.nactem.ac.uk/software.php`, accessed July 11, 2010.

[32] D. Nadeau. *Supervised Named Entity Recognition: Learning to Recognize 100 Entity Types with little Supervision*. PhD thesis, Ottawa-Carleton Institute for Computer Science, 2007.

[33] `http://nekohtml.sourceforge.net/`.

[34] `http://www.opencalais.com/`, accessed July 11, 2010.

[35] `http://opennlp.sourceforge.net/about.html`, accessed September 23, 2010.

[36] X. Qi and B. D. Davison. Web page classification: Features and algorithms. 2009.

[37] `http://www.basistech.com/products/`, accessed July 11, 2010.

[38] M. Settings. Apache Mahout-scalable machine learning algorithm. *health*, 2:67.

[39] `http://www.dcs.shef.ac.uk/~sam/simmetrics.html`.

[40] `http://nlp.stanford.edu/software/index.shtml`.

[41] J. Stefanowski and D. Weiss. Carrot 2 and language properties in web search results clustering. *Advances in Web Intelligence*, pages 955–955, 2003. `http://project.carrot2.org/`, accessed July 11, 2010.

[42] A. Stolcke. SRILM-an extensible language modeling toolkit. In *Seventh International Conference on Spoken Language Processing*, volume 3, pages 901–904. Citeseer, 2002. `http://www.speech.sri.com/projects/srilm/`, accessed July 11, 2010.

[43] `http://www.megaputer.com/textanalyst.php`, accessed July 11, 2010.

[44] K. Tomanek, E. Buyko, and U. Hahn. An uima-based tool suite for semantic text processing. In *UIMA Workshop at the GLDV*, volume 11, 2007.

[45] `http://twitter4j.org/en/index.htm`.

[46] `http://www.textanalysis.com/Products/VisualText/visualtext.html`, accessed July 11, 2010.

[47] `http://www.webknox.com`.

[48] 9 2010. `http://www.webknox.com/blog/2010/09/named-entity-definition/`.

[49] X. Zhou, X. Zhang, and X. Hu. Dragon Toolkit: Incorporating auto-learned semantic knowledge into large-scale text retrieval and mining. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. Citeseer, 2007.