Contents

Chapter 1

Introduction

Nowadays thanks to unlimited Internet access we can gather enormously huge amount of data easily within rapid pace. The main problem which occurs is to simplify the access to the content which is expected from user. People upload a lot of information not necessarily mark with appropriate sign to find it again fast. Such problem lead to develop a system which would facilitate data organization. Especially categorization is expected in Geographic Information System (GIS) e.g. Web-based application enabling map visualization or Portable Navigation Device (PND).

1.1 Problem and Thesis Scope

This thesis is describing complex problem of POI categorization and its possible solutions. Computer Information System which can facilitate to solve given problem was implemented as a practical part. Document contains also description of methods which can be applied to categorize data (which in this case is plain text).

1.2 Thesis Objectives

As main objective was to prepare ready to use java library which would enable user flexible API (Application Programmer Interface). In this context word ’flexible’ means that user can provide data in unstructured form and obtain result as a category of given input. Such library can be imported as plain jar file without any additional sources as it is compiled with all dependencies. The only one required source is input data which should be provided by user as a working material.

1.3 Research Method

As mentioned before a categorization is complex problem which has to be customized according to given data. There are a lot of text processing methods to apply. Some of them were considered to be used e.g. preprocessing, feature extraction, train classifier, Support Vector Machine, Vector Space Model.

1.4 Document Organization

This engineering thesis consists of five chapters. First chapter is an introduction and briefly describes content of thesis. In chapter two are described methods useful during POI categorization process which are theoretical part of this thesis and one of its purposes. Next three chapters describes practical part of thesis. Chapter three focuses on *Loadstone* database which is data source for POI categorization. Chapter four describes in detailed way information system architecture and implementation. The last chapter is a summary containing categorization results and thesis conclusion.

Chapter 2

Categorization Methods

2.1 Overview

Plain text is the most friendly and convenient in use for human beings as a data medium. Unfortunately amount of text which can be effectively analysed by human is quite limited. On the other hand, text as source of information for computer machines does not possess any heuristics. That is the reason to develop methods which can be applied to for particular data sources selected by human. As text content is usually complex methods are designed in such way that cover narrow categories of data sources. In general methods taken under consideration in this thesis were covering following approaches:

- Preprocessing - text analysis paying special attention to reject unnecessary phrases which would slow down categorization process [1].

- Feature Extraction - text analysis which can consider wide range of text. Useful especially to analyse human text input (Internet forum comment and entries). Uses Term Frequency-Inverse Document Frequency method (TF-IDF). The result is heuristic for further text analysis [2].

- Naive Bayesian Classifier - method enabling direct text classification to appropriate category/phrase. It requires initial knowledge (heuristics) about given input [3].

- Support Vector Machine (SVM) - a form of binary classification. It divides input to two different separable domains. In order to classify for particular domain one-to-one comparison is applied [4][5].

- Vector Space Model (VSM) - method of text indexing and weight measuring. Already weighted indices are easier to give a hint for human which document to query. In the last stage measures text content similarity according to given query - *Centroid Classifier* [6].

2.2 Preprocessing

The most important function of preprocessing is obviously downsizing of input data. Redundant or non-heuristic data can influence factors like:

- Performance - the less text to analyse the result can be obtained faster.
- Complexity - unnecessary phrases can influence in further steps on classification correctness (dim heuristics).

The most often rejected phrases are those which appear the most frequently in particular language. Concerning English language phrases like: *the, a, e.g, i.e* does not introduce any hint for further classification. This is the reason why processing of such statements can be abandoned in further steps. Each document should be human analysed properly before applying patterns to reject. Patterns will differ significantly if given input will be database entry, HTML page or plain forum comment.

2.3 Feature Extraction

Feature Extraction method as its name states allow user to extract particular parts to avoid further repetitions and introduce more uniform steps enabling human easier interpretation. Features are basically phrases which can pretend later to corresponding classes (POI categories). Phrases are dividing to two categories:

- single word
- multi-word

To compute frequency of single words TF-IDF [7][2] can be used. This method will be described later in this thesis. In case of **single word** task is not complicated because of its "non-complex nature". It is easy to detach words from each other for computer machine.

Multi-word is much more complicated construction which can consists of two or more consecutive words. Although it is difficult to clearly define how multi-world looks like, it can change meaning when atomic parts (words) are changed in order or can be translated directly. What is more human avoid repetitions of the same word when create text but when using multi-words the latter are not introduced by their counterparts to provide explicitness. Such approach allows for better context clarity. This is the reason to deeply analyse multi-words. Multi-words can be retrieved using many different methods like: *semi-supervised document classification, inspection, frequency approach, sentence comparison, mutual information* [1].

2.3.1 TF-IDF

Term Frequency - Inverse Document Frequency is basically about computing frequency of word occurrences. When considering input data (document) it is presented as a vector of weights. Vector consists of phrases weights occurring in document. Result of TF-IDF is an information about phrase frequency occurrence but in balanced way - it includes word frequency in whole collection of documents. Such approach disables local increases of importance in total (if given phrase doesn't occur relatively uniformly its weight does not increase).
Term Frequency is denoted by following formula:

$$TF(term_i, doc_j) = \frac{term_{i,j}}{\sum_0^k term_{k,j}} \tag{2.1}$$

where:
$term_{i,j}$ is amount of phrase (term) occurrences in given document doc_j
$denominator$ is sum of all phrases occurrences in document doc_j

Inverse Document Frequency is denoted by following formula:

$$IDF(term_i, D) = \log \frac{D}{1 + |d \in D : term_i \in D|} \tag{2.2}$$

where:
D is amount of documents in collection
$denominator$ is sum of all documents where $term_i$ occurred
TF-IDF is a product of both factors:

$$TFIDF(term_i, doc_j, D) = TF(term_i, doc_j) \times IDF(term_i, D) \tag{2.3}$$

where:
$term_{i,j}$ is amount of phrase (term) occurrences in given document doc_j
D is collection of documents

2.3.2 Semi-supervised multi-word extraction

Depending on the topic of input data we can try to extract multi-words using human help. As the main topic are POIs we can apply **External Glossary of Terms** or **Bag of Words** (BOW) which can consist phrases such as: "Tourist Information", "Old Town" [1]. Output of multi-word extraction can be simply compared if it is included in defined BOW. If comparison is positive multi-word is taken for further processing. To extract multi-word from given input (document) doc_j we can apply rule:

Compare two sentences:

1. Analyse each word $word_i$ in sentence $sentence_j$ of document doc_k.
2. For every word $word_i$ check if is equal to word $word_l$ in sentence $sentence_m$ of document doc_k
3. While $word_i$ == $word_l$ increase counter $counter_c$ by one.
4. If $counter_c$ is greater then one we append $word_l$ to $word_i + counter_c$

Such steps will output multi-word for sentences: $sentence_j$ and $sentence_m$. To extract multi-words for document we have to for every sentence $sentence_j$ compare with $sentence_{j+1}$ while $j < k$ where k is a number of sentences in document doc_k.

2.4 Naïve Bayesian Classifier

This method is widely used in text classification. What is important when applying this method is size of input data. Significant factor is balance between represented classes (categories) and input data (each class need to have sufficient training data). What is more, method considers complete independence between features which is not always true - *'naive' word genesis e.g (data analysis describing relation between age and income - it is quite common that older people incomes are higher than younger)* [8].
When applying Naive Bayesian Classifier we have to define following factors:

- Collection of documents $D = d_0, ..., d_i$
- Set of words (features) $W = w_0, ..., w_j$ - extracted previously in Feature Extraction phase
- Set of classes (categories) $C = c_0, ..., c_w$
- $|V|$ - dimensional vector. Document $d_i = (w_1, w_2, ...w_{|VS|})$, $|VS|$ - vocabulary size in collection of documents D, where $w_i \in \{0, 1\}$ denotes respectively non-existence or existence of ith word in vocabulary set.

First step to obtain a class to which training data are belonging is posterior probability, $P(c_j|d_i)$. Basing on this probability and multinomial model we can conclude [3]:

$$P(c_j) = \frac{\sum_{i=1}^{|D|} P(c_j|d_i)}{|D|} \tag{2.4}$$

then applying Laplacian smoothing:

$$P(w_t|c_j) = \frac{1 + \sum_{i=1}^{|D|} N(w_t, d_i) P(c_j|d_i)}{|V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|D|} N(w_s, d_i) P(c_j|d_i)} \tag{2.5}$$

where $N(w_t, d_i)$ is the count of the number of times the word w_t occurs in document d_i and where $P(c_j|d_i) \in \{0, 1\}$ depends on the class label of the document. Finally, assuming that the probabilities of the words are independent given the class, we obtain:

$$P(c_j|d_i) = \frac{P(c_j) \prod_{k=1}^{|d_i|} P(w_{d_i,k}|c_j)}{\sum_{r=1}^{|C|} P(c_r) \prod_{k=1}^{|d_i|} P(w_{d_i,k}|c_r)} \tag{2.6}$$

For considered document d_j with highest value of $P(c_j|d_i)$ class c_j is assigned.

2.5 Support Vector Machine

This method is mainly used for well prepared and defined input data. As there are many sources of ready-to-use is widely used for POI classification. Methods presented previously are more indicated to data in unstructured representation. Assuming initial conditions [9]:

- POI p_i is represented as a n-dimensinal vector $x_i \in R^n$
- $y \in Y$ and is a category (class) of POI p_i
- Category is defined as: $y = f(x_i)$
- Data set: $D = (x_1, y_1), ..., (x_i, y_i)$
- For simplicity we assume $Y = \{-1, +1\}$

SVM for every (x_i, y_i) can be defined by finding optimal hyperplane $x \times w + b$ filling requirements of:

$$x_i \times w + b \geq +1, y_i = +1 \tag{2.7}$$

$$x_i \times w + b \leq -1, y_i = -1 \tag{2.8}$$

Parameters w, b are optimized using training data D. Obviously such formulas allow only for binary classification (distinguish between two categories). If resultant data are not linearly separable (category cannot be clearly defined) there can be applied additional method *kernel trick.* If separation is available and classification over more complex set is required *one-against-one* strategy is used [4]. To optimize SVM using *Soft Margin* will obtain:

$$\min_{w,\xi} ||w||\frac{1}{2} + C\sum_{i=1}^{N}\xi_i \tag{2.9}$$

subjected to:

$$y_i(x_i \times w + b) \geq 1 - \xi_i, \forall_i \xi_i \geq 0 \tag{2.10}$$

Variable C is a parameter which allow user to control training error, while ξ_i is slack variable. Soft Margin method define hyperplane which splits two sets as much as possible but in the same time leave maximum distance from data which were separated without any problems [4]. The same situation is noticeable when considering dual form (α is a vector of the Lagrange multiplier α_i):

$$\min_{\alpha} -\sum_{i=1}^{N}\alpha_i + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i x_j \tag{2.11}$$

subjected to:

$$\sum_{i=1}^{N}\alpha_i y_i = 0, \forall_{i,0} \leq \alpha_i \leq C \tag{2.12}$$

2.5.1 Kernel trick

The most important factor about SVM is its dependency only on the input data inner products [9]. When cannot clearly separate data it is possible to transform them to higher order space θ where separation is possible. It implies we apply $\theta(x_i) \times \theta(x_j)$ for data from equation (2.11). Fortunately $\theta(x_i) \times \theta(x_j)$ is equivalent to kernel $K(x_i, x_y)$. Such substitution allows to omit computing higher order space. Kernel can be chosen from many various kernel functions e.g. polynomial [5].

2.6 VSM and Centroid Classifier

In VSM we assume that each document d is represented as a term-space vector and each term is obtained using TF-IDF. Having prepared all terms in vector, centroid classifier can be computed for every class c_i [10].
First step is to calculate document vectors for every class c_i:

$$SUM_{c_i} = \sum_{d \in c_i} d \tag{2.13}$$

Second step is normalization of the sum SUM_{c_i} to obtain centroid C_i for class c_i:

$$C_i = \frac{SUM_{c_i}}{||SUM_{c_i}||_2} \tag{2.14}$$

Where denominator in equation (2.14) is second norm of vector $||SUM_{c_i}||$. Basing on vectors of classes C_i we can define to which class document d_i belongs. Classification is done by investigation how input vector data is similar to class vector.

2.7 Conclusion

There are many categorization methods which can be used for different purposes. Their results may also differ and mainly depend on the form of input data. In case of this thesis there were three methods which were applied for categorization purpose: **preprocessing, naive classification, semi-supervised categorization using extraction**. Preprocessing was mainly applied due to performance issues - superfluous phrase removal is useful during large data set analysis. Naive classification is suited for created NACE categories section: 4.4.3 and implementation is described in section: 4.5.7. It is not implementation of **Naïve Bayesian Classifier**. Implementation approach is basing on NACE categories descriptions method. It implicitly shows how results may be irrelevant when input data does not have much in common with category description. Naïve Bayesian Classifier was not applied because of source data amount. As mentioned in section 2.4 it should be used when training data size is large. In case of this thesis source data (chapter 3) were too small to obtain reasonable results (only one cell from database entry filled with string is used when classification is performed). Semi-supervised categorization was used as a customized method which could gave the most effective results during categorization process for given input data.

Chapter 3

Loadstone SQLite Database

3.1 Overview

Problem of POI categorization comes for many data sources. One of them is database which is used in legacy Symbian applications which are used by blind people. Such applications are difficult to migrate fluently to newer platforms used in high-tech smartphones e.g. iOS or Android. Such data are organized in difficult to analyse text files. Exemplary database files can be found on Loadstone project website[1]. The purpose of this thesis was to prepare ready to use source data for Java-based platform Android. Such approach allows programmer to easily invoke Java methods delivering source data in convenient way.

3.2 SQLite Database

SQLite is a RDBMS(Relational Data Base Management System) released on Public Domain license. SQLite avails using SQL queries and flexible APIs, inter alia Java. Is widely used in embedded systems, Internet browsers. SQLite is widely used as an embedded part of system so it was the most suitable choice for purpose of this thesis.

3.3 SQLite Praxis

To deliver easy access solution source text files was converted to SQLite database [11]. Exemplary database entries are presented in table 3.1 .

[1]http://gps.zlotowicz.pl/tekstowe/index.html

name	latitude	longitude
bankomat i oddział bz wbk atm 24h bank ul. warszawska 75a wieruszów	512950200	181606400
adres sieradzka 8 zduńska wola	516032151	189307481
sklep spożywczy łaszew rządowy	511379400	186639400

Table 3.1: Sample Loadstone database entries

SQLite databases give opportunity to compress data effectively and are widely used in smartphones where resources need to have limited size. To build SQLite database special bash shell script was developed:

```
#!/bin/sh
sqlite3 $2.db <<EOS
.headers on
CREATE TABLE LoadstoneTotalDataObjectModel (
"name" TEXT,
"latitude" REAL,
"longitude" REAL,
"accuracy" REAL,
"satellites" INTEGER,
"priority" INTEGER,
"userid" INTEGER,
"id" INTEGER
);
.separator ,
.import $1 TotalData
CREATE TABLE DatabaseInfo ( version INT, createdDate DATETIME, accessedDate
    DATETIME);
INSERT INTO DatabaseInfo(version, createdDate, accessedDate) VALUES (1,datetime
    ('now'),datetime('now'));
.tables
.schema TotalData
.schema DatabaseInfo
EOS
```

As first input parameter of the script is path to text file which contains data supposed to be converted to SQLite database. As second parameter takes resultant database name. Invoke example in bash shell:

```
chmod +x createDB.sh #makes script executable
createDB.sh /directory/to/txt/file/cala_polska.txt dataBaseNameWithoutExtention
```

To invoke script properly obviously sqlite3 has to be installed on processing machine.

3.3.1 Usage in Java Project

Resultant SQLite database obtained from bash shell script can be applied as a part of Java project. It is one file which size is limited to 140 TB (according to SQLite standard). File applied in project is 263 MB. To obtain easy to apply for human communication with database inside Java programming language additional Java SQLite Database Model library was imported into project [12]. This library presents fully ORM (Object Relational Mapping)-based approach which is very intuitive for Java programmers. Such approach relieve programmers from SQL (Structured Query Language) query knowledge. It delivers nice to read and use data methods e.g.

```java
database.getObjectModelColumns()[0].getName()
// returns name of first column in SQLite database
database.getObjectModel(LoadstoneTotalDataObjectModel.class).
    getAll();
// returns all rows in database as a list of objects of
    LoadstoneTotalDataObjectModel class.
```

Programmer has to know only basic SQL syntax to formulate appropriate query in order to filter for desired data.

3.3.2 Object Model

As a point of interest categorization system is mainly focused on structuring data in intuitive way and sample data are provided in already defined standard, object model had to be developed. Database applied in Java project is storing objects of LoadstoneTotalDataObjectModel class. This class is structuring data as: ***latitude, longitude, name*** and another's which are not important for purpose of this thesis. Additionally DataModel interface was created. It was applied for purpose of future development of the system. Such design allows to append additional data sources e.g. OpenStreetMap[2]. If data source implements DataModel interface it can be used by API which is target of this thesis.

[2] `https://www.openstreetmap.org/`

Chapter 4

Application Architecture

4.1 Overview

This chapter describes application architecture which was purpose of this thesis. Application is organized into **four** essential parts:

- Data source: loadstone DBMS
- Module 1: loadstone-model
- Module 2: loadstone-api
- Module 3: loadstone-package

*Note: naming convention can differ from standard UML definitions. Module can be considered as component - represents a modular part of a system, that encapsulates its content and whose manifestation is replaceable within its environment. A component defines its behaviour in terms of provided and required interfaces [13]. Module convention comes from the fact that application life-cycle and dependency management is marshalled by tool **Maven**. Maven uses naming convention of modules and helps developer to avoid introducing circular dependencies so application is easier to maintain in future* [14].

4.2 Architecture scheme

In the figure 4.1 is depicted overall architecture of Loadstone application.

4.3 Architecture UML Class diagram

To explore more implementation of application figure 4.2 should be analysed. It clearly describes technical details of application.

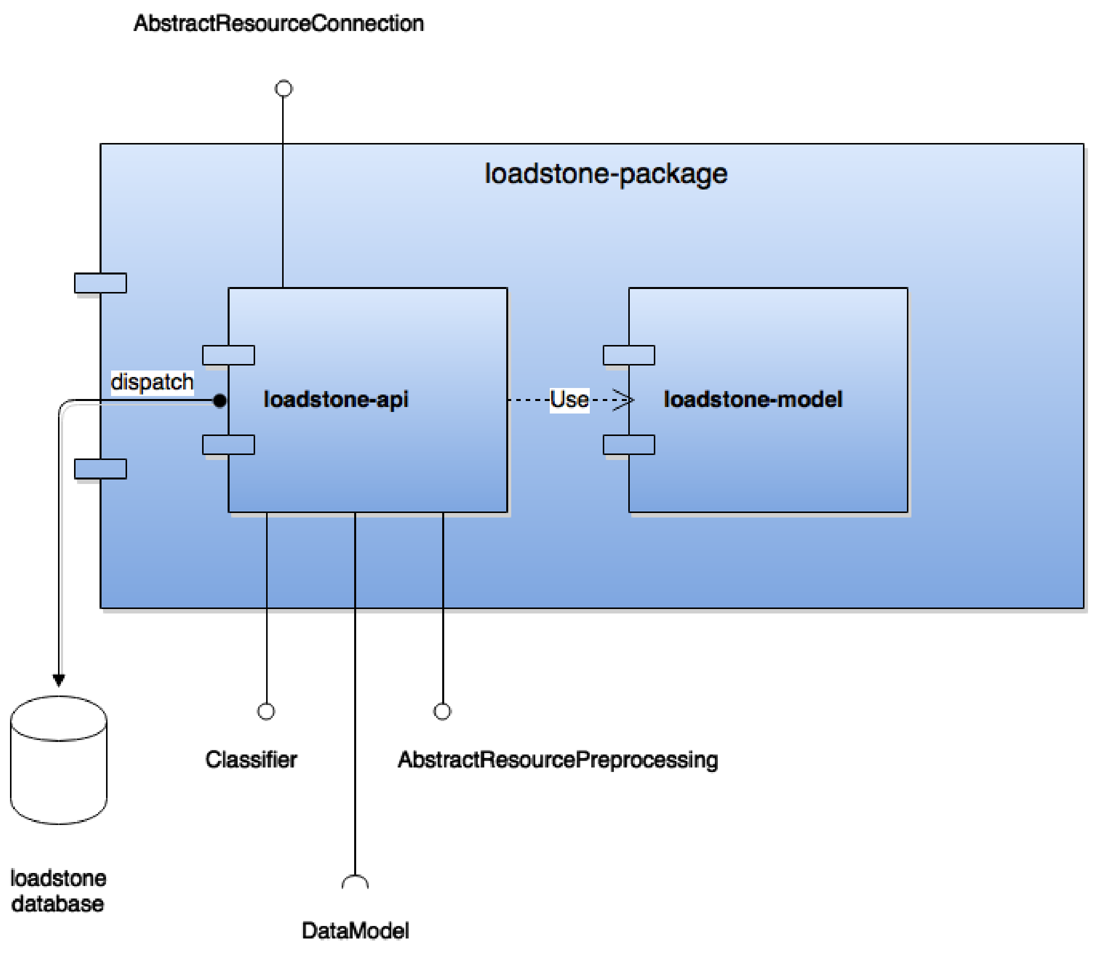


Figure 4.1: Loadstone application architecture

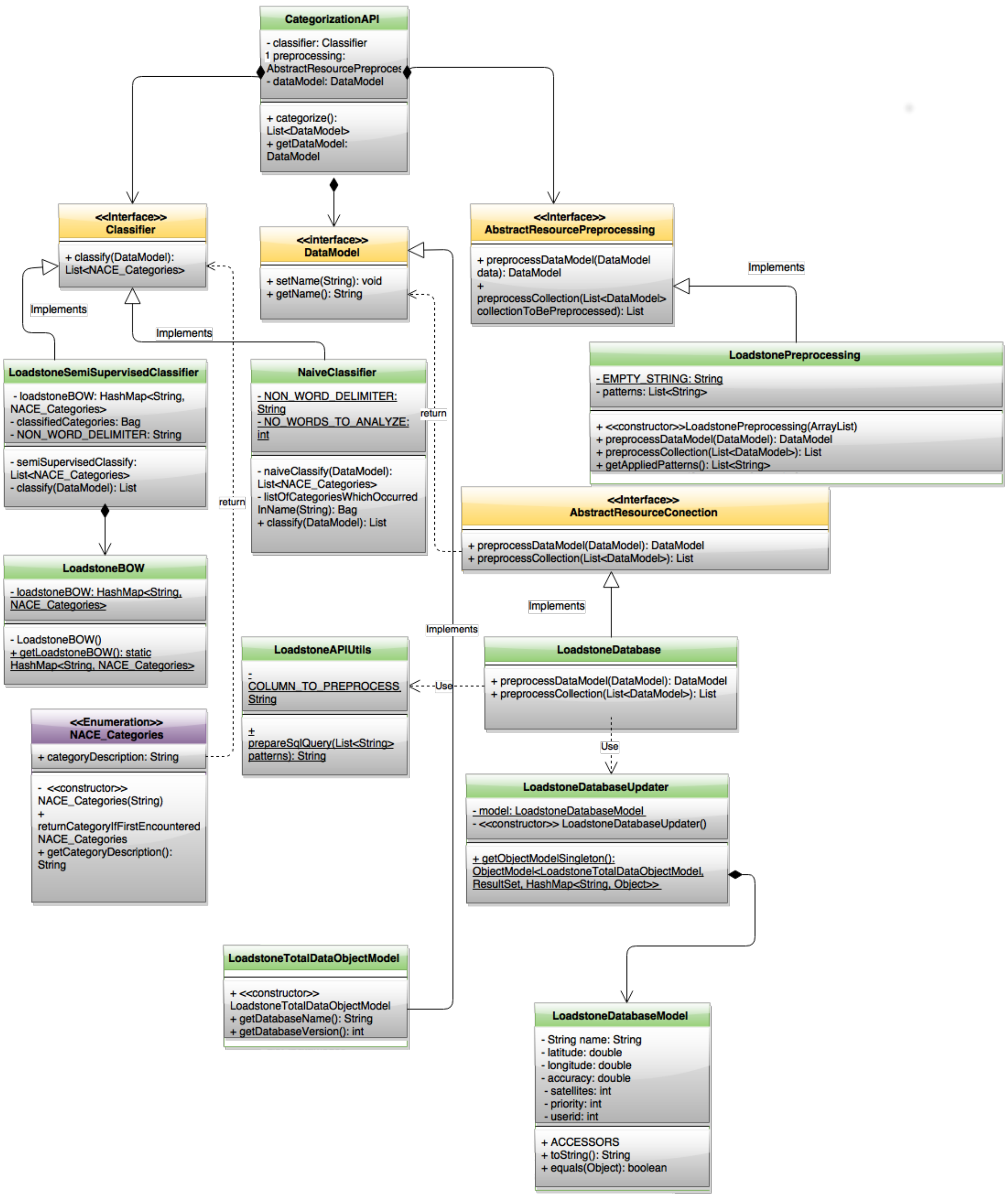

CategorizationAPI
- classifier: Classifier
- preprocessing: AbstractResourcePreproces
- dataModel: DataModel
+ categorize(): List<DataModel>
+ getDataModel: DataModel
<<Interface>> Classifier
+ classify(DataModel): List<NACE_Categories>
<<interface>> DataModel
+ setName(String): void
+ getName(): String
<<Interface>> AbstractResourcePreprocessing
+ preprocessDataModel(DataModel data): DataModel
+ preprocessCollection(List<DataModel> collectionToBePreprocessed): List
Implements
LoadstoneSemiSupervisedClassifier
- loadstoneBOW: HashMap<String, NACE_Categories>
- classifiedCategories: Bag
- NON_WORD_DELIMITER: String
- semiSupervisedClassify: List<NACE_Categories>
- classify(DataModel): List
NaiveClassifier
- NON_WORD_DELIMITER: String
- NO_WORDS_TO_ANALYZE: int
- naiveClassify(DataModel): List<NACE_Categories>
- listOfCategoriesWhichOccurred InName(String): Bag
+ classify(DataModel): List
return
LoadstonePreprocessing
- EMPTY_STRING: String
- patterns: List<String>
+ <<constructor>>LoadstonePreprocessing(ArrayList)
+ preprocessDataModel(DataModel): DataModel
+ preprocessCollection(List<DataModel>): List
+ getAppliedPatterns(): List<String>
<<Interface>> AbstractResourceConection
+ preprocessDataModel(DataModel): DataModel
+ preprocessCollection(List<DataModel>): List
LoadstoneBOW
- loadstoneBOW: HashMap<String, NACE_Categories>
- LoadstoneBOW()
+ getLoadstoneBOW(): static HashMap<String, NACE_Categories>
LoadstoneAPIUtils
- COLUMN_TO_PREPROCESS String
+ prepareSqlQuery(List<String> patterns): String
Use
LoadstoneDatabase
+ preprocessDataModel(DataModel): DataModel
+ preprocessCollection(List<DataModel>): List
<<Enumeration>> NACE_Categories
+ categoryDescription: String
- <<constructor>> NACE_Categories(String)
+ returnCategoryIfFirstEncountered NACE_Categories
+ getCategoryDescription(): String
LoadstoneDatabaseUpdater
- model: LoadstoneDatabaseModel
- <<constructor>> LoadstoneDatabaseUpdater()
+ getObjectModelSingleton(): ObjectModel<LoadstoneTotalDataObjectModel, ResultSet, HashMap<String, Object>>
LoadstoneTotalDataObjectModel
+ <<constructor>> LoadstoneTotalDataObjectModel
+ getDatabaseName(): String
+ getDatabaseVersion(): int
LoadstoneDatabaseModel
- String name: String
- latitude: double
- longitude: double
- accuracy: double
- satellites: int
- priority: int
- userid: int
+ ACCESSORS
+ toString(): String
+ equals(Object): boolean

Figure 4.2: Loadstone application UML Class Diagram

4.4 POI Categories

4.4.1 Tendency in POI categories

To effectively categorize POI there has to be defined appropriate category hierarchy. There are a lot of multiple standards which are widely used in many different solutions and technologies. Unfortunately there is no world-wide standard defined for POI categories. More frequent observation can be done when looking at companies like: **Garmin, TomTom, Waze** that those entities try to introduce their own solutions. Garmin offers its own *POILoader* which allows for own category definition [15]. On the other hand TomTom offers 500 different POI categories by default but if user is eager to add custom POIs additional steps need to be done [16]. User must prepare his own files with extension ***.ov2*** and deploy it on GPS device. What is more deployment process depend on device model. Waze also defines it's own POI category standard [17]. There is an endeavour done by **W3C** organization about POI Data Model standard (figure: 4.3). Unfortunately according do W3C wiki it is still in beta version and categories are supposed to be defined in future [18]. Only XML standard was developed (which supposed to be used as world-wide standard for POI data transfer) [19]. There is an apprehensible delamination in field of POI categories. It is difficult to find a standard which could be useful and well known for every data source. Fortunately there are standards defined by governments which describe categories in very detailed way and have endorsement by substantive position as countries governments possess.

4.4.2 NAICS

The North American Industry Classification System (NAICS) widely used in countries of North America. Developed by three organizations: U.S. Economic Classification Policy Committee (ECPC), Statistics Canada, Mexico's Instituto Nacional de Estadistica y Geografia. It is mainly used by Federal agencies to collect data about business entities for statistic purposes. It is successor of ***Standard Industrial Classification (SIC)***. NAICS classification is basing on codes which correspond to category description. Code can at most consist of six digits (the more digits included in code the more detailed category description is). Latest NAICS revision was issued in 2012 [20] [21].

4.4.3 NACE

Statistical Classification of Economic Activities in the European Community (in French: Nomenclature statistique des activités économiques dans la Communauté européenne) - NACE system widely used in European Union. Similarly to NACE it is using combination of codes and category description. Code consists of letter and digits and creates

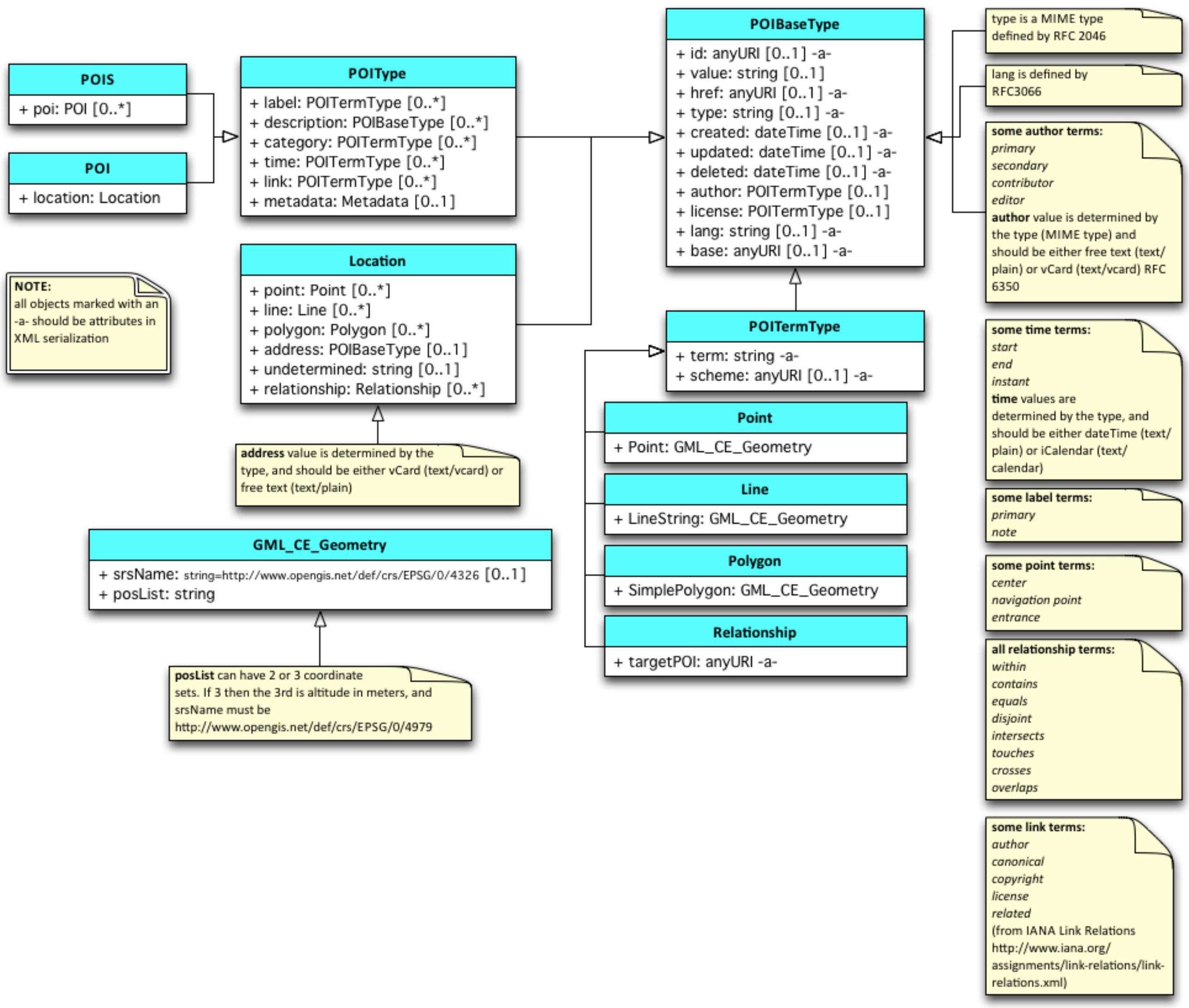

POIS
+ poi: POI [0..*]
POI
+ location: Location
NOTE: all objects marked with an -a- should be attributes in XML serialization
POIType
+ label: POITermType [0..*]
+ description: POIBaseType [0..*]
+ category: POITermType [0..*]
+ time: POITermType [0..*]
+ link: POITermType [0..*]
+ metadata: Metadata [0..1]
Location
+ point: Point [0..*]
+ line: Line [0..*]
+ polygon: Polygon [0..*]
+ address: POIBaseType [0..1]
+ undetermined: string [0..1]
+ relationship: Relationship [0..*]
address value is determined by the type, and should be either vCard (text/vcard) or free text (text/plain)
GML_CE_Geometry
+ srsName: string=http://www.opengis.net/def/crs/EPSG/0/4326 [0..1]
+ posList: string
posList can have 2 or 3 coordinate sets. If 3 then the 3rd is altitude in meters, and srsName must be http://www.opengis.net/def/crs/EPSG/0/4979
POIBaseType
+ id: anyURI [0..1] -a-
+ value: string [0..1]
+ href: anyURI [0..1] -a-
+ type: string [0..1] -a-
+ created: dateTime [0..1] -a-
+ updated: dateTime [0..1] -a-
+ deleted: dateTime [0..1] -a-
+ author: POITermType [0..1]
+ license: POITermType [0..1]
+ lang: string [0..1] -a-
+ base: anyURI [0..1] -a-
POITermType
+ term: string -a-
+ scheme: anyURI [0..1] -a-
Point
+ Point: GML_CE_Geometry
Line
+ LineString: GML_CE_Geometry
Polygon
+ SimplePolygon: GML_CE_Geometry
Relationship
+ targetPOI: anyURI -a-
type is a MIME type defined by RFC 2046
lang is defined by RFC3066
some author terms: primary secondary contributor editor author value is determined by the type (MIME type) and should be either free text (text/plain) or vCard (text/vcard) RFC 6350
some time terms: start end instant time values are determined by the type, and should be either dateTime (text/plain) or iCalendar (text/calendar)
some label terms: primary note
some point terms: center navigation point entrance
all relationship terms: within contains equals disjoint intersects touches crosses overlaps
some link terms: author canonical copyright license related (from IANA Link Relations http://www.iana.org/assignments/link-relations/link-relations.xml)

Figure 4.3: W3C UML Diagram POI Data Model

special hierarchy of: sections, divisions, groups and classes. Section is marked by letter, division by two digits, groups and classes by one digit consecutively. Latest NACE revision (Rev.2) was issued in 2008 as a result of constantly developing and newly appearing organizations especially in telecommunication business [22] [23]. Because Java application which is purpose of this thesis was developed in Poland NACE categorization standard was chosen to apply in it. Obviously information included in NACE are definitely too much detailed to be applied for POI categorization system so only main sections were taken into consideration [12]. Assignment of category and description is presented in table 4.1.

Category	Description
A	Agriculture, forestry and fishing
B	Mining and quarrying
C	Manufacturing
D	Electricity, gas, steam and air conditioning supply
E	Water supply, sewerage, waste management and remediation activities
F	Construction
G	Wholesale and retail trade, repair of motor vehicles and motorcycles
H	Transporting and storage
I	Accommodation and food service activities
J	Information and communication
K	Financial and insurance activities
L	Real estate activities
M	Professional, scientific and technical activities
N	Administrative and support service activities
O	Public administration and defence, compulsory social security
P	Education
Q	Human health and social work activities
R	Arts, entertainment and recreation
S	Other services activities
T	Activities of households as employers undifferentiated goods and services - producing activities of households for own use
U	Activities of extraterritorial organisations and bodies
NOT_CLASSIFIED	Not classified

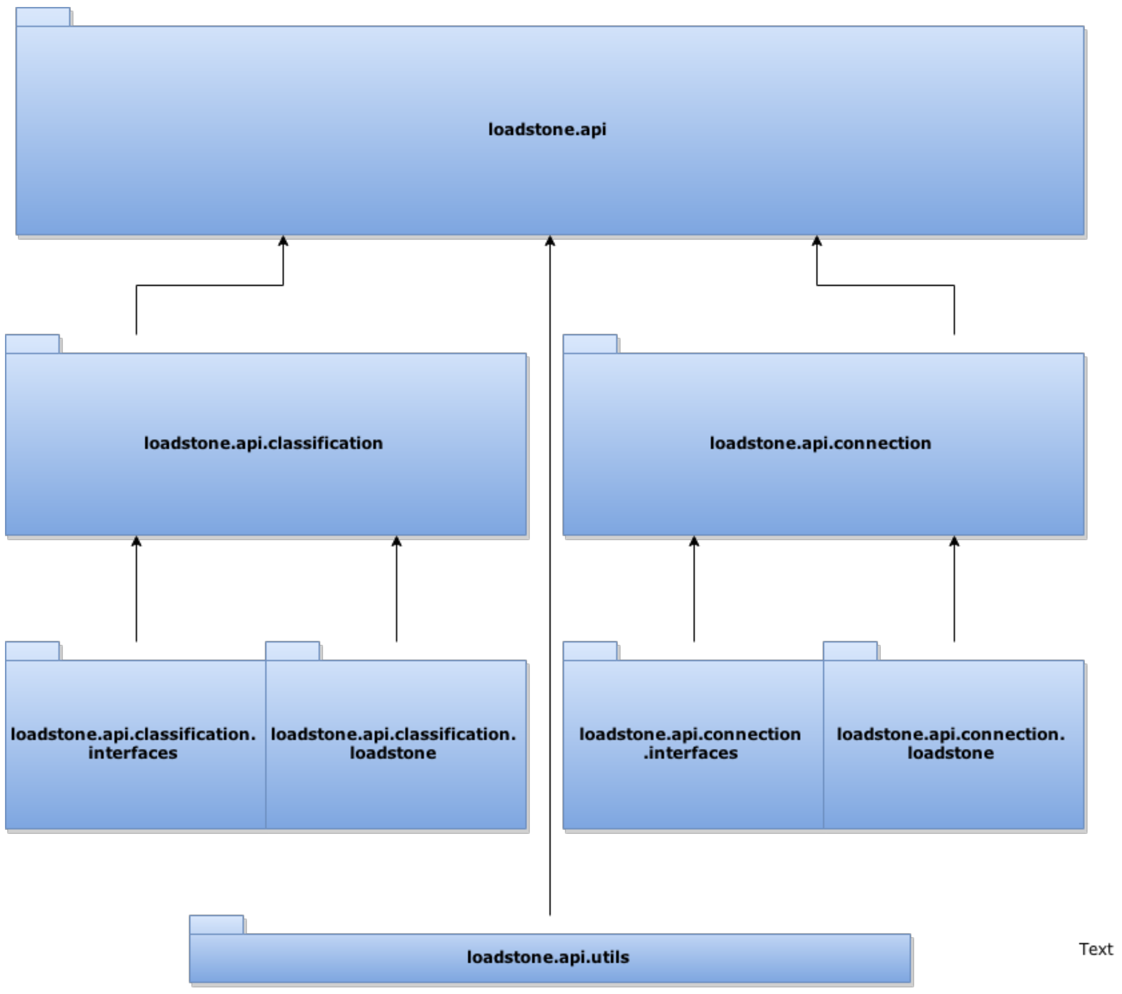
Table 4.1: Implemented NACE categories with assigned human readable description

4.5 Java library as categorization API

Main purpose of Java categorization API is to deliver an interface which would allow for convenient, correct and efficient results. API should be independent of data type which will be delivered. API responsibility is to enable functionalities (as this thesis states - categorization). To assure independence of data sources powerful mechanism of Java interfaces can be used - such approach introduce level of abstraction and allow Java library for easy communication with data sources. Java has very useful feature of importing and using libraries - when whole code is compiled to bytecode classes are bundled into resultant ***.jar*** file - it can be immediately used in another Java project by another programmer. Moreover, resultant Java library project can be compiled with dependencies so programmer who will use library later will not have to bother about classes which were used during library development are were not part of Java API [24]. This approach increases size of ***.jar*** file but release user from inconvenient dependencies downloading and importing into project.

4.5.1 Packages organization

API is divided into three packages: *connection, classification* and *utils*. Such packaging organization assures clear and concise representation of API functionalites and utilities. Functional packages are divided into similar hierarchy to interfaces and packages responsible for particular data source handling (figure 4.4). For purpose of this thesis these packages were named *Loadstone*. Obviously unit tests for corresponding executing classes are located into separate directory (according to maven directory structure) but are included in the same packages (for Java it is counterpart of namespace) [25].



loadstone.api
loadstone.api.classification
loadstone.api.connection
loadstone.api.classification.interfaces
loadstone.api.classification.loadstone
loadstone.api.connection.interfaces
loadstone.api.connection.loadstone
loadstone.api.utils
Text

Figure 4.4: Loadstone API packages organization

4.5.2 Interface loadstone.api.classification.interfaces. AbstractResourcePreprocessing

As mentioned before independence of input data is absolutely compulsory for API. To define preprocessing for each different kind of data source AbstractResourcePreprocessing interface was defined. Classes which will be strictly delegated to pre-process particular source of data by implementation of this interface will be able to communicate with API and pass already preprocessed data for further classification methods.Interface also relies on model object data (DataModel) which is defined in another module of the project - **loadstone-model**. This assures for uniform and scalable solution for next data resources. AbstractResourcePreprocessing defines two methods which has to be implemented by data source dedicated classes to use categorization methods. Interface organization is depicted in the figure 4.5:

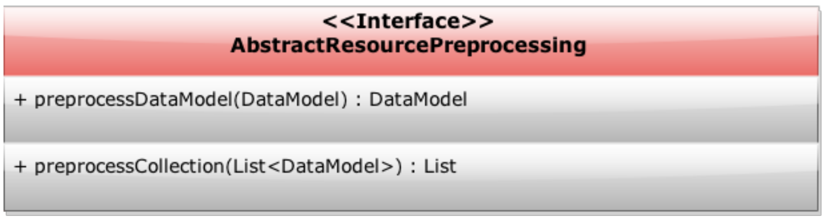

<<Interface>>
AbstractResourcePreprocessing
+ preprocessDataModel(DataModel) : DataModel
+ preprocessCollection(List<DataModel>) : List

Figure 4.5: AbstractResourcePreprocessing UML Diagram

4.5.3 Interface loadstone.api.classification.Classifier

This interface is devoted for classes implementing classification functionalities. It imposes on classes to deliver data source in appropriate way and return List. Main idea of this interface is to inform user that every data model object can be assigned to one or more classes. UML diagram on figure 4.6.

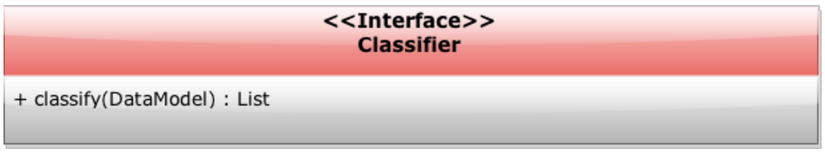


Figure 4.6: Interface Classifier UML Diagram

4.5.4 Interface loadstone.api.connection.interfaces.AbstractResourceConnection

Interface to uniform the input point for data source to API. Each new data source should implement its own class for data gathering and return it in a way interface indicate it. Then data from any other resource (e.g. Loadstone database, RESTful API) can be delivered for categorization functionalities. AbstractResourceConnection is similar to AbstractResourcePreprocessing because of imposing data delivery in term of data model single object and its collection. Interface organization is depicted in the figure 4.7. Parameters for in-

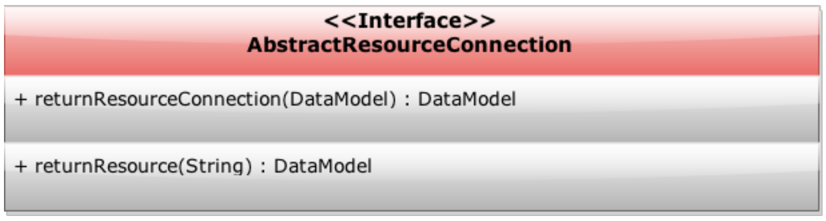

<<Interface>>
AbstractResourceConnection
+ returnResourceConnection(DataModel) : DataModel
+ returnResource(String) : DataModel

Figure 4.7: Interface AbstractResourceConnectionInterface UML Diagram

terface are respectively: List of Strings and String. Parameter types should be sufficient for particular data source invocation composition (e.g SQL statement). Using String type give user big flexibility if we are considering Java as programming language.

4.5.5 Class loadstone.api.classification.loadstone. LoadstonePreprocessing

This class is deliberately delegated to pre-process data obtained from *Loadstone* database. Using dedicated interface it assures final user about possibility of preprocessing in easy to use and in the same time customized way. Class architecture is shown on 4.8. Class or-

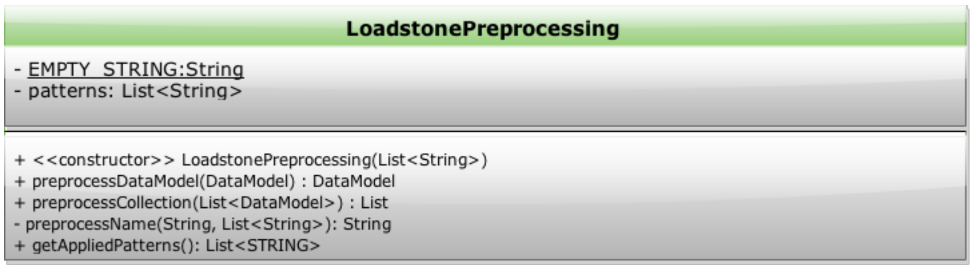

LoadstonePreprocessing
- EMPTY_STRING:String
- patterns: List<String>
+ <<constructor>> LoadstonePreprocessing(List<String>)
+ preprocessDataModel(DataModel) : DataModel
+ preprocessCollection(List<DataModel>) : List
- preprocessName(String, List<String>): String
+ getAppliedPatterns(): List<STRING>

Figure 4.8: LoadstonePreprocessing UML Diagram

ganization is simple. For preprocessing functionality is responsible private method used in methods implementation of AbstractResourcePreprocessing interface. It simply process given input - rejects spaces (default behaviour) and patterns defined by user in constructor. Method is applied for single object DataModel or collection of such objects in corresponding interface methods.

4.5.6 Class loadstone.api.classification.loadstone.LoadstoneBOW

This class is a feature for semi-supervised classification. It is basically a hashmap which values are sections of NACE categorization system and keys are strings which were the most frequent occurring in Loadstone database and assigned empirically to particular category. This class avails user traversing through map and check if category can be assigned immediately. Empirically assigned categories to BOW are presented in table: 4.2 **(it is sample of implementation)**.

4.5.7 Class loadstone.api.classification.NaiveClassifier

This class is implementing naive classification basing on descriptions from NACE categories class (table 4.1) using Classifier interface. It analyses if data model name contains word which is included in particular category description. If data model name contains two or more words from distinct categories descriptions both of them are returned as possible categories to be assigned (under the condition these words have the same number of occurrence). In case of non-homogeneous phrases occurrence from distinct categories only this category is returned which was assigned to word with highest value of occurrences. This class is basing only on heuristic of NACE categories descriptions so can be used for any data source but categorization correctness may differ significantly on input data.

4.5.8 Class loadstone.api.classification.loadstone. LoadstoneSemiSupervisedClassifier

Class which enables to classify POI using class LoadstoneBOW prepared previously. This class does not include descriptions from NACE categories descriptions (table 4.1). Only descriptions from LoadstoneBOW (table 4.2) are taken into consideration. The rule of assignment to category is the same as in case of NaiveClassifier, so multiple categories can be returned as a result. This class is implementing Classifier interface so it can be fluently used later in API.

Category	BOW member
bankomat	K
kościół	U
kurier	J
pgp	J
szkoła	P
atm	K
cmentarz	U
supermarket	G
apteka	Q
bank	K
krzyż	U
most	H
spożywczy	I
restauracja	I
sklep	G
plac	H
hotel	I
euronet	K
bunkier	R
pomnik	R
jezioro	A
poczta	J
rynek	A
jedzenie	I
katolicki	U
pko	K
samochodowy	G
warsztat	G
szlak	A
biedronka	G
orlen	D
lpg	D
straż	O
myjnia	T

Table 4.2: BOW for loadstone database keywords assignments to categories

4.5.9 Class loadstone.api.connection.loadstone.LoadstoneDatabase

This class takes responsibility for gathering Loadstone database. This class is implementing AbstractResourceConnection to get all possible entities from Loadstone database.

LoadstoneDatabaseUpdater Singleton Design Pattern

To modify data inside database loadstone.api.connection.loadstone.LoadstoneDatabaseUpdater should be used. It is using Singleton pattern to assure atomicity of operation on database. It defines static method *getObjectModelSingleton()* returns static object on which user can work to modify data. Such approach allows for data correctness without bothering some other object is using database.

4.5.10 Class loadstone.api.utils.LoadstoneAPIUtils

This class is just utility for user to construct appropriate SQL query to Loadstone database. This utility only expect from user to input patterns which are supposed to be included during search for expected tuples in database.

4.5.11 Class loadstone.api.CategorizationAPI

Class which can be enabled for final usage by user. It combines all methods of preprocessing, data source connection and classification methods. Should be used as input point for data considered to be classified and methods used for classification. Optionally user can add preprocessing method if step of this kind may be useful when POI classification is done. UML diagram on figure 4.9

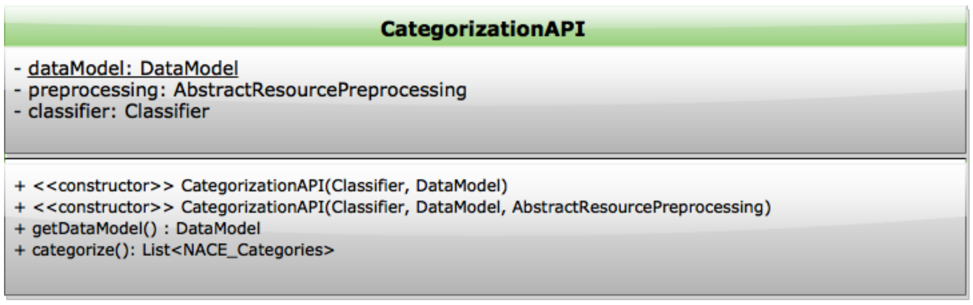

CategorizationAPI
- dataModel: DataModel
- preprocessing: AbstractResourcePreprocessing
- classifier: Classifier
+ <<constructor>> CategorizationAPI(Classifier, DataModel)
+ <<constructor>> CategorizationAPI(Classifier, DataModel, AbstractResourcePreprocessing)
+ getDataModel() : DataModel
+ categorize(): List<NACE_Categories>

Figure 4.9: CategorizationAPI UML Diagram

4.6 Library Usage

Product of created information system is API which can be used by programmer. Usage of library is simple and intuitive. Access to data source and semi-supervised classification task can be done in couple of lines in Java code. The following code snippet demonstrates how to execute categorization task using Loadstone database as data source. Chosen source data is first selected entry given by SQL query to database, contains phrase *bankomat* and will be preprocessed from phrases: *"ul.","adres" and "."*

```java
//define query condition
patternsToLookForInLoadstoneDatabase = new ArrayList<>();
patternsToLookForInLoadstoneDatabase.add("bankomat");
//patterns preparation for preprocessing
patternsToBeTrimmedFromConsideredDataModel = new ArrayList();
patternsToBeTrimmedFromConsideredDataModel.add("ul.");
patternsToBeTrimmedFromConsideredDataModel.add("adres");
patternsToBeTrimmedFromConsideredDataModel.add("\\.");
//Return data from database
List<DataModel> dataModels = new LoadstoneDatabase().
    returnResourceCollection(patternsToLookForInLoadstoneDatabase
    );
//takes first database entry satisfying query condition
dataModelConsidered = dataModels.get(0);
//Initial conditions for API - classification method, data
    source, preprocessing conditions
CategorizationAPI categorizationAPI = new CategorizationAPI(new
    LoadstoneSemiSupervisedClassifier(), dataModelConsidered,
new LoadstonePreprocessing(
    patternsToBeTrimmedFromConsideredDataModel));
//Execute categorization
List<NACE_Categories> nace_categories = categorizationAPI.
    categorize();
```

Chapter 5

Thesis results

5.1 Overview

The main purpose of this thesis is to obtain categorization system for POIs. Such task is complex and difficult to execute for complex data which are present in modern word. It is very difficult to obtain desired results for each kind of data. Text processing is wide domain and requires a lot of computing power and time to return results which are readable for human. When some heuristics is applied processing effort can be decreased in both fields of execution time and required computing power. As mentioned in previous chapters there are many methods which enable method categorization. What is more every method is aimed for different kind of input data. This can indicate that initial knowledge about input data is absolutely compulsory to achieve desired effect and effectiveness. In this chapter the results obtained from implemented information system will be presented for delivered source of data which is Loadstone database.

5.2 Customized semi-supervised classification for Loadstone database

Frequency for bag of words was prepared using bash shell script using *awk* tool which is great utility for text analysis [26]. Script traverses through the Loadstone database text file and counts occurrence of each word which exists in document. Algorithm can be expressed in following way:

Basing on Algorithm 1 it was possible to easily define those phrases which were most occurring and were significant or absolutely unnecessary for text categorization. Obtaining list of words with their occurrences in document there could be selected which can be used in BOW and which can be used for preprocessing as a hint.

```
Data: All words existing in Loadstone database text file
Result: Number of occurrences for each word
initialize array words[];
foreach word in text file do
    transform word to lower case (avail disambiguation)
    increase value by one for index word in array words[]
end
foreach index-word in words[] do
    return word frequency:words[index-word] for index-word
end
```

Algorithm 1: Analysing frequency of words in database

5.3 Preprocessing results

Preprocessing is a task which does not involve high level of complexity in text analysis. It is very important to consider all combinations of ***field separators***(this therm denotes sign of space between analysed words). Usually it is *space* sign but not always. As already mentioned this may vary among input data and such signs could be *tab key* or _ sign. The best idea to apply text preprocessing is to collect statistics which would give information which phrases could be easily applied on data without any regression and conflicts on results. In case of Loadstone database special bash script was prepared which is described in chapter section: 5.2. Results of this script gave heuristics which could be applied for preprocessing. Phrases which can be rejected from analysed text without any bothering about categorization correctness are as follows:

- ***"ul"***
- ***"adres"***
- ***"."***

Phrase *"ul"* occurred respectively **98378** table 5.1 **872770** table 5.2 - the most frequently and does not say about any category. Phrase *"."* is a dot so definitely does not have any significant influence on categorization result. Moreover such removal will ease future categorization. Phrase *"adres"* occurred **759337** table 5.2. This number was obtained from text file containing all data available for Loadstone project. There is a test suite in *loadstone.api.classification.loadstone.LoadstonePreprocessingTest* class which presents execution of preprocessing on specially prepared DataModel's for this purpose. Results for analysed texts with those particular patterns to be rejected from input text are as follows:

This example shows that phrases: *"adres ul ul. " and " "* were successfully rejected and there is significantly less data to analyse in further categorization process.
Another example is taken directly from Loadstone database. Preprocessed text is an Loadstone database entry.

Data: ***"adres ul ul. fake name to be trimmed "***
Result: ***"fake name to be trimmed"***

Example 2: Preprocessing example using mocked data

Data: ***"adres ul. kramarska 15 warszawa rembertów"***
Result: ***"kramarska 15 warszawa rembertów"***

Example 3: Preprocessing example using data extracted from Loadstone database

As it is noticeable also in this case preprocessing was successful and did not rejected any data important from categorization point of view. Phrase: *"adres ul. "* was trimmed correctly.

5.4 Classification results

5.4.1 Overview

Classification is much more complicated task than preprocessing. It involves complex analysis of input data, possible misconceptions or not precise heuristics. Moreover, categorization usually should be unambiguous but not always. Sometimes POI can be classified to more than one category as an example there can be considered a ***church*** which can be categorized as *"U - Activities of extraterritorial organisations and bodies "*(according to NACE standard notation) or may as well classified as *"R - Arts, entertainment and recreation "*. However what is worth to mention is correlation between those two categories. POI ***church*** cannot always be classified as the latter but always as the first one. Such cases can be complex and perfect solution is unlikely to find as well. The correctness of classification results may also vary among judging human beings.

5.4.2 Naive classifier categorization results

As mentioned in section: 4.5.7 naive classification bases on heuristics inherited from NACE standard. As heuristic for unambiguous results it takes information about frequency of given phrase in NACE category description. When one heuristic overcome in frequency second heuristic the latter is rejected. Two of more categories as result are returned when frequency is the same for each of given heuristic. When heuristic is not available in input data classification is unavailable. The following example can be presented for imaginary data:
As heuristic ***"Manufacturing"*** frequency was higher than ***"administration"*** (six occurrences versus three occurrences) category C is returned as result for categorization system.

Data: ***"Manufacturing Manufacturing Manufacturing Manufacturing Manufacturing Manufacturing administration administration administration"***
Result: ***"C - Manufacturing "***

Example 4: Naive classifier example using mocked data for different phrase frequency

Another example demonstrate naive classification when heuristic frequency has the same value of occurrences for each component:

Data: ***"Manufacturing Manufacturing Manufacturing administration administration administration"***
Result: ***"C - Manufacturing ", "O - Public administration and defence; compulsory social security "***

Example 5: Naive classifier example using mocked data for uniform phrase frequency

As heuristic ***"Manufacturing"*** frequency was the same as heuristic ***"administration"*** category C and O is returned as result for categorization system.

Naive classifier for Loadstone database

As heuristic is suit to data which are definitely not included in Loadstone database input data description (main issue is comparison between English and polish language) naive classifier does not have significant utility during Loadstone database analysis. Example 7 proves it:

Data: ***"bankomat i oddział bz bwk atm 24h bank ul. jana pawła ii 12 sieradz"***
Result: ***"Not classified"***

Example 6: Naive classifier using data extracted from Loadstone database

As none heuristic exists in input data from NACE categories description "Not classified" result is returned.
Note: all examples are implemented as test suite of unit tests in class loadstone.api.classification.NaiveClassifierTest

5.4.3 Semisupervised categorization for Loadstone database results

Semisupervised categorization is very useful but requires much more complex analysis. It requires knowledge about heuristics included in input data. To perform semi-supervised categorization Loadstone bag of words was prepared previously (description chapter section: 4.5.6). Thanks to initial knowledge what kind of phrases could be used for categorization, results correctness is decent. Important factor is complete isolation from heuristics included in NACE standard category description (such approach could possibly introduce regression). Following examples demonstrates performance of semi-supervised categorization intended deliberately for input data derived from Loadstone database:

Data: ***"bankomat i oddział bz bwk atm 24h bank ul. jana pawła ii 12 sieradz"***
Result: ***"K - Financial and insurance activities "***

Example 7: Semi-supervised categorization using data extracted from Loadstone database

Because of heuristics: ***bankomat, atm, bank*** which are defined in Loadstone BOW are all assigned to category K unambiguous result is retrieved. This sample clearly demonstrates difference between naive (example: 6) and semi-supervised classification. Naive classification is basing on heuristic taken from description of NACE categories table: 4.1. Input data were in completely different (Polish language) than heuristic (English language). Possible improvement for this case could be NACE categories description to Polish language.

Data: ***"bankomat and pizza"***
Result: ***"I - Accommodation and food service activities ", "K - Financial and insurance activities "***

Example 8: Semi-supervised categorization using mocked data with uniform phrase frequency

Because of heuristics: ***bankomat, pizza*** which are defined in Loadstone BOW but are assigned to different categories the result is ambiguous. Heuristics frequencies are equal so two categories are returned as categorization results.
There was also performed comparison of imaginary data indicated deliberately to NACE standard category description analysed by using method of semi-supervised categorization. The result is as follows:
In comparison to example 4 obtained result is completely irrelevant to reality and does not give correct result.
Note: all examples are implemented as test suite of unit tests in class loadstone.api.classification.LoadstoneSemiSupervisedClassifierTest

Data: ***"Manufacturing Manufacturing Manufacturing Manufacturing Manufacturing Manufacturing administration administration administration"***
Result: ***"Not classified"***

Example 9: Semi-supervised categorization using mocked data not related to Loadstone BOW

Database analysis results

In order to construct suitable semi-supervised categorization there was performed database analysis. Analysis was done using *bash* shell script **analyzeDB.sh**. Script enables whole database analysis. The result of such analysis is number occurrences of phrases separated by white space. Text file analysed was: [11]. During script execution there was processed **37549** different phrases. Execution of the script took: **2.937 sec.** Results of ten most frequent phrases are depicted in table 5.1.

Occurrences	Phrase
98378	ul.
11153	"wysokość
11117	"bankomat
9105	"parking
8914	"kościół
7517	i
6999	"kurier
6959	pgp
6691	1
6072	24h
5569	"szkoła

Table 5.1: Descending phrase occurrence frequencies all addresses

Results of of ten most frequent phrases occurrences for text file containing only POIs are depicted in table 5.2. Execution of the script took: **24.274sec.**

Occurrences	Phrase
872770	ul.
759337	"adres
296042	"skrzyż.
102080	warszawa
53792	wrocław
45019	kraków
35939	1
34733	3
30102	"dzielnica
29675	2

Table 5.2: Descending phrase occurrence frequencies POIs

Obviously not all of them were included in bag of words for semi-supervised classification. All words with adjusted category were chosen empirically. In total to Loadstone bag of words there was included **72** words.

5.5 Naive and semi-supervised classification comparison

Two categorization methods which were implemented in Java library and are delivered through API may appear to be similar to each other. Both of them are using somehow defined heuristics and return result in form of previously defined category. Main which distinguish this methods is heuristic definition. In case of **naive classification** heuristic is imposed from very beginning. User does not have any influence on the form of given classification tips. In case of Java library implementation this may appear to be a drawback because tips for classification are defined in English language. However, naive method disqualify source data which are not defined in English language (such situation is presented in example 9). Possible solution would be to translate NACE categories description (table 4.1) to other languages. This may result in higher naive classification correctness. Naive classification details are described in section 4.5.7 . On the other hand user does not have to apply any hints for classification. **Semi-supervised** heuristic can be defined by user. If user performed analysis of input data before classification e.g. phrases frequency occurrence (like in case of this thesis) and defined internal BOW, results correctness should increase (as there not exists categorization method which would enable more accurate classification than human being). Additionally, initial BOW definition solves issue of input data language - human beings are able to define keywords hints for every language. Semi-supervised classification details are presented in section 4.5.8. This two methods present different approach to classification problem solution and should be used depending on the form of input data and user involvement.

5.6 Performance results

Performance was measured for one hundred iterations of classification method invokes. Additionally to assure results correctness five iterations were added at the beginning as a warm-up. During performance measurement time of data access to database was discarded. Performance results only reveal time of implemented classification and preprocessing methods.

- Preprocessing performance results are presented in table 5.3.

Operation	Time	Error	Unit
Example 2	0,003	0,001	ms/op
Example 3	0,002	0,001	ms/op

Table 5.3: Preprocessing performance results

- Naive classifier performance results are presented in table 5.4.

Operation	Time	Error	Unit
Example 4	0,004	0,001	ms/op
Example 5	0,003	0,001	ms/op
Example 6	0,007	0,001	ms/op

Table 5.4: Naive classifier performance results

- Semisupervised categorization results are presented in table 5.5.

Operation	Time	Error	Unit
Example 7	0,001	0,001	ms/op
Example 8	0,002	0,001	ms/op
Example 9	0,003	0,001	ms/op

Table 5.5: Semi-supervised categorization performance results

As performance results states the worst case scenario is when there is applied categorization method which is not intended for given kind of data (example 6). The rest of results are on quite same level and does not cost much. Even worst case scenario is quite cheap solution for modern mobile devices and computers as all results are expressed in milliseconds. The biggest concern during consideration of presented test cases should be I/O time operation. This factor was not included as it was not purpose of this thesis. On the other hand when considering functionality and performance of whole information system this factor requires further investigation.

Hardware used for performance testing

All tests were executed on notebook with hardware parameters: CPU - Intel Core i7-4702MQ, 16 GB RAM, Graphic Card - GeForce GT 740M, HDD Hard Drive.

5.7 Conclusion

The purpose of this thesis was to provide overview of categorization methods using text analysis which can be useful during POI classification. As a result of this research there was developed Loadstone database (using script *createDB.sh*), Java library which delivers interface for *Loadstone* database access and provides categorization methods mainly customized for data included in database. Such customization was possible thanks to automatic analysis of phrase frequencies in database which functionality enables script *analyzeDB.sh*. Library can be useful when developing applications using GPS systems to deliver information about POI in the neighbourhood. Main benefit of library is independence on data source so using specially designed interface there is possibility to deliver many kind of data sources and combine it in one application. The topic of POI categorization is wide and complex domain. This domain mainly involves text processing and heuristics about possible hints which can be used to obtain expected results. As presented in chapter section 5.4 result may differ according to applied categorization method. A lot of factors need to be considered when categorization task is applied like: language of input data, model of input data (it is possible to analyse whole documents or large fragment of texts or ready to use databases), size of input data (performance in complex information systems is one of significant factor). It is really difficult challenge to estimate what kind heuristics data can be obtained and work out system which would be versatile for every data source. Another important factor is choice of technology for such information system. Java possess high native library support and community as well [27][28]. Unfortunately this involves a lot of research in Internet to find suitable collection which can be used in implementation. What is more it is often practice that code construction is getting more complex and difficult to maintain in future. Example of such practice is returning classification result as *list of categories* instead of using construction called *tuple*. Such constructions are available in programming languages like: *Scala* [29] or *Python* [30]. What is more syntax of those programming languages is easier to read (it is possible to omit long-line Java syntax - not so huge effort is pushed on object encapsulation) and flexibility. Another important issue are external tools which could be used for easier maintenance of information system. Such tool could be *Apache Spark* [31]. It increases the performance of execution and is great manager for feature extraction, transformations or SVM. On the other hand this technology is quite new and requires significant effort at the beginning to use it.

Bibliography

[1] Mukesh A. Zaveri Mita K. Dalal. Automatic text classification of sports blog data. pages 219–222. Computing, Communications and Applications Conference (ComComAp), January 2012.

[2] Christopher Manning. Natural language processing.tf-idf weighting. `https://class.coursera.org/nlp/lecture/187`. (last access: 18/06/2015).

[3] Kuan-Liang Liu and Tzu-Tsung Wong. *Naive Bayesian Classifiers with Multinomial Models for rRNA Taxonomic Assignment*, volume 10. Computational Biology and Bioinformatics, IEEE/ACM Transactions, September 2013.

[4] Chih-Jen Lin Chih-Wei Hsu. *A comparison of methods for multiclass support vector machines*, volume 13. Neural Networks, IEEE Transactions.

[5] `http://en.wikipedia.org/wiki/Support_vector_machine`. Support Vector Machine definition (last access: 18/06/2015).

[6] Yiping Yang Lu Peng, Yibo Gao. Automatic text classification based on knowledge tree.

[7] `http://en.wikipedia.org/wiki/Tf%E2%80%93idf`. TF-IDF definition (last access: 18/06/2015).

[8] Haoran Wu Bing Liu Gao Cong, Wee Sun Lee. Semi-supervised text classification using partitioned em. `http://www.cs.uic.edu/~liub/publications/DASFAA04.pdf`. (last access: 18/06/2015).

[9] Seong-Bae Park Sang-Jo Lee Su Jeong Choi, Hyun-Je Song. A poi categorization by composition of onomastic and contextual information. Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM.

[10] Fattah M.A.-Ren F. Elmarhumy, M. Automatic text classification using modified centroid classifier. volume 2, pages 38–45. Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM International Joint Conferences, August 2014.

[11] Grzegorz Złotowicz. http://gps.zlotowicz.pl/tekstowe/cala%20polska%20adresy.txt.zip. Loadstone project text file containing all addresses (last access: 19/06/2015).

[12] Sheldon Neilson. http://www.neilson.co.za/sqlite-database-model/. Website with library enabling object relational mapping of SQLite database in java (last access: 19/06/2015).

[13] wikipedia.org. http://en.wikipedia.org/wiki/Component_%28UML%29. UML langugage documentation - component (last access: 18/06/2015).

[14] maven.apache.org. http://maven.apache.org/guides/introduction/introduction-to-the-pom.html. Maven framework dependency management documentation (last access: 18/06/2015).

[15] Garmin. http://www.garmin.com/us/maps/poiloader/. POI update tutorial for Garmin devices (last access: 18/06/2015).

[16] TomTom. http://www.tomtom.com/en_gb/licensing/products/enhanced-content/find/points-of-interest/. POI as product in TomTom company (last access: 18/06/2015).

[17] Waze. https://wiki.waze.com/wiki/To_Do_lists_-_POI_Categories. Waze company POI categories (last access: 18/06/2015).

[18] W3C. http://www.w3.org/2010/POI/wiki/Data_Model#Category. POI implementation details according to W3C (last access: 18/06/2015).

[19] W3C. http://www.w3.org/2010/POI/documents/Core/core-20111216.html. POI documentation according to W3C (last access: 18/06/2015).

[20] census.gov. http://www.census.gov/eos/www/naics/. NAICS standard documentation (last access: 18/06/2015).

[21] wikipedia.org. http://en.wikipedia.org/wiki/North_American_Industry_Classification_System. NAICS standard description(last access: 18/06/2015).

[22] siccode.com. http://siccode.com/en/pages/what-is-a-nace-code. NACE standard description (last access: 18/06/2015).

[23] Central statistics office Ireland. http://www.cso.ie/px/u/NACECoder/. NACE standard documentation (last access: 18/06/2015).

[24] Oracle Corporation. http://docs.oracle.com/javase/7/docs/api/. Java 7 programming language documentation(last access: 18/06/2015).

[25] maven.apache.org. https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html. Maven framework organization documentation (last access: 18/06/2015).

[26] GNU General Public License. http://www.gnu.org/software/gawk/manual/gawk.html. Gawk language documentation(last access: 18/06/2015).

[27] google.com. https://github.com/google/guava. Guava utility java library website (last access: 18/06/2015).

[28] Apache Software Foundation. https://commons.apache.org/. Apache java library with utilities website (last access: 18/06/2015).

[29] Scala. http://www.tutorialspoint.com/scala/scala_tuples.htm. Scala programming language data structure tuples documentation (last access: 18/06/2015).

[30] Python. https://docs.python.org/2/tutorial/datastructures.html. Python programming langugage data structures tuples documentation (last access: 18/06/2015).

[31] Apache Software Foundation. https://spark.apache.org/. Spark cluster computing framework website (last access: 18/06/2015).