Industrial and Applications Paper

Exploring Text Classification for Messy Data: An Industry Use Case for Domain-Specific Analytics

Industrial Paper

Laura B. Kassner
Daimler AG
71059 Sindelfingen
laura_bernadette.kassner@daimler.com

Bernhard Mitschang
Institut für Parallele und Verteilte Systeme
Universitätsstraße 38
70569 Stuttgart
bernhard.mitschang@ipvs.uni-stuttgart.de

ABSTRACT

Industrial enterprise data present classification problems which are different from those problems typically discussed in the scientific community – with larger amounts of classes and with domain-specific, often unstructured data. We address one such problem through an analytics environment which makes use of domain-specific knowledge. Companies are beginning to use analytics on large amounts of text data which they have access to, but in day-to-day business, manual effort is still the dominant method for processing unstructured data. In the face of ever larger amounts of data, faster innovation cycles and higher product customization, human experts need to be supported in their work through data analytics. In cooperation with a large automotive manufacturer, we have developed a use case in the area of quality management for supporting human labor through text analytics: When processing damaged car parts for quality improvement and warranty handling, quality experts have to read text reports and assign error codes to damaged parts. We design and implement a system to recommend likely error codes based on the automatic recognition of error mentions in textual quality reports. In our prototypical implementation, we test several methods for filtering out accurate recommendations for error codes and develop further directions for applying this method to a competitive business intelligence use case.

Categories and Subject Descriptors

H.3.1 [**Content Analysis and Indexing**]: Linguistic Processing; H.3.3 [**Information Search and Retrieval**]: Information Filtering; H.4.2 [**Information Systems Applications**]: Types of Systems—*Decision Support*
; J.1 [**Administrative Data Processing**]: Manufacturing

General Terms

Text-Based Classification, Domain-Specific Semantic Resources

Keywords

recommendation system, automotive, text analytics, domain-specific language, automatic classification

1. INTRODUCTION

Analytics and automatized processing of unstructured data to support business processes and decisions have become a topic of interest for the research community and the enterprise world in recent years only [15]. Companies are realizing that valuable knowledge can be gained especially from the large amounts of unstructured text data which they are storing internally and able to access publicly on the social web. In past decades, this knowledge was only accessible to the human mind. The means for storing and processing large amounts of data and scalable text analytics or cognitive computing tools [5] are both relatively new developments.

In cooperation with a large automotive original equipment manufacturer (OEM), we have developed an overarching use case in the area of quality management, with a concrete example for supporting human labor through text analytics. This use case presents a particularly challenging classification problem with several hundred possible classes and mainly unstructured text data as input. We develop a modular environment for classifying these data with the help of natural language processing. In this paper, our main points are (1) a first proof of concept for a "messy" industrial data source and (2) the investigation of the usefulness of a legacy domain-specific resource in the context of a new analytics task. To this end, we evaluate various adaptations of an established classification algorithm in order to customize it to the given situation.

1.1 Motivation

Industrial enterprises are generating and collecting large amounts of unstructured text data. These data are often highly domain-specific. Most current approaches to automatically analyzing these unstructured data with traditional

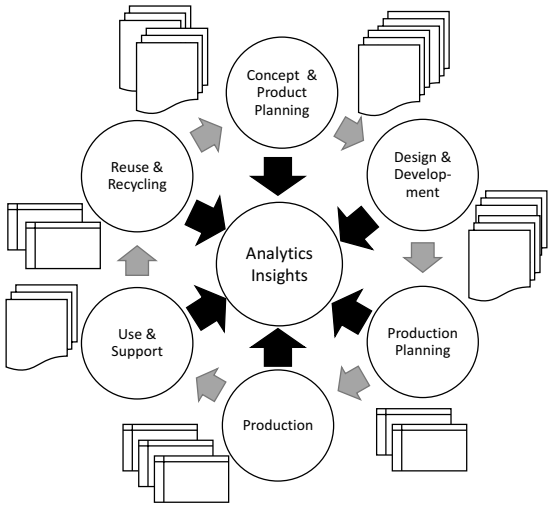

Figure 1: Data Analytics around the Product Life Cycle

analytics for structured data are either very specific and case-based or too generic [11]. What is needed is a flexible framework for analytics with re-usable and modular resources and analytics toolboxes.

Enterprises are beginning to transition to more widespread and streamlined use of text analytics. But in day-to-day business, a lot of manual effort is still used to process unstructured data, for example in quality management or customer relationship management. This effort is important because these data contain value-adding knowledge, and human expertise cannot and should not be fully replaced in these tasks. But in the face of ever larger amounts of data, faster innovation cycles and higher product customization especially in the manufacturing industry, human experts need to be better supported through data analytics. Beyond the support of current manual analytics tasks, data analytics can also provide important novel business insights (cf. Figure 1).

Domain-specific resources, for example taxonomies of domain-specific languages, are often developed for a single case-based analytics scenario and rarely re-used for others. This leads to inefficiencies and loss of valuable knowledge. Related research on this topic [11, 12] has proposed requirements and an architectural paradigm for flexible, re-usable and value-adding analytics software. Within this framework we develop a proof of concept for a toolbox using legacy code and semantic resources, unstructured data from several sources, and modular, tailored text analytics.

1.2 Contribution and Outline

When processing damaged car parts for quality improvement and warranty handling, quality experts have to read large numbers of text reports from various sources and assign error codes to damaged parts from a large pool of options. We investigate methods to handle these unstructured text data analytically and to support this labor through automatic recommendation of likely error codes, which is a specialized application of automatic classification.

This use case is different from typical classification problems in several respects: The amount of potential classes is larger (several hundred), and the data to be classified are short text reports which we term "messy data": Text which consists of non-standard, domain-specific language, riddled with spelling errors, idiosyncratic and non-idiomatic expressions and OEM-internal abbreviations.

We design and implement the Quality Engineering Support Tool QUEST with an included Quality Analytics Toolkit (QATK), a system to recommend likely error codes based on the automatic recognition of error mentions in textual quality reports. It also includes the functionality for comparing error distributions across different data sources, which we have implemented for a public data source, the database of automotive malfunctioning complaints maintained by the Office of Defects (ODI) of the National Highway Traffic Safety Administration (NHTSA) [13].

We test a domain-specific and a domain-ignorant version of a custom classification algorithm derived from k-Nearest-Neighbors (kNN), and evaluate both against a baseline which ignores the text content as well as with respect to their industrial feasibility.

We also investigate in more detail the influence of the report source and its place in the error classification process – early or late and contributed by mechanics or part suppliers.

In sum, we address three different industry-focused goals: (1) to make classification work easier for the workers who do it by sorting error codes in a meaningful way, (2) to do this as early as possible in the life cycle of a damaged car part, and (3) to make data comparable to other data sources through text analytics.

The remainder of this paper is structured as follows: In the following chapter 2, we present the research background. In chapter 3, we describe in detail the industrial context of our application (3.1) and discuss the challenges presented by the data (3.2). The methods and implementation are explained in chapter 4. We then present and evaluate two experiments and an extension of the original use case in chapter 5. We look at the feasibility of text-based error classification (5.2) and the role of the report source for classification accuracy (5.3) as well as the potential use of automated error code assignment to compare the performance of a product with competitors (5.4). Chapter 6 concludes our paper with a summary of the results and outlook on future research.

2. BACKGROUND

In prior research, we have motivated the need for Product Life Cycle Analytics integrating structured and unstructured data within a holistic framework [11]. In this chapter, we present two background foci which are relevant for the present research topic, namely text analytics in the automotive industry (2.1) and the general field of automatic text categorization (2.2), which deals with applying classification algorithms to text data.

2.1 Text Analytics in the Automotive Industry

In the automotive domain, there have been a number of efforts to use unstructured text data for business analytics. Most recently, several interconnected research projects [3, 16, 8] have developed software for extracting information about frequent problems from internal error reports and customer sentiment related to problems from social media. This research has also led to the creation of a valuable semantic resource, a taxonomy of parts and errors [17, 18], which we

use as a central component in our analytics framework (cf. 4.5.3) as well as in the domain-specific classification algorithm.

2.2 Automatic Text Categorization

Automatic text categorization has been a widely researched field since the late 1990s / early 2000s [19]. Typically, the task is to label texts as belonging to one of a small number of classes, e.g. one of five different topics for news texts or one of three known potential authors for literary works. Our task differs from this in that we have a very large number of classes.

Features for classification are usually derived by their information content across a large number of texts. Using pre-defined features such as author, user mentions and signal words in tweets [20] has also been shown to achieve high accuracies. We investigate the suitability of features drawn from a domain-specific knowledge resource.

An important part of this investigation is the mapping of words in the text to concepts from a semantic resource. We agree with the argument of [10] that words and stems do not represent the semantic content of a text very well. They try to map words to concepts using WordNet [4]. This is important for disambiguation, but also for highlighting and exposing shared concepts as latent features across texts with no shared word material. Because our semantic resource is rich in synonyms, we can map text to concepts via surface entity recognition.

[7] point out a weakness of the kNN algorithm which we also encounter – it is instance-based and thus potentially memory-intensive. They develop a modified kNN which creates generalized instances and representatives of instances based on local neighborhoods. We modify kNN to use representatives of instances based on abstractions of texts to contained concepts in our domain-specific variant, and also store these instances in a relational database with on-the-fly access to further address memory concerns.

3. PROBLEM DESCRIPTION

In this chapter, we present the industrial context of our use case (3.1) and discuss the challenges of the data we are working with (3.2).

3.1 Industrial Context

An automotive OEM is evaluating car parts which were removed during repairs in customer-owned vehicles for the warranty process and in order to gain insight into quality issues. The evaluation is a complex and multi-step process which involves unstructured text data from many sources and large amounts of human work:

The removed and potentially damaged car part is first evaluated in a short textual report by the mechanic who removed it. It is then shipped to the OEM, where an optional initial report can be written. Next, the car part is sent on to the supplier who manufactured it. The supplier evaluates the part's damage, writes a textual report and assigns a damage responsibility code (indicating whom they hold responsible for the problem). Eventually, a quality expert at the OEM assigns the car part a final error code and writes a short final report. This process of data accumulation is depicted in Fig. 2.

In order to assign the correct error code, the quality expert needs to read all reports written about the part at hand and

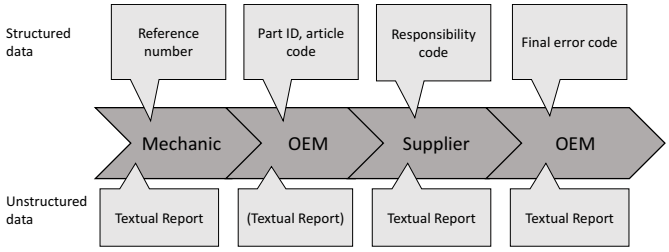

Figure 2: Process of data accumulation

then pick the error code from a list of potentially over 100 error codes available for this type of car part.

Due to the largely textual nature of the data and the large number of potential error codes, a substantial amount of the quality experts' time is taken up by assigning error codes to known errors. We want to support the quality workers by offering them automatically derived error code recommendations to speed up the decision process. If the set of error codes for a given part is smaller and sorted, the final error code assignment will take less time. The quality experts can then spend more time investigating novel or more complex errors, thus improving the overall quality of the evaluation.

3.2 Challenges of the Data

We developed our prototype around a randomized and anonymized subset of the original data input from the evaluation documentation tool. We extracted a fraction of the data concerning three larger component classes which have already been assigned error codes, such that a portion of the data can be used for evaluation. Other component classes are subject to future research to further validate the approach. Any information about individual persons (quality workers assigned to tasks, supplier contacts, etc.) was removed, as well as select mentions of supplier names and the OEM name, and the fields listing vehicle identification numbers and information about vehicle make and model. In total, we are working with data for 7500 individual car parts. These data, including the text reports, are stored across several tables in a relational database.

We define all data pertaining to an individual component as a data bundle. The data bundles for each component are structured in the following way:

- A component is identified by a unique reference number and also assigned an article code and a part ID. These have vastly different levels of granularity: In our data set, there are 831 distinct article codes and 31 distinct part IDs.
- A further challenge is the extremely high number of distinct error codes: There are 1271 distinct error codes in our data set of 7500 data bundles. 718 of these error codes only appear a single time, so we remove them for our experiments since nothing can be learned from them for the classification task at hand (for information extraction tasks we would of course consider them). This leaves us with 553 potential classes and 6782 data bundles with an error code that appears more than once. The largest number of distinct error codes for one part id in our data set is 146, and 25 of the 31 part IDs have instances of over 10 error codes.

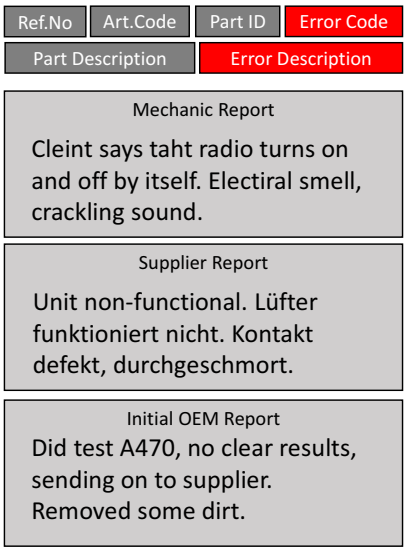

Figure 3: Structure of the data bundles before final classification, with missing structured data fields highlighted in red

- Each component is further associated with three or four textual reports: (1) the mechanic report, (2) the initial OEM report (optional), (3) the supplier report and (4) the final OEM report.

Other textual resources are the standardized descriptions of part id and error code in German and English. Fig. 3 shows a schematic overview of one data bundle and contains a fictional but representative text example.

These text resources can be used to derive textual indicators of error codes during the training phase of classification. In the testing phase, we use only the mechanic report, the optional initial report, the supplier report and the part id description. This reflects the circumstance that the final OEM report and the error code description are unavailable as sources for textual indicators in data which have not yet been assigned an error code. In 5.3, we investigate the performance of classification when using only the mechanic report or only the supplier report as input to the classifier.

The reports in our data sample are mostly a mix of German and English, but the entire data set contains several other languages. Robustly recognizing meaning in multilingual input is therefore a requirement for our system. In our prototype, we achieve this by using a multilingual semantic resource for the domain-specific approach and primarily relying on natural language processing steps which are language-independent. The domain-ignorant approach does not address multilinguality. Future investigations will also deal with how to incorporate language-specific tools.

4. METHODS AND IMPLEMENTATION

In this chapter, we present the general architecture of our analytics toolkit (4.1), develop a basic algorithm derived from kNN (4.2), discuss our adaptation of this algorithm in the domain-specific and the domain-ignorant variant (4.3), and describe the processing pipeline (4.4) as well as the prototypical implementation (4.5).

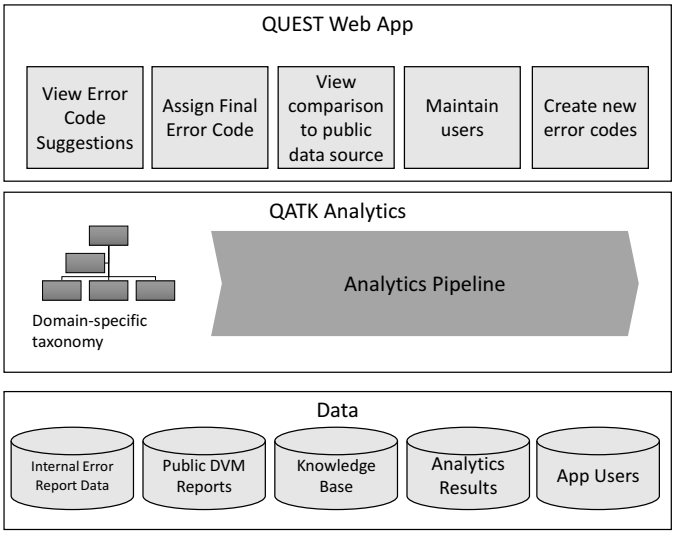

Figure 4: General architecture of the recommendation system

4.1 Architecture

The architecture of our general analytics framework comprises data sources and two main components (cf. Fig. 4), the Quality Analytics Toolkit (QATK) and the Quality Engineering Support Tool (QUEST) web application.

The QATK provides a highly modular analytics pipeline to process the text data, build a knowledge base representing structure extracted from the unstructured data, and assign scored and sorted potential error codes to new data bundles using the classification approach outlined in 4.3. This pipeline can be seen in detail in Fig. 8 and is further discussed in 4.4.

The functionality of the QATK can easily be adapted to classify data from a different source according to the same classification schema of part IDs and error codes as the internal source. This allows for comparisons of error distributions across different data sources. In 5.4, we show how such comparisons can be implemented for a public data source, the complaints database of the NHTSA ODI available via safercar.gov [13].

4.2 Basic Classification

The reports and the error key and part ID labels contain unstructured text descriptions of the problems encoded by the error keys. If we can abstract from these descriptions to structured features, we can use them as input for a classification algorithm.

In our approach and in the prototypical implementation, we employ a classification algorithm derived from the k-Nearest-Neighbor algorithm. The standard kNN algorithm (Fig. 6) determines class membership of a data point by majority vote from the classes of the k nearest neighbors of this data point, where nearness is equivalent to similarity with respect to the chosen classification features. This majority vote can also be weighted by the individual nearness of neighbors, which is determined by a similarity measure of choice between the data points.

kNN is a „lazy" machine learning algorithm – it does not build a statistical model, it just holds instances of already

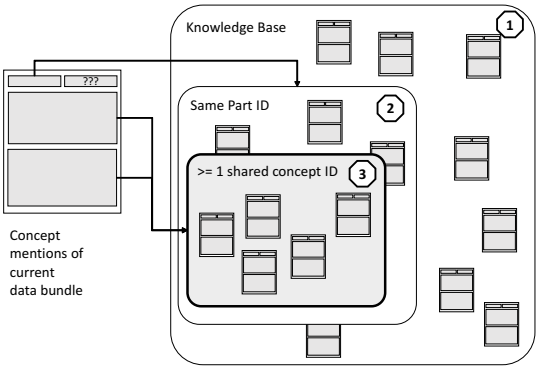

Figure 5: Selecting candidate nodes

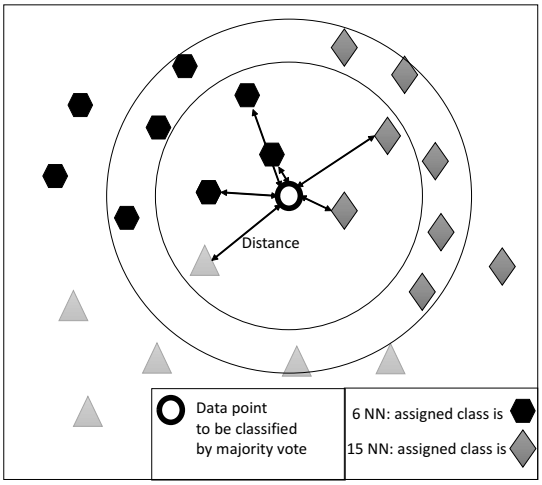

Figure 6: Standard unweighted instance-based kNN classification for k = 6 and k = 15 with class assignment by majority vote

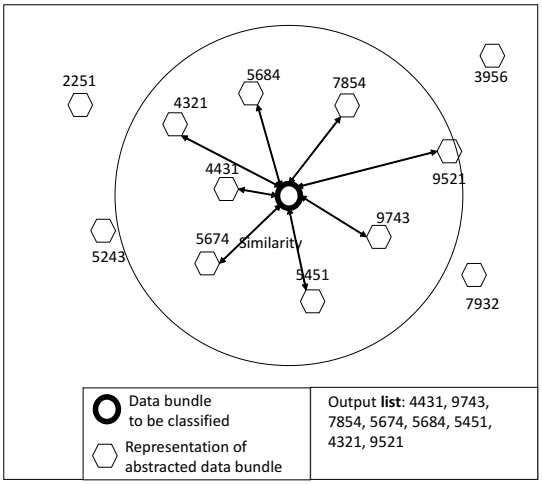

Figure 7: Adapted ranked-list kNN classification for abstracted data bundle representations

classified data in memory (or on disk, as is the case in our implementation) for comparison with the data instances to be classified.

We have decided to focus on variants of kNN in our proof of concept for several reasons: We are dealing with an extreme multi-class classification problem, and in contrast to many other algorithms, kNN handles multi-class classification in a straightforward manner. It is also instance-based and therefore allows for predictions about class membership even with a small data set and a large number of classes, which makes it especially suitable for our example data. Further, since our focus is on establishing whether this particular multi-class classification problem can be solved semi-automatically and on investigating the role of domain-specific vs. domain-ignorant features, we decided in favor of an algorithm which allows for very straightforward control over the features, is easy to implement and easy to understand, and can easily be used with different similarity or distance measures, such that we are not fixed on representing our data in one particular way. Other algorithms are to be investigated in future research after we have established that the challenges of the data can be met at all.

Thus, we can derive a bare-bones classification algorithm with maximum parametrizability:

Given set of objects S with assigned classes, object o without assigned class
for o_i **in** S: calculate similarity(o, o_i)
sort S by descending similarity
assign class to o based on sorting of S

The similarity measure, the choice of features on which to base the similarity measure and the method for deriving the class assignment from the similarity ranking can be adjusted to the needs of the use case.

4.3 Adaptation and Parametrization

Starting from the basic algorithm described in 4.2, we make the following modifications: It is unlikely that a clear majority of the nearest neighbors of a data bundle will all share the same error key because of the sparsity introduced by the large number of classes and the small size of the data set. Therefore, instead of majority vote to determine one definitive class, we output a list of all potential error keys ranked by the distance of the knowledge base instances to the data bundle, then cut off the list at k for initial presentation to the expert (see the schematic depiction in Fig. 7). A similar type of ranking categorization is already mentioned in [19]. Thus, we also address a weakness of standard kNN which becomes evident in Fig. 6 – the sensitivity to local data structures. For k = 6, the class assigned by majority vote is different from that for k = 15. Since our goal is to support the human expert, not fully automatic classification, we can avoid this inconsistency. Items lower on the list can be accessed by the quality expert in the user interface (cf. Fig. 4).

We determine the closeness of the data bundles by comparing them with respect to features derived from the text. For the domain-ignorant approach, we use all words in the text (*bag-of-words*), for the domain-specific approach, we choose mentions of parts and errors as features (*bag-of-concepts*). On average, a text has about 70 words, resulting in as many bag-of-word features. With the domain-specific approach, we detect on average 26 part/error mentions per text. We use the concept mentions as attributes without distinguishing between types of concepts (part or error). Annotating the text with the help of a domain-specific, synonym-

rich part-and-error taxonomy from prior research with the OEM [16] allows us to collapse mentions of the same part in different wordings into identical features. For example, „mud guard", „splashboard" and „fender" all belong to the same concept within the taxonomy and are all represented by the same concept ID. The taxonomy has about 1.800 / 1.900 distinct concepts in German and English, respectively. Thus, we can represent each unique combination of part ID, error key and concept mentions as a node in a knowledge base, which is derived in a first training step.

This also allows us to abstract from data instances to configuration instances, reducing the size of the knowledge base and allowing for faster data comparisons when calculating similarity measures. We thus address one of the weaknesses of the standard kNN approach in a way which is similar to the kNN Model algorithm in [7]. For the bag-of-words approach we accordingly store combinations of part ID, error key and individual words (excluding punctuation) in a knowledge base of the same structure.

We retrieve error code suggestions for data bundles from the knowledge structure by computing the similarity of potential nearest neighbors. In order to do this efficiently, we filter the knowledge nodes to first retrieve a neighbor *candidate set* fitting to the data bundle under consideration (Fig. 5). From the entire set of knowledge nodes (1), we first select the subset of nodes with the same part ID as the data bundle to be classified (2). From this subset, we select those knowledge nodes which share at least one concept mention with the data bundle under investigation (3) – or one word for the bag-of-words approach. This selection is made via the indexes of the knowledge structure. If the part ID is not found in the knowledge structure, we select all nodes into our neighbor candidate set.

Next, we compute a pairwise similarity score for each candidate node with reference to the current data bundle. We retrieve the error codes of the 25 best-scored candidate nodes. For each of these error codes, we assign an error code with associated score to the data bundle under investigation. These scored error codes are stored in a relational database and presented to the quality worker via the web app interface for final error code assignment.

To evaluate performance of the classification algorithm, we have experimented with two established similarity measures – the Jaccard similarity, computed as the number of shared attributes divided by the total number of attributes, and the overlap similarity, computed as the number of shared attributes divided by the size of the smaller attribute set.

Jaccard Similarity Coefficent: The similarity of two items (knowledge nodes) with feature sets A and B is represented by:

$$\frac{|A \cap B|}{|A \cup B|}$$

Overlap Similarity Coefficent: The similarity of two items (knowledge nodes) with feature sets A and B is represented by:

$$\frac{|A \cap B|}{min(|A|, |B|)}$$

4.4 Processing Pipeline

The classification step is embedded in a pipeline which includes linguistic preprocessing and the creation of a knowledge base. This processing pipeline is detailed in the following.

Figure 8 shows the entire analytics pipeline for the domain-specific approach. It can conceptually be split in two parts: one to extract structure from unstructured data, which includes data preparation, linguistic preprocessing and annotation with domain-specific knowledge, and one for processing the extracted structured data, which includes the building of a knowledge base and the classification step.

We assume the classical phase-oriented data mining process which differentiates a training phase from the subsequent test phase and the application phase.

In the *training phase*, we extract domain-specific classification features from within the unstructured data portion, following several steps:

1. Creating Data Bundles: read data from the database and combine related reports into one document.
2. Unstructured Data Analytics
   (a) Text Preprocessing: Tokenization and Language Recognition
   (b) Concept Annotation: mark up domain-specific concepts in the text (words describing parts and errors)
3. Structured Data Analytics
   (a) Knowledge Base Extraction: for each data bundle, extract into a "knowledge node" (cf. Fig. 9)...
      - the error code
      - the part number
      - the occurring concepts (numeric IDs)
   (b) Knowledge Base Persistence: store knowledge nodes in a relational database

After the knowledge base has been created, we exploit this knowledge for the classification step (cf. 4.3) in the *test and application phases*. Steps 1 - 2 of the process are identical to the training phase:

1. Creating Data Bundles: read data from the database and combine related reports into one text document.
2. Unstructured Data Analytics
   (a) Text Preprocessing: Tokenization and Language Recognition
   (b) Concept Annotation: mark up domain-specific concepts in the text (words describing parts and errors)
3. Structured Data Analytics
   (a) Candidate Set Generation: select knowledge nodes which share a minimum of 1 feature with the data bundle to be classified (cf. Fig. 5)
   (b) Classification of the data bundle (cf. 4.3)
   (c) Result Persistence: store scored error code suggestions in a relational database

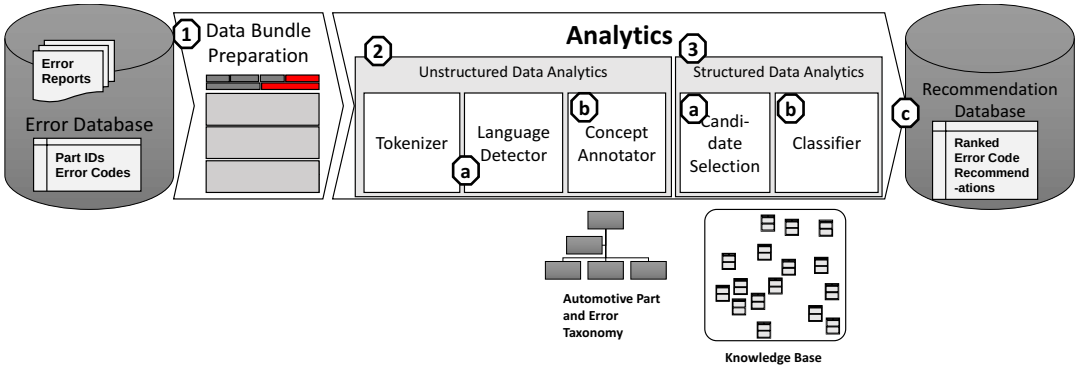

Figure 8: Detailed view of the domain-specific classification pipeline

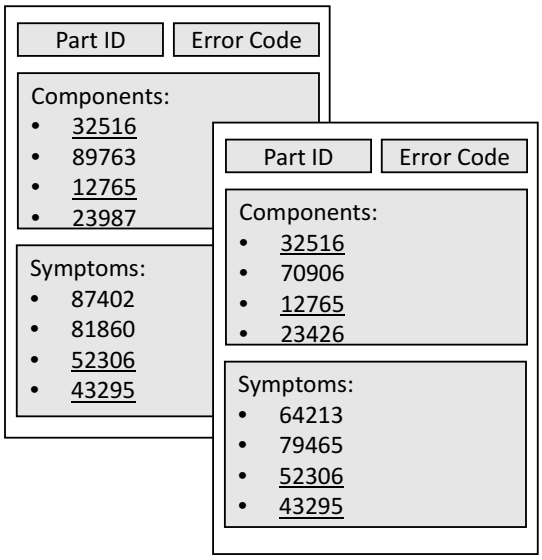

Figure 9: Two Knowledge Nodes with shared concepts underlined

The domain-ignorant approach proceeds accordingly but eliminates the concept annotation step and instead extracts all words of the document as features to be stored in the knowledge nodes.

It is obvious that this approach to a processing pipeline reflects a high degree of flexibility and extensibility for both preprocessing and classification steps. In our current setting we use the kNN-derived algorithm described in 4.3 for the classification step. This step is realized as an extension point where different classification algorithms can be plugged in easily.

4.5 Prototypical Implementation

In the following, we give a short overview of the technologies used in the prototypical implementation of the QATK framework and QUEST app.

4.5.1 Data Storage

For data storage, we use relational databases. We store raw data from the industrial source as well as from the NHTSA ODI source and the knowledge bases and classification results.

4.5.2 Text Analytics and Classification

In our implementation of the Quality Analytics Toolkit, we build on the Java version of the open-source Apache standard UIMA (Unstructured Information Management Architecture) [6], which enables us to easily build modular linguistic processing pipelines. These pipelines are composed of Analysis Engines containing annotators with single text analytics functionalities. Annotations on the text are recorded as typed feature structures with a start and end index relative to the document text in the Common Analysis Structure (CAS) which is handed over from one Analysis Engine to the next, such that annotators can build on findings from previous steps of analysis. In our case, one CAS contains one data bundle, including all available reports and text descriptions plus the part ID and error code.

We chose this framework for several reasons: It is open-source, extremely modular and well supported by the research community. High-quality implementations exist for a large number of standard natural language processing components, e.g. in the DKPro repository [21]. Functionality for quick and flexible pipeline building and testing is provided by the uimaFIT library [14].

The core of the QATK toolkit is a UIMA pipeline corresponding to the processing steps explained in 4.4 and depicted in Fig. 8. It reads the report and identifier data bundles from a database, segments the text into words using a simple custom whitespace-/punctuation-tokenizer, identifies occurrences of car part and problem synonyms in the text, builds a knowledge base from these identified concepts and uses the knowledge base to assign error code suggestions to previously unseen data bundles.

4.5.3 Domain-specificity: The Automotive Part and Error Taxonomy

A domain-specific resource used in our analytics module is a taxonomy of car parts and error symptoms, developed in prior research and originally used for an information extraction task on social media data [18, 9] . The taxonomy is stored in a custom XML format and has a shallow structure which is nevertheless well suited to the differentiations we want to make: It distinguishes components, symptoms, location and solutions. The error codes we want to recommend correspond to symptoms and also depend on components, which is why we choose to annotate the texts with occur-

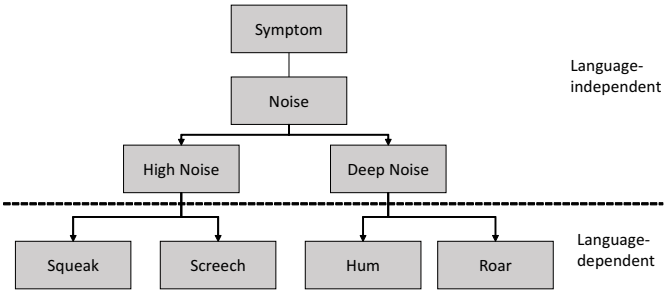

Figure 10: Automotive Taxonomy (graphic adapted from [18])

rences of components and symptoms from the taxonomy as features for our classification task. The taxonomy is multilingual – its upper category levels are language-independent with multilingual labels, its leaf categories are language-specific and contain synonyms of terms for the same concept (cf. Fig. 10).

QATK builds upon some closed-source legacy libraries for maintaining and using the taxonomy resource: An editor GUI for adding, changing and removing taxonomy concepts and concept features, as well as compiled Java archives which contain the classes needed for modifying and using the taxonomy. Among them are UIMA components for annotating occurrences of taxonomy concept words in text documents. These libraries do not entirely meet the requirements of the present use case. Therefore, some efforts had to be made in order to be able to use the taxonomy for text annotation.

We made a number of changes to the representation of the taxonomy and to the taxonomy annotator component which improve performance (cf. [12]): Annotation becomes faster, less memory-intensive, achieves higher coverage and is more accurate for multiwords. We represent the taxonomy as a trie data structure, a tree structure which allows for fast search and retrieval. Like the original approach, we expand the concepts of the taxonomy with synonyms of concept label substrings as found in the taxonomy itself.

Our optimized implementation has a left-bounded greedy longest-match approach for mapping text sequences to taxonomy concepts, eliminating concept matches which are completely enclosed by other concept matches. By applying multilingual annotation and correctly capturing multiwords, we achieve an overall higher recall of concepts than the annotator from the legacy code. For instance, the original taxonomy annotator does not recognize any taxonomy concepts in 2530 out of the 7500 data bundles, but the new annotator finds concepts in all of these.

4.5.4 User Interface

The QUEST web application partly reconstructs the user interface and functionality of the original quality engineering software which is used by the automotive OEM to record, maintain, and classify the data. In QUEST as in the original software, users can view the data and assign error codes. The core difference is that in the QUEST error code assignment interface, the user is first presented with a selection of the 10 most likely error codes in descending order of likelihood. If the user decides that the correct error code is not among these 10 codes, they can access the list of all error codes available for the part ID of the current data bundle, as is the default in the original software.

Also, users with extended rights can define new error codes right in the QUEST interface. Furthermore, all users can view the comparison of error code distributions between the OEM data set and the public US complaints database (cf. 5.4). The QUEST web app is compatible with most browsers and implements responsive design to be viewable on mobile devices. It is written in Java, uses PrimeFaces graphical components [1] and is deployed on a WSO2 web server [2].

5. EXPERIMENTS

In this section we present and discuss the results of two experiments into the feasibility of text-based classification with our data set. In 5.1 we establish the conditions for our experiments, in 5.2 we compare the performances of the domain-specific and the domain-ignorant modified classification algorithms with respect to a baseline, and in 5.3 we test the performance of classification on different report types (mechanic and supplier report). In 5.4, we describe the setup of an extended use case in the web app which is currently under development and evaluation.

5.1 Experiment Setup

We test the classification algorithm (cf. chapter 4.3) with different similarity measures and with different data abstraction models.

As a performance measure, we report accuracy defined as the percentage of test data which include the correct error code in the error code list at k <= 1, 5, 10, 15, 20 and 25, respectively.

Accuracy@k:
For D_k the set of data bundles where the correct error code is found within the first k suggestions, corresponding to the k nearest neighbors, and T the test set,

$$A@k = \frac{|D_k|}{|T|}$$

We run all experiments with stratified 5-fold cross-validation on the 6782 data bundles whose error code appears more than once in the data. This means that for each error code, we use 4/5 of the data bundles with this error code as input to the knowledge base and assign error codes to the remaining 1/5 using the knowledge base built from the rest of the data. We do this five times with distinct splits of the data and average the accuracies obtained in each iteration. The test data sets consist of 1250 data bundles on average.

We use two similarity measures, the Jaccard coefficent and the overlap measure (cf. 4.3). We work on whitespace- and punctuation-tokenized text without further preprocessing or normalization.

We compare the results of the classification to two baselines obtained without or with very little consideration of the text:

1. the *code frequency* baseline, where all error codes which are available in the database for the part ID of the data bundle under consideration are sorted by their frequency in this database, and the first k returned
2. the unsorted *candidate set* baseline (cf. 4.3), containing all nodes in the knowledge base which share the

part ID and at least one concept / word with the data bundle under consideration

The baselines themselves merit a look at their performance: The *candidate set* baseline depends on the applied variant of the algorithm, but all candidate set baselines have similar accuracy profiles. Accuracies are rather low throughout, with <1% accuracy for k = 1 and approximately linear development towards around 83 % accuracy for k = 25 (cf Fig. 11). This baseline could obviously not be used for automated recommendations of error codes.

The *code frequency* baseline performs better than the candidate set baseline, with an accuracy@1 of 35 %, accuracy@5 of 76 % and accuracy@10 of 88 %. At k = 25, it even has perfect accuracy of 100 %. Since we know that there are potentially over hundred error codes for one part ID, we assume that this is an artifact of our randomly selected data set. In any case, sorting available codes by their frequency can be a first step towards supporting quality workers in finding the correct error code more quickly.

5.2 Experiment 1 – Text-Based Error Code Prediction

In this experiment we establish whether error codes can be predicted at all on the basis of the text reports alone. We compare two variants of the classifier, a domain-ignorant bag-of-words model on the tokenized text, and a domain-specific bag-of-concept model created with the help of domain-specific text annotations from the automotive part and error taxonomy.

5.2.1 Results

The results of experiment 1 can be seen in Figure 11. We find that both the bag-of-words and the bag-of-concepts classifier outperform the baselines for k < 25 when using Jaccard similarity. The four variants we tested – bag-of-words and bag-of-concepts with each similarity measure respectively – differ considerably in their performance.

In general, overlap similarity performs worse than Jaccard, and the bag-of-concepts classifier does not perform significantly better than the code frequency baseline (in fact, slightly worse for k = 1) when combined with the overlap measure.

Combined with the Jaccard measure, the bag-of-concepts approach out-performs both baselines with accuracy@1 of 56 %, accuracy@5 of 85 %, and accuracy@10 of 92 %. For k of 15, 20 and 25, all approaches as well as the baseline deliver accuracies between 90 and 100 %.

For smaller k (1 and 5), the accuracy of the bag-of-words classifier is markedly better than that of the bag-of-concepts classifier regardless of similarity measure used, with accuracy@1 of 76 % (overlap) and 81 % (Jaccard), accuracy@5 of 93 % (overlap) and 94 % (Jaccard), respectively.

5.2.2 Discussion

Three out of the four text-based classification algorithm variants provide accuracies which out-perform the code frequency baseline, and all four could be used for recommending error codes on the basis of text reports. The bag-of-words model is currently providing better accuracies than the bag-of-concept model, especially for small k. This means that its ranking of the potential error codes more closely resembles the actual probabilities of error codes based on the content of the problem reports.

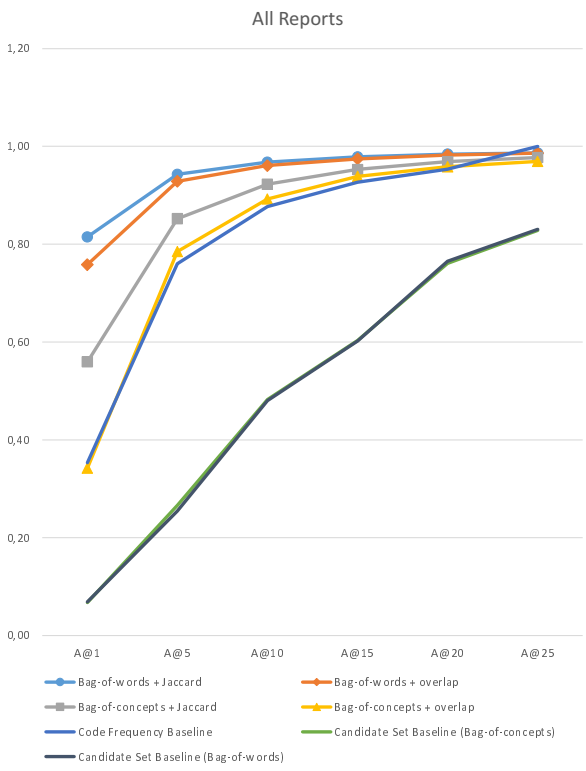

Figure 11: Results of experiment 1, with k on the x-axis and accuracy@k on the y-axis.

This tells us that the concepts which are currently being recognized using the automotive parts and error taxonomy do not represent ultimately accurate features for classification and are not the best option for recommending error codes to the quality worker. This is not altogether surprising, since the taxonomy was originally developed for a different task (information extraction, cf. [17]) and has not yet been adapted to the current data source. Adapting the taxonomy thus suggests itself as a next step.

However, the bag-of-words variant is not a feasible industrial solution because of time and memory needs due to the larger number of features per data bundle and the larger number of pairwise similarity computations to be made. On our small data set which includes only 3 out of hundreds of components, running the bag-of-words classifier takes about 11 minutes for one iteration of the five-fold cross-validation, classifying ca. 1250 data bundles, which means computation time per data bundle is at ca. 0.5 seconds. In contrast, running the bag-of-concepts classifier takes about three minutes for one iteration, which means computation time per data bundle is at ca. 0.14 seconds. When applying our processing pipeline to the entire data set with a larger number of data bundle to data bundle comparisons, it is important to keep the number of pairwise feature comparisons low. Removing German and English stopwords (articles and personal pronouns) as an additional step during the bag-of-words approach has no impact on the accuracy of classification, but shortens the runtime to ca. 7 minutes for one iteration and ca. 0.3 seconds per data bundle. This is still slower than the bag-of-concepts approach.

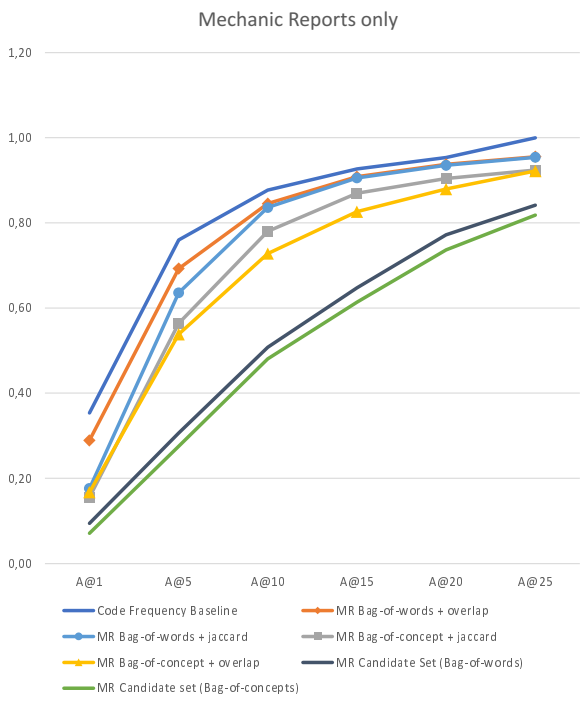

Figure 12: Classification results for features derived from the mechanic reports only, with k on the x-axis and accuracy@k on the y-axis.

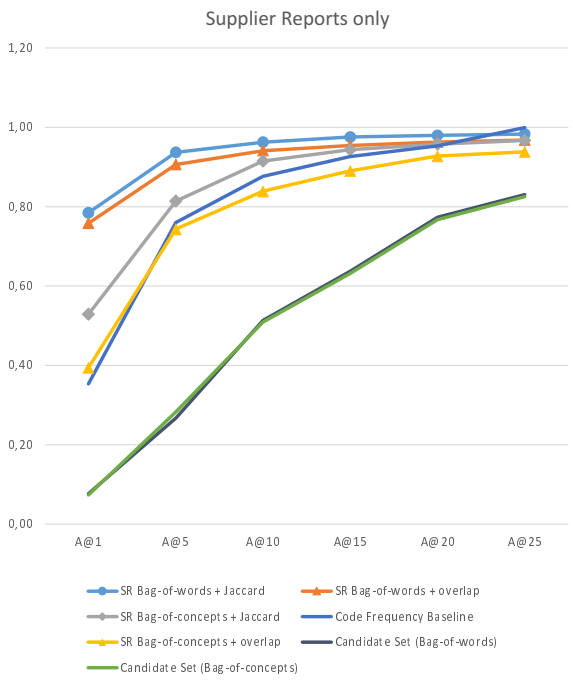

Figure 13: Classification results for features derived from the supplier reports only, with k on the x-axis and accuracy@k on the y-axis.

In contrast to the bag-of-concepts approach, the bag-of-words approach is also not suited for further domain-specific analysis steps. Improving the coverage of the taxonomy used for the bag-of-concepts approach is therefore a worthwhile avenue to pursue. An overview of the issue of taxonomy extension and an argument for re-using semantic resources across tasks is given in [12]. Investigations into methods to automate the extension of a domain-specific semantic resource are on-going.

There are thus two conclusions to be drawn from experiment 1: (1) the text reports can indeed be used to support quality workers by automatic recommendation of error codes, (2) to create a feasible industrial solution, an improved domain-specific resource is needed.

5.3 Experiment 2 – Point of Entry for Error Code Prediction

In the second experiment, we test how early it is possible to make a prediction about the error code of a report bundle. Recall that data about the nature of the problem from different sources – mechanic, OEM and supplier – accumulate over time during the quality evaluation process (cf. Fig. 2). The earliest report which reaches the OEM is the mechanic report, whereas the supplier report is added later. It would be beneficial if the error code could already be predicted on the basis of the mechanic report only.

Retaining the knowledge base models learned on all reports, we have therefore attempted classification with all variants of our adapted classification algorithm on test data bundles which included only one type of report, namely, the mechanic report or the supplier report.

5.3.1 *Results*

The results of experiment 2 can be seen in Figures 12 and 13. We find that the classifier performs very badly on test data which only include the mechanic report (cf. Fig. 12): All four variants of the algorithm have lower accuracies across the board than those provided by the code frequency baseline, with accuracy@1 between 16 and 29 % vs. the baseline's 35 %. Still, the bag-of-word models perform slightly better than the bag-of-concept models.

On test data which only include the supplier report, we observe accuracies which are nearly as good as those for the test data including mechanic report and supplier report (and in some cases an early OEM report): 78 % accuracy@1 for the bag-of-words model with Jaccard similarity, accuracies of > 90 % starting at $k = 5$ for the bag-of-words model and at $k = 10$ for the bag-of-concepts model, and a very close resemblance of accuracies between the bag-of-concepts with overlap similarity and the code frequency baseline (cf. Fig. 13).

5.3.2 *Discussion*

It is evident that the mechanic reports alone do not contain good features for predicting error codes with the adapted classifier, either as bag-of-words or as bag-of-concepts representations. In contrast, the supplier reports are a more reliable source of features. This is in accordance with observations about the information content and the data quality of the respective data sources: Mechanic reports tend to be poor in detail, focused on superficial problem description and often error-riddled, such that even human experts cannot draw conclusions about the detailed nature of the problem, whereas supplier reports tend to contain more

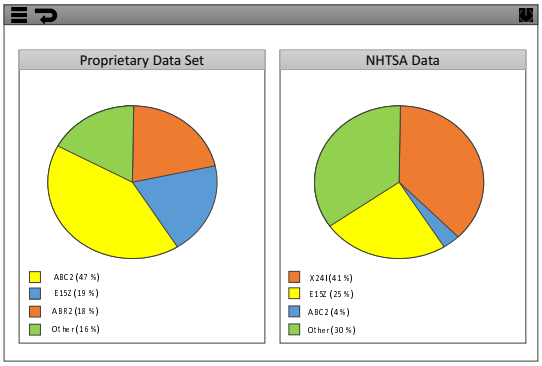

Figure 14: Data Comparison Screen in the QUEST web app, showing a side-by-side comparison of the top 3 error codes in two different databases.

detail and include descriptions of potential causes.

While we have to conclude that (1) an earlier entry point into automatic error code suggestion is not feasible, we also find that (2) even a comparatively simple text-based classification approach accurately reflects the amount of information which human experts can draw from the text sources.

5.4 Extending the Use Case – Error Distribution in Different Data Sources

As mentioned earlier, it is easy to exploit additional data sources using the knowledge bases created with the internal OEM error report data. This allows for an interesting extension of our use case. If we assign error codes from the schema we use to classify our own quality data to texts from a different data source, for instance one which covers complaints from a different market and includes reports about other manufacturers' vehicles, we can gain insights about where we stand in terms of product quality in contrast to the competitors. This is crucial business intelligence for staying competitive, e.g. by identifying brand-specific weaknesses or issues with shared suppliers.

Obviously, there will be substantial inaccuracies in the fully automatic classification of the public data source. In particular, the bag-of-words approach suffers in accuracy as soon as test and training data are different text types or in different languages, whereas the bag-of-concepts approach is in principle independent of the document language or other text features. However, an approximate impression of the distribution of similar errors can still be gained from the data. The usability and desirability of this functionality have been confirmed in conversation with the OEM.

In the QATK/QUEST implementation, we provide a mockup of this use case extension: We use our knowledge base to classify problem reports from the US-American complaints database maintained by the Office of Defects (ODI/NHTSA) [13]. In the web app, we have implemented the function for viewing side-by-side pie charts showing the distribution of the n most frequent error codes in both data sources (cf. Fig 14).

6. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we have presented a specific multi-class classification problem from a real-life industry context, namely, the filtering and recommendation of error codes for bundles of textual reports concerning damaged car parts. This problem is challenging because of the nature of the data, which are mostly unstructured text using domain-specific vocabulary, and because of the high number of classes.

We have tested alternative variants, domain-specific and domain-ignorant, of a custom classification algorithm derived from k-Nearest-Neighbors (kNN), and evaluated both against a baseline which ignores the text content as well as with respect to their industrial feasibility. We have shown the QATK/QUEST toolkit as a viable approach and presented a prototypical implementation.

In order to identify domain-specific terms in the text, we have used a legacy semantic resource originally designed for a different task. We have tested the validity of these domain-specific features with a custom classification algorithm adapted from k-Nearest-Neighbors. We have shown that the use of domain-specific knowledge leads to good accuracies of recommended classes, with up to 92% of correct classes recovered within the first 10 ranked recommendations. We have compared this domain-specific approach to a domain-ignorant bag-of-words approach and found that the domain-ignorant approach currently performs better, although it is not a viable solution in the industrial context due to performance and scalability properties.

Therefore, we plan the following extensions of our work:

- introducing more linguistic preprocessing – here we will profit from the modularity of the UIMA framework and of the QATK/QUEST toolkit
- enhancing the domain-specific taxonomy
- evaluating the extended use case and discovering more use case extensions
- evaluating the web UI in a field study with quality experts

Work on improving the coverage and maintainability of the domain-specific taxonomy is already in progress.

7. ACKNOWLEDGMENTS

Many thanks to our colleagues in the quality management and quality engineering departments for providing inspiration for the use case and ongoing discussions, as well as to our colleague C. Kiefer for important feedback and to the students of project „Mobile User-Driven Infrastructure for Factory IT Integration" for crucial implementation work. We also thank the Graduate School advanced Manufacturing Engineering (GSaME) for supporting the broader research context of this paper.

8. REFERENCES

[1] PrimeFaces JSF.
[2] WSO2 Server.
[3] M. Bank. *AIM-A Social Media Monitoring System for Quality Engineering*. PhD thesis, 2013.
[4] C. Fellbaum. *WordNet*. Blackwell Publishing Ltd, 1999.
[5] D. Ferrucci, E. Brown, J. Chu-Carroll, and J. Fan. Building Watson: An overview of the DeepQA project. *AI magazine*, pages 59–79, 2010.

[6] D. Ferrucci and A. Lally. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

[7] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer. Using kNN model for automatic text categorization. *Soft Computing*, 10(5):423–430, 2006.

[8] C. Hänig. *Unsupervised Natural Language Processing for Knowledge Extraction from Domain-specific Textual Resources*. PhD thesis, Universität Leipzig, 2012.

[9] C. Hänig and M. Schierle. Relationsextraktion aus Fachsprache - ein automatischer Ansatz für die industrielle Qualitätsanalyse. *eDITion*, 1(1):28 – 32, 2010.

[10] G. Ifrim, M. Theobald, and G. Weikum. Learning word-to-concept mappings for automatic text classification. *Learning in Web Search Workshop, ...*, 2005.

[11] L. Kassner, C. Gröger, B. Mitschang, and E. Westkämper. Product Life Cycle Analytics - Next Generation Data Analytics on Structured and Unstructured Data. In *CIRP ICME 2014*, volume 00. Elsevier Procedia, 2014.

[12] L. Kassner and C. Kiefer. Taxonomy Transfer: Adapting a Knowledge Representing Resource to new Domains and Tasks. In *Proceedings of the 16th European Conference on Knowledge Management*, 2015.

[13] NHTSA. NHTSA Data, 2014.

[14] P. V. Ogren and S. J. Bethard. Building test suites for UIMA components. *SETQA-NLP '09 Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, (June):1–4, 2009.

[15] P. Russom. BI Search and Text Analytics. *TDWI Best Practices Report*, 2007.

[16] M. Schierle. *Language Engineering for Information Extraction*. PhD thesis, Universität Leipzig, 2011.

[17] M. Schierle and D. Trabold. Extraction of Failure Graphs from Structured and Unstructured Data. *2008 Seventh International Conference on Machine Learning and Applications*, pages 324–330, 2008.

[18] M. Schierle and D. Trabold. Multilingual knowledge based concept recognition in textual data. In *Proceedings of the 32nd Annual Conference of the GfKl*, pages 1–10, 2008.

[19] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, Mar. 2002.

[20] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas. Short text classification in twitter to improve information filtering. *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '10*, page 841, 2010.

[21] TUDarmstadt. DKPro Toolkit, 2011.