Internship Report

Seminar für Sprachwissenschaft
Eberhard Karls Universität Tübingen

Author
Verena Blaschke
verena.blaschke@student.uni-tuebingen.de

Academic Supervisor
Dr. Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

Company
IBM Deutschland Research & Development GmbH

Supervisor
Dr. Peter Gerstl
gerstl@de.ibm.com

February 9th, 2018

1 Introduction

1.1 IBM

I was an intern at IBM Germany Research & Development from September 2017 to January 2018.

The International Business Machines Corporation (IBM) is an American technology company that produces hard- and software and provides consulting services, which has subsidiaries around the world. The IBM Germany Research & Development centre in Böblingen focuses on several areas of soft- and hardware research and development, including data analytics, cloud computing, mainframes and security technology.

1.2 Open Discovery Framework

I joined the Data Analytics department, which, among other things, produces software for the IBM InfoSphere[1] Information Server, a data integration platform that customers can use to monitor and analyze their data. The team that I joined is developing the Open Discovery Framework (ODF), a component of the IBM InfoSphere Information Server. The purpose of the ODF is to provide a platform for software that generates metadata for databases or parts thereof. Examples for such metadata are quality analyses and the assignment of labels. These connections between actual data and metadata are called *annotations*, and a program that creates them is called *discovery service*.

Annotations can be used for maintaining high data quality by ensuring that no data is corrupted, missing, or does not adhere to the standards. They can also be important for performing analyses on the data. To perform these tasks, it is necessary to easily access the data whose quality should be controlled or which should be used for analyses, within a single database and also across different databases. For example, if somebody has a database containing customer data and wants to use the information on the customers' ages saved in the database, they first need to identify the part of the database that contains the age-related information. If they have several databases, they need to repeat this step for every database. If the databases are large or if many of them exist, this is a very cumbersome task to do manually. This can be made easier with automatically generated annotations that identify connections between data and predefined metadata terms such as "age".

Discovery services that create annotations from database entries to predefined *business terms* are called *term-matching services*. The business terms are a set of terms that are expected to describe most of the data that a user of the ODF works with. Some of the discovery services are directly based on the contents of the database, while others use the metadata that is already associated with the data, such as column headers or table names. I mostly worked with services that create matches between column names and business terms. Mapping column names to business terms is not trivial: naming conventions may vary across databases;

[1]InfoSphere is a registered trademark of IBM in the United States.

column headers can be full words or phrases or abbreviations formed according to different rules; one concept may be expressed by several synonyms or hypo- and hypernyms, and the names may also appear in different languages (e.g. an international German company might have German column headers for one table but English ones for another).

When using the ODF to run discovery services, service pipelines are possible and encouraged. This means that it is not only possible to run several services in a direct sequence, but that services further down the pipeline can use the annotations generated by the earlier discovery services as input. Services that work with the output of other services can also be discovery services, or they can be so-called *supervisor services*, that is, programs that filter the annotations proposed by discovery services.

After one or more services are run in the ODF, the resulting annotations (or a subset thereof) are presented on the user interface. The user can then decide whether to keep or to discard each annotation.

At the moment, both the framework itself and several services that can be used within that framework are under development at IBM. In the future, the ODF shall be open source so developers from other corporations and academic institutions can create and use their own discovery services.

2 The Internship

2.1 A Simple Phonetic Matching Service

To get familiar with the ODF, I first wrote a discovery service that creates annotations by comparing column names and predefined business terms. These comparisons are done by means of the Soundex algorithm, a simple phonetic matching algorithm. After all non-alphabetic characters have been removed from the input sequence, this algorithm, as described in [Nat05], works as follows:

1. Save the first letter of the word.
2. Remove H and W.
3. Replace the following letters by digits:
    - B, F, P, V → 1
    - C, G, J, K, Q, S, X, Z → 2
    - D, T → 3
    - L → 4
    - M, N → 5
    - R → 6

4. If adjacent digits are identical, reduce them to a single digit.
5. Remove A, E, I, O, U, and Y.
6. If the original first letter was a vowel or H, W or Y, prefix it to the digit sequence. Otherwise, replace the first digit in the sequence by the original first letter.
7. Retain the first four characters of the sequence. If the sequence is shorter than that, append as many zeroes as needed to get a four-character code.

Table 1 shows some examples for the algorithm's output after its penultimate step.

Table 1: Soundex codes after step 6

POSTAL CODE	P23423	LICENCE	L252
POST CODE	P2323	LICENSE	L252
PIN CODE	P523	LCNS	L252
ZIP CODE	Z124	LICN	L25
ZIP	Z1	LIC	L2

I added a few changes that can be applied when using the Soundex algorithm within the discovery service:

Firstly, the user can set the sequence length that is used in the last step.

Secondly, if a column name or business term is a multi-word genitive construction of the form "X of Y", then the service creates Soundex encodings for both the original sequence and the switched version, "Y X" (e.g. given "date of birth", it also generates the code for "birth date"). Whichever version is more similar to the phonetic code of the other term is used for the discovery service's comparison.

The phonetic matching service compares two sequences using the Levenshtein distance between them, normalized by the sequence length. If the complement of the normalized Levenshtein distance is above a certain threshold (that can also be set by the user when starting the service), an annotation is created for the column, mapping it to the business term. Apart from references to the column and business term, the annotation also contains information on the service itself (the service name, the values for the above-mentioned parameters that can be set by the user) and on the match (the annotation's *confidence value*, which is the complement of the normalized Levenshtein distance between the phonetic codes).

The algorithm used by the phonetic matching service is fairly simplistic and it is optimized for English input, both in terms of the Soundex algorithm itself and the preprocessing of genitive constructions. However, the purpose of writing this service was to familiarize myself with the ODF.

2.2 Decision Tree Supervisor Service

As mentioned above, it is possible to build a pipeline of multiple discovery services followed by a supervisor service that filters their output. The discovery services can be different services or they can be a single discovery service that was started with different configuration parameters, or a mixture of both. The goal of the supervisor service is then to process each of the previous services' output annotations and to decide which of them should be shown to the users and ultimately potentially persisted alongside the database, and which should be discarded instead.

My main task during the internship was to write a machine-learning based supervisor service that uses knowledge about previously generated annotations and whether they were accepted or declined by the user to learn which kinds of annotations should be shown in order to enhance the user experience.

The service takes as input term-matching annotations created by discovery services. These annotations contain both service-independent and service-specific information. Service-independent information is independent in such that all term-matching services have to include it in the annotations that they create. This information includes a reference to the datatable asset (that can be used to retrieve the asset's name as well as hierarchy information (e.g. about the table that a column belongs to)), the business term, the service name, and the confidence value that a service assigned the annotation. Service-specific information on the other hand directly depends on the properties that can be set when starting a service (e.g. for the phonetic matching service, this would include the sequence length and the confidence threshold) or on additional algorithm-specific information (e.g. the annotations created by the phonetic matching service could include information on whether the preprocessing of genitive constructions was applied).

In order to train a machine-learning classifier, the annotations in the training set need labels. The possible labels are "keep" and "discard" and indicate, respectively, whether an annotation describes a good match that should be presented to the user or whether it should be discarded. Since no gold-standard data (actual annotations with labels based on whether users of the ODF accepted them in the user interface or discarded them) exists, I had to manually create and label annotations for training the tree.

The machine learning method is a decision tree. The benefit of using this method is that decision tree learning is a white-box machine learning technique–the generated tree can easily be transformed into a human-readable format. That way, specific decisions can be motivated, and looking at the entire tree can give insight into the (training) data, such as showing which services always work well/badly or for which services more information is needed to classify their annotations properly.

The construction of a decision tree from the training data is similar to the ID3 [Qui86] and C4.5 [WKQ$^+$08] algorithms and works as follows:

1. Transform the annotations into sets of feature-value pairs. (Hereafter, each such set is called a *data point*.)

    (a) Using the column name and the business term, create additional features, such as the length ratio between the column name and the business term.

2. Create questions based on the feature-value pairs.

    (a) If a feature has categorical values, generate identity questions (i.e. "Does *feature* have the value *value*?"). Generate only questions if the set of possible values is substantially smaller than the set of training annotations. This restriction ensures that the questions are not too specific (meaning they would overfit to the training data), and it can shorten the time it takes to perform step 3.

    (b) If a feature has numerical values, determine several ranges that the values fall into and generate questions about them (i.e. "Does *feature* have a value smaller than *number*?").

3. For each question, temporarily split the set of data points according to it, yielding the subset of the data points for which the question can be answered with "yes", and the subset of the data points where the answer would be "no" or where the question does not apply. Compute the label distributions of these resulting sets and the information gain (i.e. normalized entropy reduction) that the split would cause.

    (a) If any splits yield non-zero information gain, split the data set according to the question with the highest information gain. Create a tree node that contains information on the question. Add two child nodes to the new question node: one for each post-split set. Make each child node a question node or leaf node by repeating step 3 for the set of data points it describes.

    (b) If none of the questions yield any information gain, create a leaf node and assign it a label, namely the majority label of the instances it contains, and a confidence value, i.e. the proportion of the majority label instances within the leaf node.

Figure 1 shows a simple example decision tree.

The following properties can be set prior to starting the supervisor service:

- The minimum number of training data instances that a leaf node needs to contain. If a leaf node contains only very few data points, it likely overfits to the training data.
- Whether or not to collapse sibling leaf nodes with identical labels (but different confidence values). The advantage of doing so is that the size of the decision tree can be reduced without losing information with respect to the labels that the tree predicts.
- Rules. The decision tree can be initialized with rules written by the user that correspond to labels for specific features or feature combinations (e.g. "if *column name* begins with *some_prefix* and *service name* equals *my_service* then the label is *keep*").
    - When the tree has been built based on the rules, it can be queried for information on whether the rules describe the training data well. That is, if a rule leads to a leaf node where the majority of the data points have a label different from the label prescribed by the rule, the tree would mark this rule as "unhelpful".

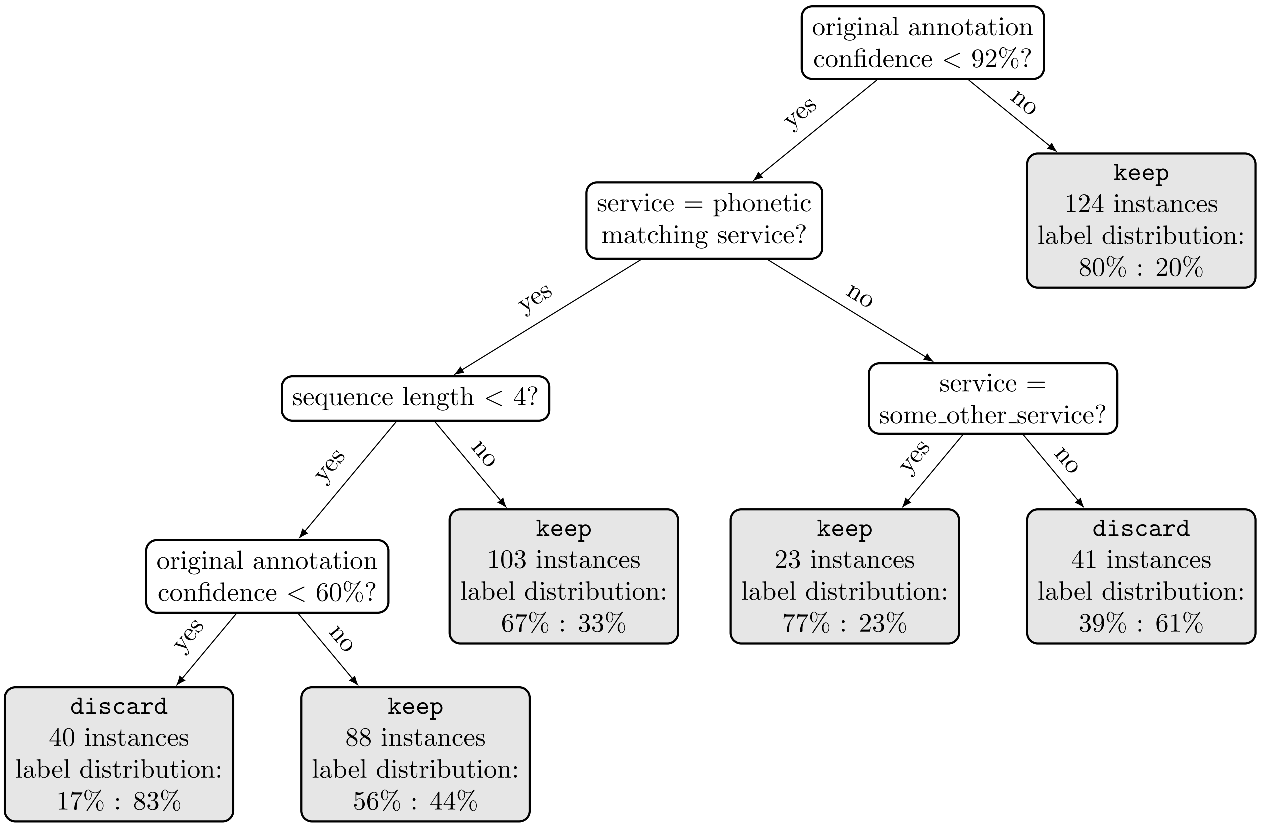

Figure 1: An example decision tree. The leaf nodes contain information on the training set instances they describe (number and label distribution (proportion of instances labelled "keep" : proportion of instances labelled "discard")).

Once the decision tree has been built, it can be queried with any new annotation that has been transformed according to step 1. If the annotation ends up in a leaf node with the "keep" label, a new annotation is created that includes all of the information that the original annotation contained, as well as information on the decision tree (the confidence value of the leaf node, the path through the decision tree leading to the leaf node, and the additional properties that can be set before building the tree from the training set).

I implemented the entire decision tree algorithm from scratch in Java[2] 7. In order to provide the additional functionalities, using already existing packages was not possible. After implementing the algorithm as a stand-alone project, I adapted it as a supervisor service that can be run as part of a pipeline in the ODF. At the moment, the decision tree supervisor service is a project that is separate from the development of release versions of the ODF, but in the future, it might be (the basis for) a supervisor service that belongs to the standard services that come with the ODF.

[2]Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

3 Conclusion and Acknowledgements

During my internship, I had the opportunity to plan, research, implement and discuss a big project. Pursuing this project helped me to deepen my knowledge of Java, Maven and Git, but it also gave me insight into software development and everyday work in a corporate context.

I would like to thank Peter Gerstl for his support and helpful ideas, which were indispensable for realizing this project. I would also like to thank all of my colleagues for the friendly environment that they welcomed me in.

References

[Nat05] National Archives and Records Administration. The Soundex Indexing System. `https://www.archives.gov/research/census/soundex.html`, 2005. Last updated 2007-05-30. Last accessed 2018-01-20.

[Qui86] J. Ross Quinlan. Induction of Decision Trees. *Machine learning*, 1(1):81–106, 1986.

[WKQ+08] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.