Hello, my name is Daoping. I have been working as a master student at Mercateo München since September 2019, and I am very happy and proud to talk about my master thesis today.

I want to welcome everyone who is remotely joining the talk today, and thank you for your interest and participation, especially considering the current coronavirus situation.

The topic today is about database cleaning, or more precisely, it is about a novel approach of anomaly detection in databases that uses probabilistic methods and vector representations, also called embeddings, of categorical values.

I will start by giving a brief introduction into the topic to give you some background and motivation.

In the second part, I will talk about how to examine and to model unclean databases from a probabilistic perspective.

In the third section, I will introduce the definition of embeddings and the intuition behind their usage in our scenario. Finally, I will show you the experimental setup and results, and I can already tell you that the results are pretty good, so the approach is making sense to some extent.

Nowadays, the potential value within data is really getting recognized. Data acts as an important enabler for advanced analytic techniques and automation applications like automated driving or image recognition.

More and more people are getting interested in developing machine-learning based algorithms, which rely on both the quantity and quality of data. If you don’t have enough data of high quality, it is very hard for you to develop and verify your new cutting-edge models.

Therefore, it is in general of high priority to obtain accurate, consistent and complete data, and also the ground truth information about it, for example, the distribution of the values, the knowledge on dependencies between tables and between columns and so on.

However, in real-world situations, these requirements oftentimes are not fulfilled.

So we first have to know, how real-world data looks like?

Here, we have a snippet of article data from our own database.

First of all, I want to emphasize the fact that I am using this data as an example to explain concepts and to give intuitions, so some statements on **this particular** data snippet I am going to make, might deviate from the truth of these specific data entries or from your expert knowledge. So please forgive me if you find any of the examples to be fishy. Keep in mind that they are just examples.

So here, each data tuple is comprised of attributes like different ids, the manufacturer of the article, the keyword that tells us what the article is, the price we pay etc.

The first attribute set\_id is particularly interesting. It is generated by our dooble algorithm and serves as an identifier that tells us, which specific product an article should be. That is to say, the articles that have the same set\_id should be the same specific product.

If we look closer at the data, we can see that there are some problems.

The first three data tuples have the same set\_id and unit, so they should be the same product sold in the same amount, but the first article is way too expensive comparing to the others.

It might be correct and intentional by the supplier, but it also could be that it is not the same product and the set\_id is wrong. What we have here is an outlier that might need further human inspection.

Moreover, we can have missing values in our database, which can be trivially detected but hard to be repaired.

We can have articles with different manufacturers but the same set\_id. In this case, we want to know whether the set\_id or the manufacturer is wrong, and then to determine the article that contains wrong information.

I don’t want to enumerate all possible flaws in a database, and I think you already get the idea and also realize that we need to address this problem in some way.

So basically, the goal is to detect these potential flaws **in an automated fashion,** because if we do this by hand, it would take countless human hours for the entire database, which contains millions of articles.

Besides manual detection, there exists a collection of conventional techniques, which already do the job automatically.

For instance, we can use heuristics, which, simply put, are daisy-chained if else statements. We can use them to write a discriminative script that determines whether there is any conflict or violation in a tuple.

The first problem with this method is that, as data is constantly evolving and growing, we often need to update old heuristics and add in new heuristics, which would eventually become painful and not maintainable. The second problem is that there could be heuristics that conflict with each other, so you have to think about prioritization.

Also, you need very complicated and domain-specific heuristics to answer questions like, is it Knipex or Primium that is the true manufacturer of set\_id blablabla.

Another way to do anomaly detection is to use statistical methods that reason about correctness by looking at the occurrences of attribute values. For example, if the manufacturer Knipex is observed 10 times more often than Primium, then there is a high chance that Primium is anomalous.

The problem with this approach is that the column attributes of our database have high cardinalities. In our context, the term cardinality describes the number of distinct values that can appear, while the term high-cardinality means that there is a huge number of distinct values, which occur sparsely in the observed data.

Simply put, we do not have enough data cor enough evidence to compensate for the large number of distinct values, so we cannot reason about correctness solely based on occurrences.

To find a solution to solve the anomaly detection problem under difficult real-world conditions, I tried to state the question: How does a human seek errors in a database? Can we come up with a technique that implements human intuition to perform the task?

If we go back to the data tuples from previous slides, and think closer about, how and why a human would suspect the first article to be an outlier. Well, its because we have a lot of external knowledge on the relationships between the attributes.

We know that similar products should have similar prices, we know that articles with the same set\_id **should** be the same particular product and thus similar prices. And sometimes we have information that is more concrete, just like “Products of Knipex are generally cheaper than Wera”.

The way a human performs anomaly detection is to accumulate such domain knowledge, and to evaluate a confidence score for each observed data tuple.

So the main idea of my approach, is to convert such domain knowledge into probability distributions, and to calculate confidence scores for data tuples by evaluating their joint probabilities.

The process of accumulating domain knowledge is basically the process of building a graphical model. The nodes of a graphical model are random variables, which in our context are the column attributes, and the connections between these nodes represent conditional probability distributions that are based on domain knowledge.

The graphical model I use to model dirty databases is a Bayesian network, because it is one of the simplest models available and seems suitable for our problem setup. A Bayesian network is directed and acyclic, meaning that there are no loops in the graph. And most importantly, it allows us to implement the joint distribution of the random variables with the product of conditional distributions and prior distributions.

On the slide, you can see that we have some tables providing prior information and conditional probabilities that are already calculated. Although this graph is very much simplified, I think that you can catch the basic idea and realize by yourself that the confidence score of a tuple is, in this case, the product of the corresponding probability values.

Let’s have a look at how exactly domain knowledge is translated to conditional distributions.

For example, we know that similar products have similar prices, and prices might be normally distributed around some average.

To model this knowledge, we can use a normal distribution and evaluate the parameters, which are the mean and variance, from the observed database.

The conditional distribution in this case would be the price distribution conditioned on keyword and manufacturer, and because price is a continuous value, this conditional distribution is a continuous distribution.

Given e.g. keyword equals Wapuzange and manufacturer equals Knipex, the price is then considered to be normally distributed around the average price of all “Knipex Wapuzange” articles with a certain observed variance.

And since we are only **extracting information from** the observed database, without involving extra external information, like e.g. the true price of “Knipex Wapuzangen”, we can re-use this probabilistic modeling procedure for other relationships and dependencies.

There is another situation where the conditional distribution is discrete. Consider e.g. the knowledge “Different manufacturers make different products”, both attributes are not numerical, but categorical.

In this case, we can use the categorical distribution, which is a discrete distribution that specifies the probability of each possible value separately.

For example, for a given keyword, we want to model the distribution of manufacturers and consider a categorical distribution. The probability of a certain manufacturer is proportional to the number of observed occurrences of that manufacturer together with the given keyword.

That is, we want to find all manufacturers that occurred together with the given keyword, count their occurrences, and normalize the count in order to make sure that the resulting probabilities sum up to one.

Therefore, if we write this down formally, the distribution of manufacturers, given keyword Kabelschere, is a categorical distribution, where each manufacturer has a probability that is calculated from the occurrence statistics.

Now we know how to convert external knowledge into probabilistic expressions and how to chain these expressions to build a Bayesian network.

From the programming point of view, we are interested in tools and programming paradigms, that facilitate the process of specifying probabilistic models and performing various probabilistic operations, such as calculating marginal probabilities or performing inference etc.

So I decided to try out the probabilistic programming paradigm, which has become quite a buzzword recently. It provides us the basic ingredients and building blocks to build a probabilistic model, and it has functions that we can directly use to calculate the marginal probabilities for our data entries.

Also, since there doesn’t yet exist a solution for database modeling and anomaly detection based on probabilistic programming, doing this would be quite experimental, but valuable and interesting.

There are many different probabilistic programming toolboxes that can be used for our purpose. In the current implementation, I was using a new Julia toolbox from the MIT called Gen.

The scripts I implemented with Gen are basically generative functions that resemble a Bayesian network by simulating the data generation process.

This means that, the process of accumulating domain knowledge and encoding it into the scripts is the process of building a simulator, that is able to generate new, correct data entries by following the encoded domain knowledge.

And since this simulator knows what is the way to generate correct data, it also has the ability to detect data that is unlikely to be generated, which is exactly what we are looking for.

To sum up this section, what we have achieved now is a generative function, that resembles the Bayesian network we have specified, and evaluates the marginal probabilities, which are also confidence scores, for the observed database.

For our articles, since each of them has a confidence score now, we can sort them after the score and examine the articles with the lowest scores.

To this point, it seems like that we have already reached our goal. But no, this probabilistic approach **alone** would not, and in fact, did not work. Let me show you why it didn’t work and what is my solution to the problem.

First, lets trace back to the example data, and consider the following conditional distribution of price given keyword equals Ratschen-Kabelschere. Since it is a continuous distribution, we are interested in evaluating the mean and variance from the data.

However, as you can see here, we only have a single article that has the keyword Ratschen-Kabelschere.

Meaning that we do not have enough evidence, or enough observation of Ratschen-Kabelschere, to evaluate a credible mean and variance.

And this is not a problem that is only related to the keywords, but it is also related to other column attributes like manufacturer, set\_id and so on.

If we cannot solve this problem, which is referred to as the high-cardinality problem I mentioned earlier, the probabilistic model cannot succeed because it relies on distributions with correct and reasonable parameters.

If we look closer, we can notice that there are some other articles with similar keywords. So maybe, if we count them in, and evaluate the mean and variance from this small neighborhood, we can get some mean and variance that are more plausible than without this neighborhood information.

This is quite a clever idea, but in fact, finding such neighborhoods is a very hard task, even for human.

In this example, you can identify that Ratschenkabelschere and Ratschen-Kabelschere are the same thing, but what about Wapuzange and Eck-Rohrzange? And, if we consider set\_id, there is no easy way, even for human, to do this similarity measuring task.

The question now becomes, can we measure the similarities between categorical values, by only using the information from the observed data itself.

After countless discussions and brainstorming sessions, the idea of using embeddings emerged.

Embeddings are basically high-dimensional vectors that represent discrete values, and they are mainly used in the area of natural language processing.

There, people create word embeddings, such that words that share similar contexts are located close to each other in the high-dimensional space.

If we adopt this idea and apply it to our use case, we could learn embeddings for e.g. keywords, such that similar keywords are located close to each other.

Then, we can establish for each categorical value, in this case a keyword, a neighborhood of arbitrary size, depending on how sparsely that keyword occurred in the observation.

For the conditional distribution of price given keyword equals Ratschen-Kabelschere, we find out that Kabelschneider and Ratschenkabelschere are the closest neighbors in the embedding space, so we can evaluate the necessary parameters from this neighborhood.

Now, we need a technique to learn such meaningful embeddings, so we look, again, to the area of natural language processing and see how they do the job.

A very popular category of embedding training models is called Word2vec. These models are neural networks with a specific embedding layer, which contains trainable embeddings.

So basically, a word2vec model uses a large amount of text sequences, and for each text sequence, the model tries to predict a word in the sequence, given the rest of the words in that sequence.

For example, we have the sentence “brown fox jump over dog”, the model takes as input “brown fox over dog”, which are the context words, and tries to predict “jump”, which is the target word.

Since at the beginning, the embeddings are randomly initialized and the model knows nothing about these words, it would make prediction mistakes.

These mistakes are propagated back to the trainable embeddings, so after each training step, the model gets a little bit better.

And after the model has learnt millions of sentences, we extract the trained embeddings, and find out that they are distributed in such a way that the semantic relationships between the original words are retained.

Unfortunately, we are not particularly looking for word embeddings, but rather embeddings for categorical values. And we don’t have sentences, but data tuples of tabular form. So we cannot directly apply the word2vec models to our scenario.

In order to adapt the word2vec model for our need, we need to look at the similarities and differences between text and tabular data.

One important similarity is that both sentence and data tuple are value sequences with contextual information. This information is so rich, that you can use it to map individual values into high-dimensional space in a meaningful fashion, but besides this contextual information, we don’t really have another reliable source of knowledge to do the mapping.

Regarding the differences, there are plenty of things that we have to consider.

While text sequences can be of variable length, the length of a tabular data tuple is fixed, meaning that we don’t have to specify the input size for our embedding training model.

Secondly, the word2vec model considers a **single vocabulary**, which contains every word that the model sees. For tabular data, we don’t have a single vocab, but a clear structure of attributes that have different domains.

For example, we know and we are pretty sure that a set\_id value is a set\_id, and not a keyword, so we can construct vocabularies separately to leverage this attribute knowledge, instead of mixing all categorical values together into a single vocab.

And, more importantly, in our database, we have numerical values, like prices, which don’t necessarily need to be mapped into the embedding space, but they must be incorporated during the embedding training process.

Imagine that, if we neglect numerical values, we are throwing a lot of essential information away, and we don’t want to do this.

In word2vec, people don’t have this problem, and they treat everything as words. So for us, we have to come up with some different model structure that is able to incorporate numbers.

So, what I decided to do, is to restate the prediction problem into a binary classification problem.

Why? Because, since we have multiple vocabularies, if we want to keep the original word2vec procedure, more complication must be added into the model.

And the result would be that, in each training step, the model has to figure out, which vocabularies the target and context words belong to, and updates the embeddings accordingly, which, believe me, is very complicated.

So instead of asking the model to predict a target value, we ask our model to tell us, whether the input data tuple, originates from our article database or not.

This means that, the model should output 1, for all article data entries from our database.

Ok, the structure seems to be complicated, but it really isnt.

Simply put, the model minimizes the distances between embeddings of those categorical values, which occurred together in the same context in the observed database.

The numerical values are participating in the training process, by being concatenated to the intermediate output vector, which is the input of the final layer, that does the binary classification job.

This is illustrated at the bottom of the diagram.

This model is in fact able to produce meaningful embeddings, and improve our probabilistic approach.

So we now come to the evaluation measures and results.

There are two things we want to evaluate. The first thing is the effectiveness of using probabilistic modeling for anomaly detection. The second thing is to check, whether using embeddings, to establish these small neighborhoods, gives us any advantage.

But, again, there is a small problem.

Since we are dealing with our article database, we don’t have the information on, which articles are anomalous, and which are correct.

So if we ask the probabilistic model to give us e.g. the top 50 articles with lowest confidence score, to verify these 50 articles is, by itself, a painstaking and time-consuming task.

So what I decided to do, is to regard the already existing articles as clean and normal, and based on these normal articles, I manually generate wrong article data, consisting of value combinations that have never been observed in the database.

Then I mix these manually generated anomalies with the normal data, build a probabilistic model for it, and train embeddings using this mixed data.

Then, we let the model evaluate confidence scores, and check whether those manually generated articles have lower confidence scores than the real articles.

The exact data I used for evaluation is the top 200 sets of articles from our database. There are in total around 9000 real articles, and 450 manually generated anomalies.

The Bayesian network I built, is this one on the slide. Here I assumed, that the price is dependent on set\_id, catalog\_id and manufacturer, and there are some interacting dependencies between these attributes.

About evaluation measures, we use precision and recall to evaluate and to compare models regarding their anomaly detection performance.

Let’s say you decide to identify 10 anomalies, the precision tells you, how many of these 10 cases are correctly identified, and the recall tells you, how many of the total anomalies you have detected, and how many are still missing.

In our scenario, what we exactly do, is that after calculating the scores, we sort the articles after these scores, and mark those articles with low scores as anomalous.

The question now is, how many articles should be marked as anomalous.

What we could do, is that we start with marking one article as anomalous, calculate precision and recall, and then, we increase the number to two, and calculate precision and recall again. Then we increase the number again, until all manually generated anomalies are covered and identified.

The precision and recall values, that are calculated along this process, can be drawn into a coordinate system, and become a precision-recall curve, that gives us crucial insights into the model’s performance.

We essentially want precision to stay high, while recall increases. So we want the curve to have large y-values, as long as possible.

Here we come to the first results, where the green curve is the pr-curve of the probabilistic model without using embeddings, and the blue curve is the pr-curve with embeddings.

First, we can see that the probabilistic model without embeddings performed better than a random classifier, as it is able to distinguish between manual anomalies and real articles, but the performance is bad.

When we let embeddings to join the process, we can see a huge performance boost. With embeddings, the model made no detection mistakes for the first 50 detections.

Then, it starts to become a bit confused, and starts to misclassify real articles as anomalies.

But, we need to remember that, our article database itself contains outliers and anomalies, and it is definitely possible that, some of the existing outliers deviate more from our knowledge and observed statistics, that the anomalies that we manually generated.

If we compare the recall at the time point, where we condemn the top 452 articles, which is the exact number of manually generated outliers, the model with embeddings successfully detected 54 percent of total outliers, and the model without embeddings 29 percent. Which is also a sign that embeddings are very beneficial.

I was also interested in the impact of domain knowledge on performance, so I decided to build a minimal Bayesian network.

And this reduced graph only says, that the price is dependent on other attributes, without any further information on the dependencies between those attributes.

Intuitively, this kind of knowledge reduction would have a great negative impact on performance, which can be observed in the right-hand side precision recall curve.

It shows that, the probabilistic model without embeddings has become a random classifier.

On the left-hand side, it is interesting to see that, although the precision drastically decreases, the performance is still OK, at least much better than without embeddings.

This shows that, using embeddings can compensate for a large amount of information loss, meaning that they are particularly useful, for databases, where we don’t have much domain knowledge to benefit from.

Coming to the conclusion, my work was able to create meaningful embeddings, by using only the dirty database itself.

I was able to show that embeddings are indeed helpful, considering the performance increase, and the fact that embeddings compensate for the limitations on the side of domain knowledge.

And, also very importantly, I think that anomaly detection with probabilistic modeling is definitely a feasible approach.

For future work, there are many things that can be done and improved.

For instance, we shall further examine the neural network model, to get a better understanding on its working mechanism.

Also, we shall conduct experiments with larger datasets, which I didn’t accomplish, mainly due to time constraints.

And also, there is a huge potential to extend this probabilistic model, and to think about how the detected anomalies can be automatically repaired.