Invoice Matching with Level of Confidence

Master thesis Computing science

Yasmin van Dijk

February 24, 2023

Contents

1 Problem description

1.1 Purchase-to-pay

The purchase-to-pay process starts when an employee or department of a company needs goods or services from a different company. Several documents within this process are of importance, including purchase orders, invoices and goods receipts [33]. Figure 1 gives an overview of the documents that appear in the process. Depending on the company structure, creating a purchase request can be a requirement. This document lists all needed goods and services, and their quantities. If the purchase request is approved, a purchase order (PO) is created and sent to the supplier, the company that delivers the goods and services. In cases where a purchase request is not required a PO is made and sent to the supplier directly. The PO lists all goods and services that are needed from that supplier, with their quantities and prices.

When goods are delivered they should contain a packing slip, listing the delivered goods and their quantities. The receiving company needs to check whether all items listed on the packing slip where actually delivered in the listed quantities, as well as check whether any of the goods have been damaged and should be returned. The delivered goods and their quantities are recorded on a goods receipt (GR).

In order to get paid, the supplier sends an invoice to the buying company. An invoice lists the delivered goods and services, and their quantities and prices. Additional costs such as freight and administration costs can also be listed on the invoice. Whenever a company receives an invoice, they need to check whether the goods and services where actually ordered and delivered as well as check if it is not a duplicate invoice [32]. They also need to check whether the quantities and prices are correct, and whether agreed discounts have been applied.

Checking whether invoiced goods and services match up with ordered goods and services can be done by comparing the invoice with the corresponding PO, also called 2-way matching. By comparing the GR with the PO, the company can check whether the ordered goods were delivered. Comparing the invoice, PO and goods receipt is called 3-way matching. If everything is correct the invoice can be paid, which is the final step in the purchase-to-pay process.

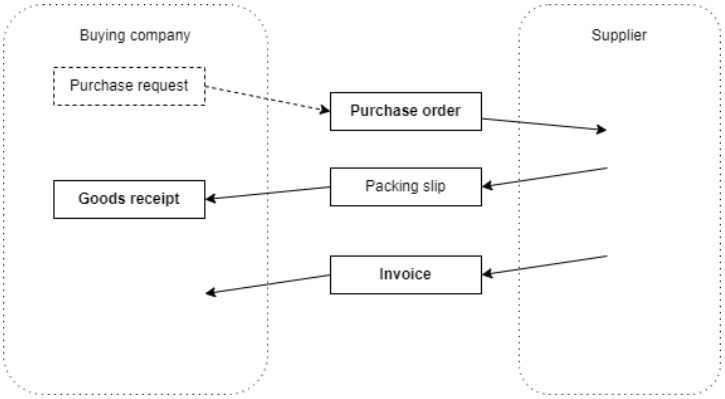

Figure 1: important documents in the purchase-to-pay process

1.2 3-way matching

An important step in the purchase-to-pay process is 3-way matching, as it helps companies in determining which invoices are correct and should be paid. To perform 3-way matching goods receipts should be compared with the corresponding PO's to determine if all ordered goods have been delivered. Invoices should be compared to PO's to determine if the invoiced goods match up with the ordered goods. If these three documents match up, the invoice can be paid.

The PO usually contains a PO number that is used to identify it. The supplier is expected to include the PO number on the invoice for the corresponding order. When a company receives an invoice, they can then use the PO number on the invoice to find the PO they need to compare that invoice with.

Documents can be compared on header-level or on line-level. For header-level matching, data in the header of the documents is compared. Data on the lines on the documents is compared for line-level matching. PO's and GR's are matched on line-level. The lines are compared on item numbers, item descriptions and quantities. PO's and invoices can be matched on header-level or line-level. For header-level matching the PO-number, supplier name, administration and total price are compared. For line-level matching the lines are compared on item number, item description, quantity and price.

Several challenges can occur when performing 3-way matching. To compare an invoice with the corresponding PO, this PO should be found first. Suppliers are expected to include a PO number on the invoice, which is used

as an identifier for the PO. Sometimes this number is missing. In those cases, other fields can be used to find the PO, such as supplier name, administration and total price. However, these fields do not necessarily contain unique values. When using such fields to find the PO, it is likely that multiple PO's are found and will have to be compared to the invoice.

Invoices can be sent over the internet in machine readable formats such as XML, also known as electronic invoices or e-invoices. Invoices can also be sent as paper documents through the mail or through e-mail in formats such as PDF. Because such documents are not machine readable, information from the invoice can not be entered directly into accounting software. Instead, either manual entry or optical character recognition software (OCR) can be used. Both methods can lead to errors. Typing errors can occur during manual entry, while OCR can misread or miss characters, or misinterpret to which data field a certain piece of text belongs. Missing and incorrect invoice data complicates the 3-way matching step, because this data is used to determine which documents to compare and is used when comparing PO's and invoices.

For line-level matching, quantities, prices and article numbers or descriptions are compared. These values do not necessarily match up exactly. Article numbers can differ between companies when different companies use different article numbers to identify the same item. For example, some companies use internal catalogues to order goods. These catalogues might contain article numbers which are different from the article numbers used by suppliers. Article descriptions can also vary because different companies can have different ways of describing the same item.

Prices can deviate when item prices vary over time, or when agreed discounts have not been applied on the invoice. The invoice can also contain additional costs that were not included on the PO, such as freight and administration costs. For line items, units other than quantity can be used, such as volume or weight. In those cases values should be converted to the same unit of measurement before comparing between PO's and invoices.

Another challenge for 3-way matching are invoices and PO's that don't match to just one other document. Multiple invoices can match to the same PO. This happens when goods on one PO are split up into multiple deliveries across different days, which results in several invoices and goods receipts all matching to one PO. A single invoice can also match to several PO's when multiple orders are combined.

2 Previous work

2.1 Data matching

Christen [4] gives an overview of the steps in the data matching process and research related to data matching. The goal of the data matching process is finding data records within a dataset, or among several datasets, that refer to the same entity. Figure 2 gives an overview of the steps in the data matching process for two datasets. This process can be used as a basis for a 3-way matching algorithm, because the aim of 3-way matching is to find PO's, goods receipts and invoices that refer to the same order.

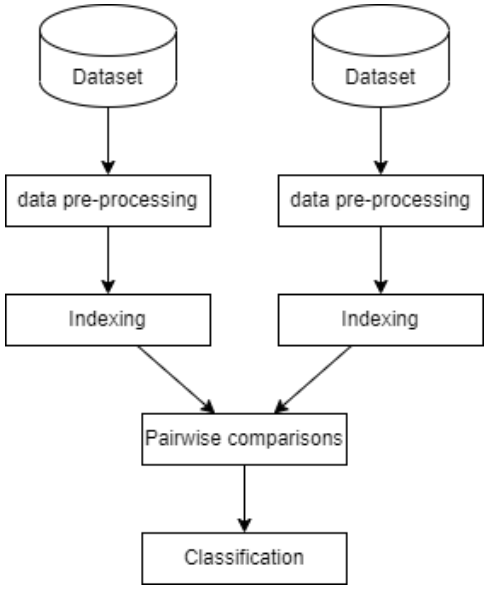

Figure 2: overview of the data matching process

Matching data records is often challenging due to a lack of common identifiers between datasets. If such identifiers are not present, records can often be matched by comparing other attributes. The attribute values are not necessarily unique to the corresponding entities, therefore the choice of attributes to compare is important.

This lack of entity identifiers is also a problem in 3-way matching. PO's usually contain a PO number which can be used to uniquely identify them. If the supplier puts this number on the invoice, it can be used to find the corresponding PO. However, this number is often missing from invoices. If this number is missing, other fields such as supplier name and total price

can be used to find the corresponding PO.

Another challenge in the data matching process is missing or incorrect data. OCR systems are often used to automatically extract data from invoices that are not machine readable. Data can also be extracted from invoices manually. Both methods can lead to missing or incorrect data.

Christen describes the following steps in the data matching process: data pre-processing, indexing, record pair comparison and record pair classification. The data pre-processing step deals with the reformatting and standardisation of data. The indexing step aims to reduce the computational complexity of the process significantly, by only comparing those pairs of documents in detail that are likely to match. In the comparison step, pairs of records are compared in detail and in the classification step, it is decided whether pairs of documents match based on their detailed comparison.

2.2 Data pre-processing

Matching effectiveness can be improved by pre-processing the data records. Data cleaning is sometimes described as a separate step of data pre-processing in the literature. Winkler [37] describes the following pre-processing steps: reformatting into a common form, free-form name standardization, and free-form address standardization. Different records might use different formats for field values such as dates. Reformatting such values into a common form allows for comparison between records. For fields that contain free-form values, punctuation is removed and different spellings are converted to a common form. Then the values are extracted into components such as street names and street numbers for addresses. This allows for comparison of the different components between different data records. Pre-processing and standardization is typically performed using rule-based systems.

2.2.1 Data cleaning

Christen [4] describes the following data cleaning tasks: handling missing values, smoothing noisy values, and identifying and correcting inconsistent values. Data cleaning can improve matching effectiveness, if used in ways that support the matching techniques.

Records that contain many missing fields can be excluded, however this can lead to missing true matches as these records are no longer considered for matching. Certain fields that often have missing values can be excluded from use for matching. If a field is crucial for matching it should not be excluded, but should still be used for those records that do have a value in that field.

Missing values can also be filled in, either manually or automatically, by determining the most likely value. A rule or classification based approach is commonly used for data matching.

Smoothing of noisy values is typically not used for data matching. Values are considered noisy if they contain errors or are outside an expected range. They can be smoothed by replacing them with an average or median value of similar values or by the value of a cluster centroid. By replacing values information that can be used to discriminate between records is lost.

Another data cleaning task is identifying and correcting inconsistent values. Values can be inconsistent within a single record or between several records. If values are inconsistent within a single record, values should only be corrected if it is known which value is incorrect. Otherwise more errors are introduced. Inconsistencies between records can occur when different codings are used for the same field. These values should then be changed such that the same coding is used.

2.2.2 Data pre-processing

Christen also describes several pre-processing steps. The first step is removing unwanted characters and tokens. Tokens are values separated by whitespace characters. Certain characters such as surplus whitespace and tokens that do not contain information useful for matching are removed. Other characters and tokens are converted to standardized forms.

The next step is standardization and tokenization. Standardization is performed by correcting words and abbreviations and replacing them by standard forms using look-up tables or hard-coded rules. Based on the look-up tables or hard-coded rules tags are assigned to the tokens, as part of the tokenization process. These tags can then be used in the subsequent pre-processing step.

The third step is segmentation of tokens into output fields. The tags assigned to the tokens are used to determine how to segment the tokens into different output fields. The goal of this step is to segment values into output fields, such that each output field contains a single piece of information. For example, a date value is split into day, month and year, and a free-format address value is split into the city, postal code, street name and street number.

The final pre-processing step is verification of the values in the output fields. This includes checking correctness of the values as well as combinations of those values. For example, the format of values in IBAN number fields can be checked, as well as the combination of supplier name, IBAN

number and COC number. If a value or combination of values is found to be invalid it can either be corrected or flagged. If no corrections are made the flag can be used to indicate that certain values are inconsistent. For data records that contain inconsistent values, these inconsistencies should be taken into account when comparing data records and determining whether they match.

2.2.3 Effect of data cleaning on matching quality

Randall et al. [30] perform a systematic investigation of the effect of data cleaning on linkage quality. They show that removing missing and uninformative values can improve linkage quality, while extensive data cleaning can decrease linkage quality, as variability in field values is reduced. Instead of extensive data cleaning they recommend the use of approximate comparison methods. These methods can deal with slight errors and variations in values where exact matching methods require values to match exactly. However, in their experiments they used exact matching.

They describe several data cleaning methods, such as reformatting values into a common format, and removing alternative missing values and uninformative values. Alternative missing values are usually a special coding used instead of leaving a field empty when no value is available or known. For fields that contain values such as names phonetic encoding can be used. With phonetic encodings such fields can be compared on sound instead of spelling. This can help deal with typing errors, as names with the same sound but a different spelling will have the same phonetic encoding.

2.2.4 Conclusion

The specific data cleaning and pre-processing methods needed for invoice matching depend on the dataset used. Invoice and order header fields that are often used for matching are PO-number, supplier name, administration and total amount. For this thesis no extensive data cleaning methods will be used, only documents containing no values useful for header matching will be removed. Data pre-processing methods will be used, such as putting values in a common format. We will not use methods such as correction or verification of values.

2.3 Indexing methods

Comparing each possible PO and invoice pair is inefficient for large amounts of documents, since the majority of document pairs do not match. A method

is needed to decide which pairs of documents to compare, to reduce the total number of document comparisons while missing as few true matches as possible. In literature this is often called blocking or indexing.

Christen [5] gives an overview and experimental evaluation of various indexing techniques. Such techniques generate keys for each of the data records, based on their attribute values. These keys are then used to generate pairs of documents to compare in more detail. Such keys are called blocking keys, and all documents with the same blocking key form a block. A common issue to consider when generating such keys are errors and missing data. True matches can be missed when keys are generated based on fields that contain errors or missing data. A solution is generating multiple keys based on different attributes. Another common issue is the size of the generated blocks. Smaller blocks result in a lower total computation time, while possibly missing more true matches.

Christen describes the indexing step as a process consisting of two phases, the build and the retrieve phase. For each document a blocking key value (BKV) is generated. The next phase is the retrieval phase. Pairs of documents are generated by pairing documents based on their BKV. These pairs of documents are then compared in detail, in the next step of the data matching process.

2.3.1 Traditional blocking

The first indexing technique described by Christen is a traditional blocking approach. In this approach only pairs of documents with the same BKV are generated. True matches can be missed if data records contain errors or have missing data in the fields that are used to generate the BKV's. A solution is generating multiple BKV's based on different data fields. For this approach it is difficult to predict the total number of document pairs that will be generated, as this depends on the frequency distribution of the BKV's.

2.3.2 Sorted neighbourhood indexing

Another indexing technique is sorted neighbourhood indexing. The idea of this technique is to sort the datasets on their BKV's and sequentially move a window over these sorted documents. Pairs of documents are then generated only if they appear within a window. With this technique the total number of pairs of documents that is generated does not depend on the distribution of BKV's but on the window size and the sizes of the datasets.

Choosing a too small window size can lead to missing true matches, as some blocks might be larger than the window size. A solution is to generate blocking keys that are concatenations of several fields, such that the resulting blocks are smaller. Because the datasets are sorted on the BKV's, true matches can be missed if documents contain errors or variations in the first few positions of a value that is used for the generation of the BKV's. A solution to this problem is the use of several blocking keys based on different record fields, or the use of blocking keys based on reversed field values.

2.3.3 Q-gram-based indexing

Q-gram-based indexing is a technique which creates blocks consisting of documents with similar, instead of only exactly the same BKV. Blocks are created based on q-grams, substrings of the BKV's of length q. Each document can therefor be part of multiple blocks. With this technique more true matches can be found, but more pairs of documents are generated which increases the total time needed for the comparison step.

2.3.4 Suffix array-based indexing

Another indexing technique for which documents can be part of multiple blocks is suffix array-based indexing. Blocks are created based on BKV's and their suffixes, down to a minimum length. A document can be part of several blocks, depending on the length of its BKV. The maximum size of blocks is limited by a parameter for the maximum block size. Blocks that contain more than this maximum will not be used to generate pairs of documents. A variation of this technique uses substrings of the suffixes down to a minimum length, to deal with errors at the end of BKV's.

2.3.5 Canopy clustering

Canopy clustering is an indexing technique that creates overlapping clusters based on similarities between BKV's, using similarity measures such as Jaccard or TF-IDF/Cosine. The clusters are created based on approaches using thresholds or nearest neighbours. For the threshold based approach it is difficult to estimate the total number of record pairs that will be generated for comparison, while for the nearest neighbour approach this number can be calculated.

True matches can be missed by the nearest neighbour approach if some BKV's are frequent and the size of the clusters is too small to capture

all these pairs with the same BKV. Similar to the sorted neighbourhood indexing technique, a solution is generating BKV's that are concatenations of several fields. Both approaches are not sensitive to errors in the beginning of BKV's because of the similarity measures used.

2.3.6 String-map-based indexing

The final indexing technique described by Christen is string-map-based indexing. This technique maps BKV's to objects in a multidimensional Euclidian space, using a string similarity measure that is a distance function. Groups of similar BKV's are then generated by extracting clusters of similar objects from this space. Overlapping clusters can be generated using a threshold or nearest neighbour-based approach.

2.3.7 Redundancy

Certain indexing methods result in overlapping blocks, which can increase the effectiveness [23]. Documents can be inserted into multiple blocks which can increase the number of true matches found. This also leads to redundancy and thus a higher time-complexity. If pairs of documents appear together in multiple blocks, the same pair of documents will be compared multiple times.

Papadakis et al. propose two methods to eliminate the redundant document pair comparisons. The first method is optimal, as it eliminates all redundant document pair comparisons. This method sets up a data-structure which is used to efficiently check if a document pair has been compared already. This method does however have a quadratic space-complexity because of this data-structure. The second method they propose approximates the optimal solution at a lower space-complexity. This methods discards blocks that only contain redundant pairs and merges highly overlapping blocks.

2.3.8 Comparison of indexing methods

Experiments by Christen [5] show that q-gram-based, string-map-based, suffix array-based indexing techniques and threshold-based canopy clustering have slow runtimes, while traditional blocking and versions of the sorted neighbourhood approach have the fastest runtimes.

The overall efficiency of the different techniques is measured by the f-score of pairs completeness (PC) and pairs quality (PQ). PC is the number of true matched record pairs generated divided by the total number of true

matches, and corresponds to the recall. PQ is the number of true matched record pairs generated divided by the total number of record pairs generated, and corresponds to precision.

The f-score is the harmonic mean of PC and PQ. Based on the f-score, traditional blocking performs best, followed by threshold based canopy clustering, q-gram based indexing and a version of the sorted neighbourhood technique. Traditional blocking performs surprisingly well compared to other more complex techniques. The author argues that the definition of suitable blocking keys has a greater effect on the overall efficiency than the specific indexing technique used.

2.3.9 Conclusion

In this thesis we will use traditional blocking for the indexing step and we will compare several blocking keys. As discussed in 2.3.8, the choice of blocking keys has a greater effect on the results than the choice of indexing method. More research is needed to find out which indexing method, as well as which blocking keys work best for invoice matching specifically. In future research other indexing methods can be compared to traditional blocking, to find out if more sophisticated methods can improve the invoice matching process.

2.4 Comparison methods

The document pairs generated in the indexing step are compared in detail in the comparison step. The document pairs are compared on corresponding attribute values, each of these comparisons results in a similarity value between 0 and 1 [4]. These values correspond to the degree of similarity, with a value of 1 corresponding to an exact match. Exact matching methods result only in similarity values of 0 and 1, while approximate matching methods give results between 0 and 1. Several attribute values are compared between a document pair, resulting in a vector of similarity values, also called a comparison vector. In the classification step this comparison vector is used to classify a document pair as a match, non-match or possible match.

Christen [4] argues the need for approximate comparison methods, as documents can contain errors and variations which can not be resolved by data cleaning and pre-processing. Christen describes several approximate comparison methods for strings as well as for other types of data such as numerical values and dates. Two variations of exact comparison functions are described for strings, truncate and encoding comparison. The truncate

comparison function compares only the beginning or end of two strings for an exact match. The encoding comparison function first encodes two strings using an encoding function such as phonetic encoding and then performs an exact comparison on these encoded strings.

2.4.1 Edit distance

Christen also describes several methods for approximate string comparison. Edit distance functions compute a distance value, which can be converted to a similarity value. The edit distance between two strings is defined as the smallest number of edits needed to convert one string into the other. Levenshtein is an edit distance function, which computes the smallest number of character insertions, deletions and substitutions needed. The Damerau-Levenshtein edit distance adds transposition of two adjacent characters as an edit operation. This corresponds to the common typing error of swapping the position of two adjacent characters.

Variations of the Levenshtein edit distance give different cost values to the different edit operations, as well as different cost values for the substitution edit operation based on the specific characters that are substituted. Smith-Waterman edit distance is similar to the Levenshtein edit distance. It has five different operations, each with a different match score. Besides matching and mismatching characters, it also has an operation for an approximate match between two similar characters. Similarity between characters can be based on sound, shape or another measure, based on the type of errors the comparison function needs to handle. There are also two operations for dealing with gaps in strings. Each operation adds a match score or cost, using this total match score a similarity value can be calculated.

2.4.2 Q-gram based string comparison

Q-gram based string comparison compares strings on their q-grams, substrings of length q. The similarity value corresponds to the number of q-grams the strings have in common. There are two variations to this method. The first adds $q-1$ special characters at the beginning and end of the strings, such that each of the characters from the original string appear in the same number of q-grams. The second variation adds positional information to the q-grams, such that only q-grams within a set maximum distance are considered as common q-grams.

2.4.3 Jaro and Winkler

Jaro and Winkler have both developed approximate string comparison functions for the comparison of names. The comparison function by Jaro calculates a similarity value based on the characters in common between a certain distance, and the number of transpositions of adjacent characters. A modification to this comparison function by Winkler gives higher values for strings that have matching prefixes.

2.4.4 Monge-Elkan

The Monge-Elkan approximate string comparison function was developed to deal with strings that contain multiple words, such as business names. The strings are first split into tokens, the similarity is then calculated by finding the best matching pairs of tokens and taking the average similarity of those token pairs.

2.4.5 Extended Jaccard

The extended Jaccard comparison function calculates a similarity based on similar tokens or common q-grams between strings and is based on the Jaccard coefficient. The Jaccard coefficient is calculated as the ratio of the size of the intersection divided by the size of the union of two sets.

2.4.6 SoftTFIDF

The SoftTFIDF string comparison function is based on TFIDF, which is a method for determining similarity between queries and text documents based on frequencies of words within a document and among a collection of documents. The main idea is that words are more relevant for determining the similarity if they appear more often in a document, and more relevant if they appear less often among the whole collection of documents. Instead of calculating the similarity based on exactly matching words, SoftTFIDF calculates the similarity based on similar tokens.

2.4.7 Longest Common Substring

The Longest Common Sub-string comparison function calculates the similarity based on the total length of common substrings of a minimum length between two strings, by iteratively finding and removing those substrings. This comparison method deals well with strings that consist of several words that appear in a different order.

2.4.8 Numerical values

Christen [4] describes methods for approximate comparison of several value types other than strings. For numerical values two methods are described. The first method allows for an maximum absolute difference between the two numerical values. If the difference is greater than this value the similarity value is 0, otherwise the similarity is calculated based on the difference between the two values.

The second method calculates the similarity between numerical values based on the relative difference. This method allows for a maximum percentage difference, numerical values with a percentage difference above this maximum get a similarity value of 0.

2.4.9 Date values

There are several methods for comparing date values [4]. The similarity between dates can be calculated based on the absolute difference in days, given the allowed maximum absolute difference. This is similar to the comparison of numerical values based on their absolute difference. Similarity between dates can also be calculated by comparing the day, month and year values. The similarity is then based on the number of values that match exactly. Another method is to compare dates based on the difference between the number of days since some other date.

2.4.10 Comparison of matching improvement

Invoice data can be entered into a system manually which can lead to typing errors. The data can also be extracted from an invoice by using OCR, which can lead to scanning errors. Porter and Winkler [25] show that matching effectiveness can be improved by using approximate string comparison functions instead of exact matching. A string comparison function based on exact matching always returns a value of 0 for strings that do not match exactly, even if they are very similar. Approximate string comparison function return a value between 0 and 1, based on how similar two strings are.

They describe several string comparison functions and compare their effect on matching effectiveness. The first string comparison function they describe was introduced by Jaro, and accounts for insertions, deletions and transpositions. They describe three enhancements to this comparison function. The first is by McLaughlin and deals with scanning and typing errors. The second is by Winkler and gives increased value to matching characters

on the beginning of strings, as typically the fewest errors occur at the beginning of a string. The third enhancement they describe is by Lynch and Winkler. This one adjusts the comparison function value for longer strings, if more than half of the characters beyond the first four match. They also describe a string comparison method based on bigrams. This method returns the number of bigrams two strings have in common divided by the average number of bigrams in the two strings.

Their experiments show that the use of approximate string comparison functions improves matching quality for datasets that contain typographical errors. The variant based on the comparison function by Jaro results in a smaller amount of possible matches, the document pairs that require manual review.

2.4.11 Conclusion

For this thesis we will use approximate comparison methods, as header data can contain variations due to typos and OCR errors. Total amounts can also vary slightly between an order and invoice, due to price variations for example. With approximate comparison methods, similar but slightly different values will receive a similarity value closer to 1, while they will receive a similarity value of 0 with exact matching.

With approximate matching we can differentiate between pairs of values that are similar but slightly different and pairs of values that are dissimilar. For string values such as the supplier name we will use a similarity measure based on the Levenshtein distance. For numerical values we will use numerical comparison methods based on the absolute and relative differences between values.

2.5 Classification methods

The pairwise comparisons of documents result in comparison vectors, which can be used to classify the pairs of documents as matches and non-matches. Christen [4] describes several classification methods based on comparison vectors.

2.5.1 Threshold-based classification

The first method described is threshold-based classification. The sum of values in the comparison vector is compared to either one or two classification thresholds. If a single threshold is used, the document pair is classified as a

match if the sum of values in the comparison vector is greater than or equal to the threshold, otherwise it is classified as a non-match.

If two thresholds are used, a lower and upper threshold are defined and the pair of documents can also be classified as a potential match. If the sum of values in the comparison vector is below or equal to the lower threshold, the pair of documents is classified as a non-match. If the sum is above or equal to the upper threshold the pair is classified as a match. Otherwise the pair is classified as a potential match. For pairs of documents that are classified as potential matches, manual review is needed to determine whether the pair of documents match. The values for the thresholds can either be selected manually or learned from a dataset for which the true matches and non-matches are known.

A drawback of this approach is that by taking the sum of values in the comparison vector, each value contributes equally in classifying a document pair, even though some of the values might be more useful for classification. A solution to this is assigning weights to each of the document attributes and using these to calculate the weighted sum of comparison vector values.

2.5.2 Probabilistic classification

Fellegi and Sunter [11] propose a method for record linkage by defining linkage rules. A linkage rule classifies a pair of documents as a match, non-match or possible match based on their comparison vector. The optimal linkage rule, at specific error levels, is defined as the linkage rule that minimizes the number of documents classified as possible matches at those error levels. The error levels are defined by two probabilities: the probability a true match is classified as a non-match, and the probability a true non-match is classified as a match.

For each of the attributes two weights are calculated, one for the case where the attribute values match and one for the case where the attribute values don't match. These agreement and disagreement weights are calculated based on two conditional probabilities: the probability an attribute value matches given that two documents are a true match, and the probability an attribute value matches given that two documents are a true non-match. These probabilities can either be estimated or they can be calculated from a dataset where matches and non-matches are known. Using these weights an overall weight can be calculated for a pair of documents, which is then used to classify the pair as a match, non-match or possible match using an lower and upper threshold. The values for the thresholds are determined based on the error levels.

The method proposed by Fellegi and Sunter creates comparison vectors by checking if attribute values match exactly. Porter and Winkler [25] show that matching quality can be improved by using approximate string comparison methods instead of exact matching.

Winkler and Thibaudeau [38] propose a method for calculating agreement and disagreement weights based on the frequency of attribute values in a dataset. They argue that less frequent attribute values are more discriminative for classifying a pair of documents.

The probabilistic record linkage method by Fellegi and Sunter assumes conditional independence between attribute values. Winkler [36] describes how to combine their method with Bayesian networks to model conditional dependencies between attributes.

Another probabilistic classification method is Naive Bayes classification [2]. This method is based on Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Here $P(A|B)$ denotes the conditional probability of A given B. If we have a dataset X and classes Y, and each instance $x_i \in X$ belongs to exactly one class $y_j \in Y$, we can use a Naive Bayes classifier to model the probability of an instance x having class y as follows:

$$P(y_j|x_i) = \frac{P(x_i|y_j)P(y_j)}{P(x_i)}$$

For invoice matching instances of X correspond to comparison vectors, with similarity values as attributes. The classes Y correspond to the matches and non-matches. A classification is then made by assigning the class with the highest probability.

Naive Bayes classifiers make the assumption that the values of individual attributes are independent from each other. In practice, they are not independent, but literature shows Naive Bayes classifiers give good results despite this assumption.

Several papers use a Naive Bayes classifier for spam classification of emails. Both [9], [1] and [31] use a Naive Bayes classifier with a threshold for the probability an email is spam. If the probability for the class spam is above the threshold, an email is classified as spam. This helps reduce the number of non-spam emails classified as spam. Depending of the cost

of misclassification, a threshold can be set such that the expected cost is minimized.

Delany et al. [10] describe and compare two possible methods for calculating confidence levels for classification. The first method uses confidence measures based on k-nearest neighbours classifiers. The other method uses the probability estimated by Naive Bayes as a level of confidence for classification.

They make a distinction between different levels of confidence. Emails that are classified as spam with a high level of confidence are considered spam, while emails that are classified as spam with a lower level of confidence are considered possible spam. This allows the user to check whether an email that is considered as possible spam actually is a spam email.

Their idea is that the classifier should only have a high level of confidence for classifying an email as spam if the email is truly spam. They argue that probabilities calculated by Naive Bayes can't be used as confidence level for classification as it sometimes has high probability for a wrong classification. However, the consequence of making a wrong classification depends on the specific application. The authors used classification to find spam emails, which can have the consequence of missing an important email when it is wrongly classified as spam. For 3-way matching the consequences of wrong matches are either having to do manual matching because a match is not found, or having to do a more detailed comparison on a pair of documents that do not match. Assuming that it is concluded whether the documents really match in a more detailed comparison, false matches should at most result in a higher computation time.

Pronk et al. [26] describe a method for calculating a level of confidence for Naive Bayes classifiers. For each classified instance, their method estimates the variance of the classification probability for each class, based on the training data. Based on this variance and a parameter k they define confidence intervals for the classification probabilities.

In general, for Naive Bayes classification the class with the highest classification probability would be assigned to an instance. The authors first compare the highest classification probability and its confidence interval with the classification probabilities and confidence intervals of the other classes, to determine whether they can make a reliable classification. If the confidence interval of the class with the highest classification probability overlaps with the confidence interval of any other class, the instance does not get assigned a class as they argue they are not able to reliably classify the instance.

Larger values of k result in larger confidence intervals, which decreases the number of instances that can be classified reliably. This parameter is used to decrease the classification error at the cost of the coverage, the fraction of instances that can be classified.

They evaluate their method by measuring its performance on the classification of TV programs. They classify programs as positive or negative for multiple people, based on their viewing history. They compare classification results for different values for k. Their results show that the classification error can be reduced significantly, at the cost of the coverage. They also show that they can achieve a higher coverage with a larger training dataset.

2.5.3 Cost-based classification

There are two types of classification errors, classifying a true match as a non-match, and classifying a true non-match as a match. The objective for most classifiers is to minimize the total amount of errors made. However, the different type of errors can have different consequences and therefore different costs assigned to them, depending on the specific data matching application. For invoice matching, the different classification errors have different consequences. False matches increase computation time, as they will require a detailed comparison. For false non-matches and document pairs classified as possible match, manual matching is needed.

Verykios et al. [34] describe a cost-based model for classification. Costs are defined for each possible combination of classification outcome and actual matching status. The objective of this classifier is to minimize the total expected costs for classification.

2.5.4 Rule-based classification

With rule-based classification, document pairs are classified by logical rules in the form of $P \Rightarrow C$, with a predicate P and a classification outcome C [4]. The rules are applied to the comparison vectors of document pairs. The predicate is a boolean expression, consisting of terms. Each term is a test on a value of the comparison vector, resulting in a boolean value. For example, the predicate could contain a term checking if the similarity value of the supplier names is above 0.8. These rules can be represented as 'if-then' statements consisting of a condition and a conclusion [29].

If a document pair is checked against a rule, it is classified with the corresponding class C if the result of the predicate is true for that document pair. If different rules have different classification outcomes, the ordering of

the rules is important. A document pair is classified by the first rule for which the result of the predicate is true.

The set of rules can either be generated manually based on domain knowledge, or learned from a dataset with known match status. To learn the rules a sequential covering algorithm can be used. Such an algorithm learns the rules one by one, by starting with a rule with an empty predicate and iteratively adding a new term, and checking its accuracy and coverage. This process is repeated until the rules cover the whole dataset.

Compared to other classifiers, rule-based classifiers are easy to understand because of the explicit classification rules [20]. The model can be interpreted and the classification results can be explained based on these rules.

Qian et al. [28] present SystemER, an entity resolution tool that learns a rule-based classification model. They argue that such models are verifiable and customizable, as the model consists of rules that are readable for users that understand the data. The model is learned iteratively using active learning. This requires a user to manually label the document pairs that are likely false positives or likely false negatives. A document pair is considered a likely false positive if it is classified as a match but has a relatively low confidence, while a document pair is considered a likely false negative if it is classified as a non-match but has a relatively high confidence. The confidence of a document pair being a match is calculated based on the boolean comparison features.

2.5.5 Supervised classification

If training data is available, supervised classification methods can be used [4]. The training data should contain labelled examples of matching and non-matching pairs of documents. This training data is then used to train a classification model. Test data is used to determine the matching quality of the trained model. Test data is labelled data that has not been used to train the model, and must be in the same format as the training data. If the quality of the model is good enough, the model can be used to classify new unlabelled data.

Nasteski [22] gives an overview of supervised machine learning methods, several of these methods can be used for classification. Loog [21] gives an overview of the ideas behind classification techniques, and gives some examples of supervised classification models.

A decision tree is a classifier that consists of nodes and edges [4], [22], [21]. Each node is either an internal or leaf node. The internal nodes correspond to a test on a specific value in the comparison vector, the leaf nodes correspond to a classification outcome. The outcome of a test in an internal node determines which outgoing edge to follow. A document pair is then classified by starting at the root node and following edges until a leaf node is reached. A decision tree is generated by starting with an empty tree, and recursively adding new nodes that split the training data into matches and non-matches as best as possible. The tree can be pruned afterwards to reduce its complexity.

A support vector machine (SVM) is a binary classifier that maps the training data to a multi-dimensional vector space such that the two classes are separated by a decision boundary [4]. A decision boundary corresponds to a hyperplane in high-dimensional space, or a line in two-dimensional space. The optimal decision boundary has the widest margins to the training data of both classes.

Bayesian classification models predict the output class by calculating the probability that input data has a specific class [22], [21]. The model learns conditional probabilities between values from the training data and the output classes. The class with the highest probability is assigned to the input data.

Logistic regression can be used for classification of two output classes. Training data is fit to a logistic function, which maps the input data to values between 0 and 1. Based on this value the predicted class is determined, the values 0 and 1 correspond to the output classes.

Neural networks are supervised classifiers that are built up out of neurons, small computational units that take multiple inputs and return a single numerical output [21]. These neurons usually consist of a function that maps the inputs to a linear value, and an activation function that maps that linear value to a value between 0 and 1. Nodes can be connected in parallel, and provide inputs to subsequent nodes. A set of nodes that provides inputs to a subsequent set of nodes, is called a layer. Complex neural networks can be constructed with many layers and several activation functions.

A nearest neighbour classifier determines the distance between feature vectors of records in the training set and a record to be classified [21]. The record is then assigned the same class as the record in the dataset that has the closest feature vector. When considering the k closest feature vectors, the most common class among these records is assigned. This is called k nearest neighbours.

Köpcke et al [19] evaluate several classification approaches for entity resolution. These contain learning-based as well as non-learning based approaches. They compare the different classification approaches on runtime and matching quality, by using the same datasets and blocking strategies. The non-learning based approaches consist of threshold based methods and a method based on Fellegi and Sunter. The learning-based methods consist of SVM's and decision trees. Their results show that learning-based methods have a better matching quality than non-learning based methods, when comparing on multiple attributes.

Christen [4] describes several challenges that occur when using supervised classification for data matching. The number of non-matches is usually much larger than the number of matches, even after performing an indexing step. The classification technique and the quality measure that are used should handle this imbalance. A solution is to use a training dataset which contains equal amounts of matches and non-matches. The training of a supervised classification model requires a large enough training dataset. This dataset should also be representative for the data that will be classified by this model. Obtaining such a dataset can be challenging, as labelling a dataset manually is costly and time-consuming.

2.5.6 Active Learning

Supervised classification methods usually require large training datasets to achieve high accuracy. If only a small training dataset is available, active learning methods can be used [4]. An initial model is trained on the dataset, this model is then used to classify new data. The data that was most difficult to classify is then classified manually, and added to the training dataset. A new model is then trained on this training dataset. By repeating this process of adding new labelled data to the training dataset and training a new model, the accuracy of the model improves.

Qian et al. [27] introduce a system based on active learning that learns classification rules for entity resolution. Their system learns multiple rules by searching for examples that are likely false positives or likely false negatives, non-matches that are classified as matches and matches that are classified as non-matches. The learning process repeats the following steps. First a new rule is learned on the training dataset. The system then finds likely false positives and likely false negatives. These examples are labelled by a user and added to the training dataset. The new rule is then either

accepted or rejected, based on its precision.

Examples for labelling are found by estimating the confidence a record pair matches based on similarity measures. Record pairs that were classified as a match but have a relatively low confidence are considered likely false positives, while record pairs that were classified as non-matches but have a relatively high confidence are considered likely false negatives. The authors describe bootstrap as a possible method to estimate the confidence. The training dataset is sampled uniformly with replacement to create several sets of training data. For each of these sets a rule is learned. The confidence of a document pair is then calculated as the fraction of rules that classify the document pair as a match. A drawback of using bootstrap to estimate the confidence is that it requires evaluating multiple rules per document pair, depending on the number of training sets used.

Instead of using bootstrap, they use a similarity measure to estimate the confidence. The classification rules consist of predicates, which result in boolean values. The predicates are assumed to measure how well a document pair matches. The predicates should result in more positive values for document pairs that match better. The confidence for a document pair is then calculated as the fraction of the predicates that evaluate to true for that document pair.

They perform experiments comparing this system to other supervised and active learning classification systems for entity resolution. The results show that their system achieves high precision and recall, while only requiring a minimal amount of manual labelling.

Christen et al. [8] propose an active learning approach for entity resolution that uses an informativeness measure to identify examples for manual labelling. This measure is a weighted sum of the entropy and uncertainty of the similarity vector of a document pair. The entropy of a similarity vector is based on information entropy, which measures how balanced a dataset is. The entropy of a similarity vector is high if it is close to similarity vectors of both matches and non-matches in the training set. Similarity vectors are considered close to each other when their Cosine similarity is high. The uncertainty of a similarity vector is defined by the number of similarity vectors in its search space. The search space is the area between the similarity vector itself and the closest similarity vector in the training dataset of the other class. If the number of similarity vectors in this search space is higher, the uncertainty is lower.

First, an initial training dataset is generated, by selecting a subset of similarity vectors and labelling them manually. Further examples for manual

labelling are then selected by finding a limited number of similarity vectors of unlabelled document pairs that are close to similarity vectors with high informativeness. The idea is that similarity vectors that are close to more informative similarity vectors in the training set are more informative as well. By selecting more informative similarity vectors less manual labelling is needed to generate a suitable training dataset.

They perform experiments comparing their approach to other approaches based on supervised and active learning. These experiments show that their approach achieves comparable results to supervised learning based approaches using a much larger dataset. They also show that their approach outperforms previous active learning based approaches that are limited in the number of examples that can be labelled manually.

Wang et al. [35] propose a method for efficiently selecting training data for entity resolution based on active learning. Their method is based on the cluster structure of similarity vectors, and aims to minimize the amount of manual labelling needed to accurately classify document pairs.

The vectors are recursively split into clusters, for each cluster a subset of the most informative vectors is selected for manual labelling. They set a limit on the total amount of examples that can be labelled manually. If the purity of the cluster is high enough all the vectors in that cluster are classified as either matches or non-matches, and added to the training dataset. Otherwise, if the size of the cluster is above a predefined minimum cluster size, it can be split further. The purity of a cluster is defined as the proportion of document pairs labelled as the majority class out of all labelled document pairs in that cluster. Once the limit on the total amount of examples to label manually is reached, the training dataset is returned.

They performed experiments comparing their approach to several fully supervised, and unsupervised learning based approaches. They show that their approach achieves comparable matching quality while significantly reducing the amount of manual labelling needed.

Christen et al. [7] propose a method for training data selection based on the approach by Wang et al. [35]. The amount of examples that can be labelled manually is limited. Because of this limit, they argue that the order in which the clusters are processed by the algorithm is important. The clusters that are most likely to improve the quality and coverage of the training dataset should be considered first.

They present several methods to decide on an ordering for the clusters. The first method orders clusters in the order in which they are created, the

same method as used in [35]. Other methods order the clusters, either in ascending or descending order, based on the purity of their parent cluster, their distance to the similarity vectors consisting exclusively of 0's or 1's, cluster size, ratio of pairs classified as matches and non-matches, or a combination of these measures. Their experiments show that the measures based on cluster size, ratio of pairs classified as matches and non-matches and the combination of measures achieve the best match quality.

Chen et al. [3] present HeALER, a committee-based active learning method for entity resolution. With committee-based active learning, multiple machine learning models are trained. These models form a committee that are used together to classify data. Data is then labelled as follows: each of the models in the committee predicts a label. The label that was predicted most by these models is chosen as the label for that data.

The document pairs that are considered most informative are chosen for manual labelling. A document pair is considered informative when the models in the committee disagree on the predicted label. For each document pair labelled by the committee a disagreement value is calculated, based on the total number of pairs of models in the committee with a different classification result.

Usually, for committee-based active learning a single type of classifier is used for the models in the committee. To create a diverse set of models using a single type of classifier each model can be trained using a different subset of the training data or by using different parameter settings. The authors argue that a committee consisting of models of several classifier types can give better classification results. They perform experiments comparing a single machine learning model and two committees of models with a single type of classifier to a committee of models with different types of classifier. Their results show that a committee consisting of different types of classifier gives better classification results.

2.5.7 Conclusion

There are several classification methods that can be used for data matching. For this thesis we will use Naive Bayes classification. With this method we can calculate a classification probability. Our idea is to use this probability to determine a level of confidence for match classifications.

2.6 Level of confidence

Zadrozny and Elkan [39] describe a method for calibrating the classification probabilities calculated by a Naive Bayes classifier. Calibration is needed, as the calculated probabilities are usually too extreme. The values are either very close to 0 or very close to 1. Therefore they can not be used directly as a level of confidence for the classification outcome.

The probabilities are calibrated as follows. Classification probabilities are calculated on a training dataset using a Naive Bayes classifier. The instances in the training dataset are then sorted on this classification probability. The sorted set of instances is then divided into equal sized bins, and for each bin the upper and lower classification probability is determined. The calibrated probabilities are estimated per bin by calculating the fraction of training instances in that bin belonging to a specific class. The calibrated probability for a newly classified instance is then determined as the calibrated probability for its corresponding bin.

2.7 Parallelization

For large datasets the entity resolution process can be time-consuming. Several methods using parallelization have been proposed to reduce the total running time.

Gazzari and Herschel [13] propose a framework for the task-based parallelization of the entity resolution process. Most of the literature focusses on data parallelism. Each of the individual steps are parallelized by dividing the dataset into subsets and running the steps in parallel on these subsets. The steps of the entity resolution process are executed sequentially. With task-based parallelism however, the different steps are executed in parallel as well, which allows partial results to be returned continuously.

They describe how the steps in the entity resolution process should be adapted to allow for task-based parallelization. They also describe a possible implementation as a pipeline consisting of several stages. Each of the stages perform one or more steps of the entity resolution process on a stream of input data, and continuously output results.

2.7.1 MapReduce

MapReduce is useful for data-intensive tasks that can be performed in parallel. Kolb et al. [14] propose two methods based on MapReduce for the pair-wise similarity computations and classification of the Cartesian product of two datasets. The Cartesian product of two sets of documents is the

set of all document pairs with one document from each of the datasets. For MapReduce two functions are defined, 'map' and 'reduce' which work with data represented as key-value pairs. The map function is executed first, the reduce function second using results from the map function, grouped by the key values. These functions are performed in parallel for subsets of the data. The first proposed method performs the pair-wise comparisons in the map function, while the second method performs the pair-wise comparisons in the reduce function. They show the time-complexity can be reduced significantly by performing the similarity computations and classification in parallel using MapReduce.

Kolb et al. [16] propose efficient methods for entity resolution with sorted-neighbourhood blocking using MapReduce. By using MapReduce the blocking and comparison steps can be performed in parallel efficiently. They show methods for performing single-pass as well as multi-pass neighbourhood blocking using MapReduce. With single-pass neighbourhood blocking a single blocking key is used, while for multi-pass neighbourhood blocking multiple blocking keys are used. Both methods require only a single MapReduce job.

Kolb et al. [17] propose a method for eliminating redundant document pair comparisons in entity resolution using MapReduce. In the map function they create the blocking keys for the documents. For each of the documents, the map function outputs one key-value pair per blocking key. The blocking key is used as key, and the value contains the document and a specific subset of the blocking keys for that document. In the reduce function, these subsets of blocking keys are used to determine if a pair of documents should be compared. Experiments show that the elimination of redundant pair comparisons significantly reduces the time complexity.

Kolb and Rahm present Dedoop, a framework for entity resolution based on MapReduce [15]. This framework allows users to perform entity resolution within one or among two datasets, either on the Cartesian product or using blocking methods with user-specified keys. The implemented blocking methods are Standard Blocking (traditional blocking) and Sorted Neighbourhood Blocking. Both have a multi-pass version, for blocking with multiple keys. For comparison of documents several numerical and string similarity measures can be used, and for classification of documents match rules can be specified manually or a machine learning algorithm can be used. Dedoop has methods for load balancing of the comparison and classification

of documents, as blocks can vary significantly in size. Dedoop also has methods to prevent redundant comparisons and classifications of document pairs. Experiments using Dedoop show that execution times for entity resolution of large datasets can be reduced significantly by performing the comparison and classification in parallel using MapReduce.

2.7.2 Apache Spark

Gagliardelli et al. present SparkER [12], an enitity resolution system implemented on Apache Spark. Spark is a system used for processing of large datasets, on a single node or distributed among clusters. This system consists of several independent modules, which are each parallelizable. The modules each perform separate steps of the entity resolution process. Output of a module is used as input for a subsequent module. The first module is the blocker module, which performs the blocking phase and outputs candidate pairs. The next module is the entity matcher, which classifies the candidate pairs as matches and non-matches. The last module is the entity clusterer. This module clusters the pairs that were classified as matches into groups that represent the same entity.

Qian et al. present SystemER, an entity resolution system [28] which uses active learning to learn a rule-based classification model that is human-comprehensible. The model learning part of the system is implemented on Spark, which allows it to run on a single node or in parallel on a cluster of multiple nodes. Experiments on large datasets show that using multiple nodes can speed up computation times significantly.

Papadakis et al. [24] present JedAI, an end-to-end entity resolution system. It supports both serial as well as parallel execution. The algorithms for parallel execution are implemented on Apache Spark. They perform experiments comparing the runtime for both serial and parallel execution. Their results show that for larger datasets the run-times are significantly lower when using parallelization, while for smaller datasets the run-times are similar for both serial and parallel execution.

2.7.3 Conclusion

For data matching with very large datasets, parallelization can be very useful, as it can reduce the total computation time significantly. Such methods

will not be implemented for this thesis, but they are interesting for further research, especially if computation time becomes an issue.

2.8 Quality and Complexity measures for data matching

Several quality and complexity measures have been proposed for the data matching process. Calculating these measures requires a dataset of known matches and non-matches.

Christen and Goiser [6] describe several quality measures that are commonly used for data matching. To calculate these measures the true matches and non-matches should be known for the dataset that is used. The measures are based on the numbers of document pairs classified correctly as matches and non-matches, also known as the true positives (TP) and true negatives (TN), and the numbers of pairs of documents classified incorrectly, also known as the false positives (FP) and false negatives (FN).

Measures calculated using the number of true negatives should not be used, as this number is usually significantly larger than the other numbers in data matching problems [4], [6]. The number of true negatives represents the number of true non-matches. Measures that are calculated without using the number of true negatives are precision, recall and the F-measure.

Christen and Goiser also recommend using the precision-recall and F-measure graphs to compare the matching quality of different techniques. A precision-recall graph can be created by plotting precision values for different recall values. An F-measure graph can be created by plotting the F-Measure for different threshold values for a classifier.

2.8.1 Precision

For data matching, precision is defined as the number of document pairs correctly classified as a match divided by the total number of document pairs classified as a match:

$$\frac{|TP|}{|TP| + |FP|}$$

Precision is also known as positive predictive value. It is a measure for the validity of the pairs classified as a match.

For the indexing step, precision is also known as the pairs quality. A high pairs quality means the document pairs found by the indexing step are mostly matches.

2.8.2 Recall

For data matching, recall is defined as the number of pairs of documents correctly classified as a match divided by the total number of true matches:

$$\frac{|TP|}{|TP| + |FN|}$$

Recall is also known as sensitivity. It is a measure for the completeness of the pairs classified as a match.

For the indexing step, recall is also known as the pairs completeness. A high pairs completeness means most of the matches are found by the indexing step.

2.8.3 Reduction ratio

The indexing step reduces the number of document pairs that are considered for comparison. The reduction ratio is a measure for this reduction:

$$1 - \frac{N_b}{|A| * |B|}$$

N_b is the number of document pairs generated by the indexing step. $|A|$ and $|B|$ correspond to the sizes of the sets of documents. This measure tells us how well the comparison space is reduced, but does not say anything about the quality of the generated document pairs.

2.8.4 F-measure

The F-measure is the harmonic mean of two values, usually the precision and recall:

$$2 * \frac{precision * recall}{precision + recall}$$

Typically, a high precision can be achieved at the cost of recall while a high recall can be achieved at the cost of precision. This measure can be used to find a compromise between precision and recall.

For indexing methods we can calculate the F-measure of the recall and reduction ratio, to find a compromise between effectiveness and efficiency [18].

2.8.5 Conclusion

Quality and complexity measures that are not based on the number of true negatives can be used to measure the efficiency and effectiveness of the steps in a data matching algorithm. For the indexing step we will calculate the reduction ratio, recall and their F-measure. For the classification step we will calculate the precision, recall and their F-measure.

Usually not all true matches are found in the indexing step. The true matches that are missed in the indexing step will also be missed in the classification step, as these pairs are not considered for classification. The results of the classification step of a data matching process are therefore affected by the indexing step. Christen [4] recommends calculating quality measures for the classification step based on the whole dataset if possible, to measure its effectiveness separately from the indexing step.

3 Research questions

The main research question for this thesis is as follows:

- **What is an effective method for matching invoices with orders, with a level of confidence?**

To answer this main research question the following sub-questions will be answered:

1. **Which data pre-processing methods are needed for 3-way matching?**
2. **Which indexing methods are effective for 3-way matching?**
3. **What methods can be used for determining a level of confidence that a PO and invoice match, by comparing them on a header-level?**
4. **How can the effectiveness of a 3-way matching algorithm be determined?**

4 Matching scenario's

Several scenario's should be taken into account when matching invoices to PO's and goods receipts. For example, PO's and invoices can have a different total amount due to additional costs or price differences in unit prices. Invoices can match partially to a PO due to partial deliveries, or they can match to multiple PO's when multiple orders are combined on a single invoice. For this thesis we will focus on the scenario where one invoice matches to one PO, for completeness we will also describe the other scenario's. In future work, we will look into methods to also support the other scenario's.

4.1 Exact matching on header-level

In the most simple scenario, an invoice can be matched exactly on header-level to one PO and one goods receipt. The PO and goods receipt match up when all ordered goods have been delivered in the correct quantities. The invoice and PO both contain header-level data that can be used to determine if they match. If the PO number, supplier name, administration and total price amount on these documents match exactly, the PO and invoice are matched on header-level.

Table 1 shows an example of a PO, goods receipt and invoice where all goods have been delivered and the PO and invoice match exactly on header-level.

PO nr. #123

line	article nr.	quantity	unit price	total amount
1	11	15	5	75
2	12	10	20	200
3	13	20	10	200
				475

GR #987

line	article nr.	quantity
1	11	15
2	12	10
3	13	20

Invoice #1007

PO nr. #123

line	article nr.	quantity	unit price	total amount
...				
				475

Table 1: exact header matching

4.2 Price-differences within tolerances on header-level

PO's and invoices can be matched even if they have a small difference in the total price amount. The price difference should then be within an absolute and relative tolerance. The absolute tolerance is defined as the maximum allowed price difference and the relative tolerance is defined as the maximum allowed percentage difference.

Table 2 shows an example of a PO, goods receipt and invoice where all goods have been delivered. The total amount on the PO and invoice does not match exactly, but the difference is within both an absolute tolerance of 10 and a relative tolerance of 2%.

PO #123

line	article nr.	quantity	unit price	total amount
1	12	10	20	200
2	13	20	10	200
				400

GR #987

line	article nr.	quantity
1	12	10
2	13	20

Invoice #1007

PO nr. #123

line	article nr.	quantity	unit price	total amount
...				
				405

Table 2: header matching with price difference within tolerances

4.3 Price-differences within tolerances on line-level

Sometimes invoices can't be matched on header-level, these should then be compared to PO's and goods receipt on line-level. When comparing invoices and PO's on line-level, the article number or description, the unit price and the quantities are compared. Lines can be matched if they have a small difference in the unit price. This price difference should also be within an absolute and relative tolerance.

Table 3 shows an example of a single line on a PO, goods receipt and invoice. Each of the goods on that line have been delivered, but the total amounts for that line do not match up between the PO and invoice. The difference is within both an absolute tolerance of 10 and a relative tolerance of 2%.

PO #123

line	article nr.	quantity	unit price	total amount
...				
2	12	10	20	200
...				
				...

GR #987

line	article nr.	quantity
...		
2	12	10
...		

Invoice #1007

PO nr. #123

line	article nr.	quantity	unit price	total amount
...				
5	12	10	20,10	201
...				
				...

Table 3: line matching with price difference within tolerances

4.4 Additional costs

Invoices can contain additional costs such as freight or administration costs. These are usually not included on the PO, which can lead to a price difference in the total price amount. The additional costs can be included on the invoice on a separate line, or they can be added to an item line.

4.4.1 Additional costs on an item line

Table 4 shows an example of a PO, goods receipt and invoice where the invoice has additional costs included on an item line.

PO #123

line	article nr.	quantity	unit price	total amount
1	12	10	20	200
				200

GR #987

line	article nr.	quantity
1	12	10

Invoice #1007

PO #123

line	article nr.	quantity	unit price	costs	total amount
1	12	10	20	50	250
					250

Table 4: additional costs on an item line

4.4.2 Additional costs on a separate line

Table 5 shows an example of a PO, goods receipt and invoice where the invoice has additional costs included on a separate line.

PO #123

line	article nr.	quantity	unit price	total amount
1	12	10	20	200
				200

GR #987

line	article nr.	quantity
1	12	10

Invoice #1007

PO #123

line	article nr.	quantity	unit price	costs	total amount
1	12	10	20	0	200
2				50	50
					250

Table 5: additional costs on a separate line

4.5 Discounts

Invoices can contain discounts for specific items on line-level. These can be either absolute or relative discounts. An absolute discount decreases the total line amount by a specific amount, while a relative discount decreases the total line amount by a specific percentage.

4.5.1 Absolute discounts

Table 6 shows an example of a PO, goods receipt and invoice with an absolute discount included on the PO and invoice.

PO #123

line	article nr.	quantity	unit price	discount	discount percentage	total amount
1	12	10	20	5	0	195
						195

GR #987

line	article nr.	quantity
1	12	10

Invoice #1007

PO nr. #123

line	article nr.	quantity	unit price	discount	discount percentage	total amount
1	12	10	20	5	0	195
						195

Table 6: absolute discounts

4.5.2 Relative discounts

Table 7 shows an example of a PO, goods receipt and invoice with a relative discount included on the PO and invoice.

PO nr. #123

line	article nr.	quantity	unit price	discount	discount percentage	total amount
1	12	10	20	20	10	180
						180

GR #987

line	article nr.	quantity
1	12	10

Invoice #1007

PO nr. #123

line	article nr.	quantity	unit price	discount	discount percentage	total amount
1	12	10	20	20	10	180
						180

Table 7: relative discounts

4.6 Missing or incorrect data

PO's can have missing or incorrect data due to OCR or typing errors. If the invoice has a missing or incorrect PO number, other header-level data can be used to find the corresponding PO and goods receipt. If the PO number on the invoice is incorrect due to OCR or typing errors the corresponding PO can be found by searching for PO's with a PO number that is very similar.

4.7 Partial deliveries

Sometimes multiple deliveries are needed for one PO. The supplier can then send separate invoices for each of these deliveries. Each of these invoices then matches partially to the PO.

PO #123

line	article nr.	quantity	unit price	total amount
1	11	15	5	75
2	12	10	20	200
3	13	20	10	200
				475

GR #987

line	article nr.	quantity
1	11	15
2	13	10

Invoice #1007

PO nr. #123

line	article nr.	quantity	unit price	total amount
1	13	5	10	50
2	11	15	5	75
				125

Table 8: partial deliveries

4.8 Multiple orders combined on one invoice

Sometimes a supplier creates an invoice combining multiple orders. This invoice then matches to multiple PO's.

PO #123

line	article nr.	quantity	unit price	total amount
1	11	5	5	25
2	12	10	20	200
				225

PO #124

line	article nr.	quantity	unit price	total amount
1	12	15	20	300
2	13	5	10	50
				350

GR #987

line	article nr.	quantity
1	11	5
2	12	10

GR #988

line	article nr.	quantity
1	12	15
2	13	5

Invoice #1007

PO nr. #123, #124

line	article nr.	quantity	unit price	total amount
1	11	5	5	25
2	12	25	20	500
3	13	5	10	50
				575

Table 9: multiple orders combined on one invoice

5 Proposed algorithm

For invoice matching we need to find which invoices and purchase orders refer to the same order. We want to find these document pairs efficiently, without having to compare each possible invoice and order pair in detail, while missing as few matching document pairs as possible.

In section 2 we have looked at the data matching process, which we will use as a basis for our invoice matching algorithm. The data matching process consists of methods for finding related documents among large datasets efficiently. The data matching process consists of the following steps: pre-processing, indexing, pairwise comparisons and classification.

We also looked at parallelization methods for the data matching process. These will not be implemented in our proof-of-concept, but they are interesting for larger datasets. Finally, we looked at quality and complexity measures for the indexing step, as well as quality measures for the classification step. These will be used to measure the efficiency and effectiveness of our algorithm on the provided dataset.

5.1 Pre-processing methods

The first step in the data matching process is pre-processing of the data. This is also the first step in our invoice matching algorithm. In this step data is first cleaned and then transformed into a format such that attributes can be compared between documents more easily.

For textual attributes such as supplier names we can remove extra whitespaces and special characters such as dots and parentheses. We can also make these values lowercase such that inconsistencies in upper-, and lower-casing between the invoices and purchase orders does not influence the similarity values.

For supplier names we can also remove common abbreviations such as 'b.v.' or 'bv', as these can be written differently or missing, which can lead to different values for the same supplier name. For example, with 'Easy Systems b.v.' and 'EASY SYSTEMS' we get a very low similarity value if we calculate the Levenshtein similarity, even though both names refer to the same company. After our preprocessing methods both names are written the same, 'easy systems', resulting in a similarity value of 1 and an exact match of the values.

Invoices sometimes contain multiple PO-numbers. We will split up these PO-numbers such that they can be used separately to generate blocking keys, and so they can be used for the header-level pairwise comparisons.

For example, an order might contain the ordernumber '1007005' and an invoice might contain the ordernumbers '1007003,1007005'. For the invoice, we would then split the ordernumber value into a list containing the values '1007003' and '1007005', such that we can generate two separate blocking keys and compare each value separately to the ordernumber value on the order.

5.2 Indexing methods

The next step in the data matching process is the indexing step. In subsection 2.3 we have looked at several indexing methods. With these methods we generate pairs of documents that are likely to match based on their attribute values, reducing the total number of pairs of documents we need to compare.

We will use traditional blocking for the indexing step, as this is a simple yet effective method. We will compare several blocking keys to find out which are most effective for matching invoices with purchase orders. These blocking keys will be based on header-level values: the PO number, the supplier name and the administration.

First we will compute blocking keys for each of the purchase order headers, and store them in an inverse index array. Then, for each of the invoice headers, we compute its blocking key and use it to find the purchase orders in the inverse index array with the same blocking key. Then for each of those purchase orders we generate the invoice and purchase order pairs.

5.3 Pairwise comparisons

In subsection 2.4 we looked at several comparison methods, which can be used to calculate similarity values for text values as well as numerical and date values. For each of the document pairs generated by the indexing step a comparison vector is generated, by calculating similarity values based on corresponding attributes on the invoice and purchase order headers.

For fields with a text value, such as the supplier name, we will calculate a similarity value based on the Levenshtein distance. For fields with numerical values, such as the total amount, a similarity value will be calculated based on the relative numerical difference.

5.4 Match probability

The last step in the data matching process we looked at is classification of document pairs based on their comparison vectors. In subsection 2.5

we looked at several classification methods that have previously been used for data matching. For our invoice matching problem, we want to classify document pairs as either matches or non-matches, as well as give a level of confidence for the classification.

5.4.1 Naive Bayes classification

For each of the comparison vectors we will calculate a match probability using a Naive Bayes classifier. These probabilities will be calibrated as described in 2.6. The calibrated probabilities will then be used to determine the confidence levels for calculated match probabilities.

Using Naive Bayes theorem [2] we can calculate the probability that documents have a class y_j, based on their comparison vector X as follows:

$$P(y_j|X) = \frac{P(X|y_j)P(y_j)}{P(X)}$$

For classification we want to assign class y_i to a comparison vector X such that $P(y_j|X)$ is maximized. $P(X)$ is constant for all classes, therefore we can assign the class y_j to X for which $P(X|y_j)P(y_j)$ is maximized, if we just want to classify the comparison vectors. For this thesis we are interested in the probabilities $P(y_j|X)$, specifically the match probability $P(match|x)$ for each comparison vector x.

For our classification problem there are two classes: match and non-match. Each document pair has exactly one of these two classes. Therefore we can calculate the match probability for a given comparison vector X as described in [2]:

$$P(match|X) = \frac{P(X|match)P(match)}{P(X|match)P(match) + P(X|non\text{-}match)P(non\text{-}match)}$$

Assuming the individual attributes x_i, $i = 1..p$ in X are independent, this can be rewritten as follows:

$$P(match|X) = \frac{\prod_{k=1}^{p} P(x_k|match)P(match)}{\prod_{k=1}^{p} P(x_k|match)p(match) + \prod_{k=1}^{p} P(x_k|non\text{-}match)P(non\text{-}match)}$$

The conditional probabilities used by the Naive Bayes classifier will be estimated from a training dataset.

5.4.2 Laplacian smoothing

The probability $P(x_k|y_i)$ can be calculated based on a dataset with instances X and classes Y as follows:

$$P(x_k|y_i) = \frac{|\{x \in X : x \textit{ has value } x_k \textit{ and class } y_i\}|}{|\{x \in X : x \textit{ has class } y_i\}|}$$

If no instance x with a value x_k and a class of match occurs in the dataset, the probability $P(x_k|match)$ will receive a value of zero. Laplacian smoothing is used to make each conditional probability non-zero. Without smoothing, combinations of comparison vector values and classes that do not appear in the dataset will cause the conditional probabilities they occur in to become zero, which can result in classification errors.

With Laplacian smoothing the conditional probabilities are calculated as follows:

$$P(x_k|y_i) = \frac{1 + |\{x \in X : x \textit{ has value } x_k \textit{ and class } y_i\}|}{|Y| + |\{x \in X : x \textit{ has class } y_i\}|}$$

5.4.3 Conditional probabilities for continuous variables

Conditional probabilities are calculated based on values in comparison vectors. For exact matching these values are either 0 or 1. The conditional probabilities can then be calculated for both classes. When using approximate matching these are continuous values between 0 and 1, therefore we can not calculate the conditional probabilities for all possible values. Instead we can estimate probability density functions for these attributes as described in [2].

We calculate the mean μ_i and standard deviation σ_i for each of the attributes with continuous values, for each of the different classes i. The conditional probabilities for an attribute value x given class y_i are then calculated as follows:

$$f(x|y_i) = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu_i}{\sigma_i})^2}$$

5.4.4 Level of confidence

With the estimated conditional probabilities for our Naive Bayes classifier we can calculate match probabilities for the comparison vectors in our training dataset. These match probabilities are then calibrated as follows to obtain the levels of confidence for match classifications. The vectors are sorted on

the calculated match probability and divided into 10 equal sized bins. For each bin the upper and lower match probability is determined, as well as the fraction of true matches.

For comparison vectors in the test set we can then calculate the match probability using the Naive Bayes classifier. We can determine the level of confidence as the calibrated match probability for the bin that corresponds to the vectors match probability.

6 Experiments

6.1 Dataset

The dataset used for the experiments consists of several CSV files containing real-world invoice and purchase order data. The data is retrieved from a SQL database. This data was stored by a commercial purchase-to-pay software solution which has performed some validation steps to verify and correct values. This could influence our results, as dealing with incorrect values is part of our research, however we do not have access to the original data. Besides verifying and correcting values, id's and codes for the supplier and the company that placed the order were added. These will not be used to create blocking keys or calculate similarity values. The dataset contains the following files:

- **invoice headers**: header-level data from invoices
- **order headers**: header-level data from purchase orders
- **invoice lines**: line-level data from invoices
- **order lines**: line-level data from purchase orders
- **match results**: id's for matching invoice and purchase order headers, and id's for matching invoice and purchase order lines

Table 10 shows the numbers of rows and columns in the CSV files of the original dataset. The file with the match results will be used as the ground truth for our experiments. Document pairs that appear in this file will be considered matches, document pairs that do not appear in this file will be considered non-matches.

	nr. of rows	nr. of columns
invoice headers	3608	217
order headers	2456	80
invoice lines	3438	61
order lines	5020	83
match results	5000	14

Table 10: Numbers of rows and columns in the original dataset

6.2 Data cleaning

Before we used the dataset for our experiments we performed some data cleaning. We removed the rows that were incorrectly formatted. From the file with the matching results we removed the rows with id's for invoices or purchase orders that were not present in the files with header data. From the file with invoice header data we remove the rows for invoices that do not match to any purchase order. We also removed all the columns from the files that we do not need for our experiments. Table 11 shows the numbers of rows and columns in the cleaned dataset.

	nr. of rows	nr. of columns
invoice headers	2201	15
order headers	2397	15
match results	2292	3

Table 11: Numbers of rows and columns in the cleaned dataset

We also performed the following pre-processing steps. The values in the field with PO numbers on the invoice headers are split into a list of individual PO numbers. Supplier names are made lower-case, and special characters and abbreviations commonly used in company names are removed.

The cleaned dataset contains the following header-level data for each invoice:

- **Invoice ID**: unique id used to distinguish between invoices in the dataset
- **Organization element (ID, code & name)**: id, code and name of the holding or company to which the invoice is sent
- **Company (ID, code & name)**: id, code and name of the company to which the invoice is sent
- **Order numbers**: list of order numbers referenced on this invoice
- **Supplier (ID, code & name)**: id, code and name of the company which has sent this invoice
- **Net sum**: total price amount excluding taxes

- **Gross sum**: total price amount including taxes
- **Tax sum**: total tax amount
- **Currency code**: code for the specific currency used for price amounts on the invoice (EUR, GBP, ...)

The cleaned dataset contains the following header-level data for each order:

- **Order ID**: unique id used to distinguish between orders in the dataset
- **Order number**: identifier added by the company that sent this order
- **Invoicing supplier (ID, code & name)**: id, code and name of the company that will send the invoice corresponding to this order
- **Company (ID, code & name)**: id, code and name of the company which has sent this order
- **Main supplier (ID, code & name)**: id, code and name of the company to which the order is sent
- **Net sum**: total price amount excluding taxes
- **Gross sum**: total price amount including taxes
- **Tax sum**: total tax amount
- **Currency code**: code for the specific currency used for price amounts on the order (EUR, GBP, ...)

6.3 Data analysis

Before running any experiments we performed some data analysis on the dataset. First we calculated the percentages of missing values for each of the attributes for the order headers and the invoice headers. Next we calculated the percentages of exact matches for values of corresponding attributes between the invoice and order headers. Values for the net sum, gross sum and tax sum are considered missing if they have a value of 0. We want to create blocking keys and calculate similarity values based on attributes that mostly have no missing values for both the invoice and order headers. For creating the blocking keys we want to use those attributes for which most of the corresponding values match exactly between the matching documents.

Table 12 shows the percentages of missing values per attribute for the order headers. Six of the attributes have no missing values. The other attributes have between 1,5 % and 98,9 % missing values. Table 13 shows the percentages of missing values per attribute for the invoice headers. Only the tax sum attribute has missing values. Values for the tax sum are missing for 45,89 % of the invoice headers, while they are missing for 98,87 % of the order headers. Company name has 58,49 % missing values in the order headers.

For this specific dataset each order header either has the same value the invoicing supplier name and the main supplier name, a missing value for the invoicing supplier name or missing values for both names. Therefore we will not use the invoicing supplier name for creating blocking keys and calculating similarity values. The values for the organization element and company name attribute on the invoice headers always have the same value, so we will use only the company name for creating blocking keys and calculating similarity values.

attribute name	% of missing values
Order number	0
Invoicing supplier id	52,19024
Invoicing supplier code	0
Invoicing supplier name	4,046725
Company id	0
Company code	0
Company name	58,48978
Main supplier id	59,741344
Main supplier code	0
Main supplier name	1,5018773
Net sum	18,481436
Gross sum	20,06675
Tax sum	98,87359
Currency code	0

Table 12: Percentages of missing values per attribute for order headers

attribute name	% of missing values
Organization element id	0
Organization element code	0
Organization element name	0
Company id	0
Company code	0
Company name	0
Order numbers	0
Supplier id	0
Supplier code	0
Supplier name	0
net sum	0
gross sum	0
tax sum	45,888233
currency code	0

Table 13: Percentages of missing values per attribute for invoice headers

Table 14 shows percentages of exactly matching values for corresponding attributes between matching invoice and order headers. For the order numbers we consider the values to match exactly if the invoice header contains the order number on the order header in its list of order numbers. The order numbers match exactly for 99,78 % of the matching documents. However, these values have already been verified and corrected. Using the order number for indexing and classification will not give realistic results, as the percentage of exactly matching order numbers is likely much lower for matching documents in an original dataset.

The values for the currency code match exactly for 99,08 % of the matching documents. If we use this attribute as a blocking key, candidate matching pairs will be generated for most matching document pairs. However, since the currency code attribute does not have many distinct values, this blocking key will likely result in a relatively large number of candidate matching pairs.

Invoice attribute name	Order attribute name	% of exact matches
Organization element id	Company id	100
Organization element code	Company code	100
Organization element name	Company name	45,898777
Company id	Company id	100
Company code	Company code	100
Company name	Company name	45,898777
Order numbers	Order number	99,78185
Supplier id	Invoicing supplier id	51,61431
Supplier code	Invoicing supplier code	51,61431
Suplier name	Invoicing supplier name	67,495636
Supplier id	Main supplier id	45,898777
Supplier code	Main supplier code	51,61431
Supplier name	Main supplier name	70,11344
Net sum	Net sum	21,727749
Gross sum	Gross sum	8,944154
Tax sum	Tax sum	0
Currency code	Currency code	99,08377

Table 14: Percentages of exactly matching values for corresponding header level attributes for matching document pairs

6.4 Indexing

We implemented traditional blocking for the indexing step. We compared the following blocking keys:

- order number
- supplier name
- company name
- currency code

We calculated the following quality and complexity measures based on the indexing results for each of the different blocking keys:

- reduction ratio
- pairs completeness
- f-measure of the reduction ratio and pairs completeness

6.5 Comparison methods

Comparison vectors are created by calculating similarity values for the corresponding attributes on the invoice and order headers. We will calculate similarity values for the following attributes:

- **ordernumber**: For the ordernumber values we will use exact matching. If an invoice header contains multiple PO numbers we will compare each of these to the PO number on the order header individually. We then consider the values to match exactly if the invoice header contains any PO number that matches exactly to the PO number on the order header.
- **supplier name**: For the supplier name we will compare the preprocessed values and calculate a similarity value based on Levenshtein distance.
- **company name**: For the company name we will compare the preprocessed values and calculate a similarity value based on Levenshtein distance.
- **net sum**: Similarity values for the net sum will be calculated based on the relative numerical distance, with a maximum percentage difference of 20%.
- **currency code**: Similarity values for currency codes will be based on exact matching.

6.6 Classification

We will use the complete set of invoice and order pairs generated from the cleaned dataset to create and test the Naive Bayes classifier. For each document pair a comparison vector is calculated, and their class (match or non-match) is added. A document pair gets a class of match if a line with both the invoice id and order id is present in the matching results file, otherwise it gets a class of non-match.

This set of comparison vectors is then split up into a training and a test set, both containing half of the comparison vectors. The training set will be used to calculate the conditional probabilities needed for the Naive Bayes classifier. With this classifier we will then classify all document pairs in the test set and gather their calculated match probabilities. We will then use these results to calculate quality measures for classification.

Table 15 shows the probabilities calculated on the training set. For this dataset almost all matching invoices and purchase orders have a matching PO number while most non-matching document have no matching PO number, which results in values for the conditional probabilities that are either very high or very low. Similarly, for most matching documents the currency codes match exactly.

$P(match)$	0.00043880608
$P(non\text{-}match)$	0.9995612
$P(ordernumber\ match \mid match)$	0.9982744
$P(ordernumber\ non\text{-}match \mid match)$	0.0017256256
$P(ordernumber\ match \mid non\text{-}match)$	0.00000037942817
$P(ordernumber\ non\text{-}match \mid non\text{-}match)$	0.99999964
$P(currency\ code\ match \mid match)$	0.9887834
$P(currency\ code\ non\text{-}match \mid match)$	0.011216566
$P(currency\ code\ match \mid non\text{-}match)$	0.45030382
$P(currency\ code\ non\text{-}match \mid non\text{-}match)$	0.54969615

Table 15: calculated probabilities on the training set

For the attributes company name, (main) supplier name and net sum we calculate approximate similarity values. To use these values in our Naive Bayes classifier, we calculate their mean and standard deviations for matching document pairs and non-matching document pairs in the training set. These are then used to calculate probabilities using probability density functions.

6.7 Level of confidence

To obtain a level of confidence for pairs of documents classified as a match, we will calibrate the match probabilities as follows. We sort the comparison

	match		non-match	
	mean	st.dev.	mean	st.dev.
company name	0.44411495	0.49474573	0.23396295	0.3902542
(main) suppliername	0.89104575	0.22310196	0.20167556	0.16481934
netsum	0.27572972	0.4377369	0.01884554	0.11034476

Table 16: mean and standard deviation of similarity values calculated on the training set

vectors in the training set, that are predicted matches, in descending order on their predicted match probability. We then divide them into 10 equal sized bins, and calculate the fraction of true matches per bin. This fraction is the calibrated match probability, which we will use as our level of confidence.

7 Results

Table 17 shows how the numbers of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) are determined for the indexing and classification step. For the analysis of the results of the indexing step, the generated document pairs are considered predicted matches (positives). The document pairs that were not generated by the indexing step are considered predicted non-matches (negatives).

	predicted	
actual	match	non-match
match	TP	FN
non-match	FP	TN

Table 17: Confusion matrix for matching results

7.1 Indexing methods

Table 18 shows the numbers of true positives, true negatives, false positives and false negatives for traditional blocking as indexing method using different attributes as blocking key. Table 19 shows the reduction ratio, recall and their f-measure for these different blocking keys. For this specific dataset the order number results in a high reduction ratio as well as a high recall when used as a blocking key for the indexing step. As the order numbers have been verified and corrected, this result is not representative for datasets with original header-level data.

Looking at the indexing results for the company name, (main) supplier name and the currency code as blocking keys we can see the (main) supplier name results in the highest reduction ratio as well as highest F-measure, and a recall of 0,70. The currency code results in the highest recall and the lowest reduction ratio, likely due to the relatively small number of distinct values for the currency codes.

7.2 Matching results

Table 20 shows the numbers of true positives, true negatives, false positives and false negatives for classification on the test dataset using a Naive

blocking key	TP	TN	FP	FN
ordernumber	2287	5273504	1	5
company name	1052	4179453	1094052	1240
(main) supplier name	1607	5134036	139469	685
currency code	2271	2893280	2380225	21

Table 18: Indexing results for different blocking keys

blocking key	reduction ratio	recall	F-measure
order number	0,9995663	0,99781847	0,9986916
company name	0,79242873	0,45898777	0,5812855
(main) supplier name	0,97325975	0,7011344	0,8150839
currency code	0,5484102	0,9908377	0,7060402

Table 19: quality and complexity measures for indexing with different blocking keys

Bayes classifier, both with and without indexing. Table 21 shows the quality measures for these classification results.

For all these indexing methods the precision for classification is low. This is due to the large number of false positives. False positives occur when non-matching document pairs are classified as a match. This probably means that our classifier is not able to properly distinguish between matching documents and non-matching documents that are similar. This is not resolved by indexing as these similar non-matching documents often have the same blocking key and thus will not be filtered out by indexing.

The recall is similar for classification without indexing and with indexing using (main) supplier name or currency code as blocking key. With indexing using the company name as a blocking key the recall is much lower. This likely means the company name often does not match exactly for matching documents and therefore many matching document pairs are filtered out in the indexing step when using company name as a blocking key. Due to the low precision the F-measure is low for classification with and without indexing.

blocking key	TP	TN	FP	FN
no indexing	887	2528489	107054	270
company name	506	2620829	14714	651
(main) supplier name	829	2557464	78079	328
currency code	887	2542774	92769	270

Table 20: classification results on the test set without indexing and with indexing using different blocking keys

blocking key	precision	recall	F-measure
no indexing	0,008217452	0,76663786	0,016260609
company name	0,03324573	0,43733793	0,06179398
(main) supplier name	0,010505905	0,7165082	0,020708174
currency code	0,00947083	0,76663786	0,018710515

Table 21: quality measures for classification on test set without indexing and with indexing using different blocking keys

7.3 Level of confidence

Table 22 shows the calibrated match probabilities for classification. With the calibrated match probabilities we can estimate the level of confidence that a match classification is correct. Most predicted matches have a very high match probability, while the calibrated match probability is very low for all ranges of match probabilities. This is due to the high numbers of false positives, even for document pairs classified as a match with high match probabilities. Therefore the level of confidence a document pair matches is low even when the calculated match probability is high.

lower match probability	upper match probability	calibrated match probability
1	1	0,027052065
0,9999606	1	0,003242542
0,9927696	0,9999606	0,00046322
0,9455181	0,9927696	0,03149898
0,9122687	0,94548285	0,001574949
0,9122687	0,9122687	0,003335186
0,9122687	0,9122687	0,003335186
0,9122687	0,9122687	0,003335186
0,9122687	0,9122687	0,004168983
0,500206	0,9122687	0,004168983

Table 22: calibrated match probabilities for comparison vectors in the training set classified as a match

8 Conclusions

8.1 Pre-processing methods

We have performed pre-processing on the company and supplier names on the order and invoice headers, as well as on the ordernumbers on the invoice headers. For the company and supplier names we have removed special characters, made them lowercase and removed common abbreviations such as 'bv'. This was needed because company and supplier names can be written differently on the invoice and the order header. Usage of upper-, and lowercase and special characters such as dots can differ, and a common abbreviation can be present on one document and missing on the other.

The values in the ordernumbers field on the invoice headers have been split into lists of separate ordernumbers. This was needed to be able to compare the ordernumber on an order header to each individual ordernumber on an invoice header.

Depending on the header-level attributes used for indexing and classification, we might need other data pre-processing methods. For example, if we want to use date values we might need to extract the day, month and year into separate fields, to be able to compare date values more efficiently.

8.2 Indexing methods

With indexing we generate only those document pairs that are likely to match, based on their blocking keys. We have used traditional blocking and compared results for several blocking keys. Effective indexing methods should reduce the total number of document pairs that are compared as far as possible while missing as few matching document pairs as possible.

We use the reduction ratio to measure how effective an indexing method is at reducing the total number of documents pairs that are generated for comparison. To measure what fraction of all matching document pairs are generated by the indexing step, we use recall. A higher recall means we miss fewer matching documents. We calculate the F-measure of the reduction ratio and recall to determine the overall effectiveness of an indexing step, based on these two measures.

We have implemented traditional blocking and compared the effectiveness of this method for several blocking keys. On our dataset we get a high reduction ratio and a decent recall using the supplier name as a blocking key. More research is needed to find an indexing method that achieves both high reduction ratio as well as high recall.

8.3 Level of confidence

We can use a Naive Bayes classifier to classify document pairs as a match or non-match. Our Naive Bayes classifier calculates a match probability for each document pair that is classified. By calibrating the match probabilities, we can use this match probability to determine the level of confidence a document pair classified as a match is actually a match.

We calibrated the match probabilities generated on the training set by our classifier. The levels of confidence for our classifier are low for all ranges of match probabilities. This is due to the high numbers of false positives, even for higher match probabilities. If we can improve the precision of our classifier we could achieve higher levels of confidence.

We expect the number of false positives is high due to similar orders that are sent to the same supplier periodically by the same company, resulting in many similar order and invoice pairs. One way to distinguish between these pairs is by header-level date values such as the creation date of the order or the due date on the invoice.

8.4 Quality measures

In the literature we have found several quality measures recommended for determining the effectiveness of the data matching process. For the indexing step we can use the reduction ratio, recall and their F-measure. The goal of the indexing step is to reduce the total number of document pairs that need to be compared, which is measured by the reduction ratio. We also need to make sure the indexing step misses out on as few matching document pairs as possible, which is measured by the recall.

For the classification step we can use precision, recall and their F-measure. With precision we measure what fraction of document pairs classified as a match actually are matches. With recall we measure the fraction of matches that are correctly classified as a match.

8.5 Answers to research questions

To answer our main research question we defined several sub-questions. The sub-questions and their answers are as follows:

1 **Which data pre-processing methods are needed for 3-way matching?**

Invoices and orders usually contain header level fields containing company names, such as the name of the supplier and the name of the company who receives the invoice. For company names we remove special characters, make them lowercase and remove common abbreviations. For the ordernumber field on invoice headers we separate the invididual ordernumbers into a list of ordernumbers.

For 3-way matching we need to compare the ordernumbers, supplier names and company names. As each ordernumber should be compared individually, splitting them into a list is necessary. For supplier and company names the pre-processing methods are necessary as they help remove differences in writing between values that refer to the same name.

2 **Which indexing methods are effective for 3-way matching?**

We have implemented traditional blocking as an indexing method. Depending on the blocking key used, we can achieve either high reduction ratio or high recall. More research is needed to find an indexing method that achieves both high reduction ratio and high recall on datasets with invoice and order headers. We can look into other blocking keys and other indexing methods to find which methods are most effective.

3 **What methods can be used for determining a level of confidence that a PO and invoice match, by comparing them on a header-level?**

We can compare corresponding attribute values on the invoice and order headers, and compute similarity values using exact matching or approximate matching. With these similarity values we can make comparison vectors, which we can classify as a match or non-match using a classifier. We have used a Naive Bayes classifier, which outputs a match probability. By calibrating the match probabilities we can determine a level of confidence for document pairs classified as a match.

4 **How can the effectiveness of a 3-way matching algorithm be determined?**

We have based our algorithm on the data matching process. The goals of our 3-way matching algorithm are similar to that of the data matching process: efficiently finding pairs of documents that are considered matches. Therefore we can use quality measures designed for determining the effectiveness of data matching processes. For the indexing step we use the reduction ratio, recall and their F-measure. For the classification step we use the precision, recall and their F-measure.

Now that we have answered our sub-questions we can answer our main research question:

- **What is an effective method for matching invoices with orders, with a level of confidence?**

We can base our algorithm for matching invoices with orders on the data matching process. By using indexing methods we can reduce the total number of document pairs that need to be compared.

Classification methods can be used to classify document pairs as matches or non-matches. With a Naive Bayes classifier we can calculate a match probability for document pairs. By calibrating match probabilities we can determine a level of confidence based on the match probability calculated for a document pair classified as a match.

For such an algorithm to be effective the indexing method should have both high reduction ratio and high recall, and the classification method should have high precision and high recall.

9 Future work

In this thesis we have researched methods for efficiently matching invoices and orders, as well as methods for determining a level of confidence for match classifications. Based on these methods we have proposed an algorithm and developed a proof-of-concept. We also tested our proof-of-concept with a dataset containing real invoice and order header data. In future work we can look into methods to improve our algorithm, as there is room for improvement, especially in the classification step.

9.1 More realistic dataset

First of all, we would like to test our algorithm with a more realistic dataset. Pre-processing steps had already been applied to the dataset we used for testing our proof-of-concept. This was especially noticeable in the ordernumber values, as nearly all matching documents had matching ordernumbers.

On datasets where values have not been corrected or added, we expect these ordernumbers to both be missing more often as well as contain incorrect values.

9.2 Pre-processing methods

Including other attributes could improve the indexing and classification step. We could use other attributes to create blocking keys, as well as calculate similarity values. If we choose to include other attributes we might need other pre-processing methods depending on the type of values. For example, with date values we might need to extract the day, month and year values into separate fields.

For pre-processing, we can also look into validation methods. For example, we can check if company names are written correctly, or whether combinations of certain fields are valid. We could perform experiments to determine if correcting such values can improve our matching results.

9.3 Indexing methods

For our proof-of-concept we have implemented traditional blocking. In future work we could implement other indexing methods to compare their effectiveness. We can also try to improve our indexing results by defining other blocking keys.

9.4 Classification methods

We have implemented a Naive Bayes classifier for classification of document pairs. In future work we would like to improve our classification results. We could add similarity values based on other attributes to our comparison vectors. We could also add more detailed comparisons based on line-level data.

We can also compare other types of classifiers. In previous work machine learning and active learning methods have been used successfully for data matching.

9.5 Matching scenario's

For this thesis we have focussed on the most basic matching scenario, where one invoice matches completely to one order. In further research we would like to support other scenario's as well. For partial matching and combined invoices we will have to look into methods for line-level matching.

9.6 Parallelization

Several methods for parallelization of data matching processes have been proposed in previous work. These could be interesting for our algorithm if computation time becomes an issue.

References

[1] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. In *Proc. of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning (ECML 2000), May*, 2000.

[2] D. Berrar. Bayes' theorem and naive bayes classifier. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, 403, 2018.

[3] X. Chen, Y. Xu, D. Broneske, G. C. Durand, R. Zoun, and G. Saake. Heterogeneous committee-based active learning for entity resolution (healer). In *European Conference on Advances in Databases and Information Systems*, pages 69–85. Springer, 2019.

[4] P. Christen. *Data Matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer, 2012.

[5] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, 2012.

[6] P. Christen and K. Goiser. Quality and complexity measures for data linkage and deduplication. In *Quality Measures in Data Mining*, pages 127–151. Springer, 2007.

[7] P. Christen, D. Vatsalan, and Q. Wang. Efficient entity resolution with adaptive and interactive training data selection. In *2015 IEEE International Conference on Data Mining*, pages 727–732. IEEE, 2015.

[8] V. Christen, P. Christen, and E. Rahm. Informativeness-based active learning for entity resolution. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 125–141. Springer, 2019.

[9] S. J. Delany, P. Cunningham, and L. Coyle. An assessment of case-based reasoning for spam filtering. *Artificial Intelligence Review*, 24(3):359–378, 2005.

[10] S. J. Delany, P. Cunningham, D. Doyle, and A. Zamolotskikh. Generating estimates of classification confidence for a case-based spam filter.

In *International Conference on Case-based Reasoning*, pages 177–190. Springer, 2005.

[11] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[12] L. Gagliardelli, G. Simonini, D. Beneventano, and S. Bergamaschi. Sparker: Scaling entity resolution in spark. In *EDBT 2019: 22nd International Conference on Extending Database Technology*. PRT, 2019.

[13] L. Gazzarri and M. Herschel. Towards task-based parallelization for entity resolution. *SICS Software-Intensive Cyber-Physical Systems*, 35(1):31–38, 2020.

[14] L. Kolb, H. Köpcke, A. Thor, and E. Rahm. Learning-based entity resolution with mapreduce. In *Proceedings of the third international workshop on Cloud data management*, pages 1–6, 2011.

[15] L. Kolb and E. Rahm. Parallel entity resolution with dedoop. *Datenbank-Spektrum*, 13(1):23–32, 2013.

[16] L. Kolb, A. Thor, and E. Rahm. Multi-pass sorted neighborhood blocking with mapreduce. *Computer Science-Research and Development*, 27(1):45–63, 2012.

[17] L. Kolb, A. Thor, and E. Rahm. Don't match twice: Redundancy-free similarity computation with mapreduce. In *Proceedings of the Second Workshop on Data Analytics in the Cloud*, pages 1–5, 2013.

[18] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.

[19] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.

[20] X. Li and B. Liu. Rule-based classification, 2014.

[21] M. Loog. Supervised classification: Quite a brief overview. *Machine Learning Techniques for Space Weather*, pages 113–145, 2018.

[22] V. Nasteski. An overview of the supervised machine learning methods. *Horizons. b*, 4:51–62, 2017.

[23] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*, pages 85–94, 2011.

[24] G. Papadakis, G. Mandilaras, L. Gagliardelli, G. Simonini, E. Thanos, G. Giannakopoulos, S. Bergamaschi, T. Palpanas, and M. Koubarakis. Three-dimensional entity resolution with jedai. *Information Systems*, 93:101565, 2020.

[25] E. H. Porter and W. E. Winkler. Approximate string comparison and its effect on an advanced record linkage system. In *Advanced record linkage system. US Bureau of the Census, Research Report*. Citeseer, 1997.

[26] V. Pronk, S. V. R. Gutta, and W. F. J. Verhaegh. Incorporating confidence in a naive bayesian classifier. In *International Conference on User Modeling*, pages 317–326. Springer, 2005.

[27] K. Qian, L. Popa, and P. Sen. Active learning for large-scale entity resolution. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1379–1388, 2017.

[28] K. Qian, L. Popa, and P. Sen. Systemer: A human-in-the-loop system for explainable entity resolution. 2019.

[29] B. Qin, Y. Xia, S. Prabhakar, and Y. Tu. A rule-based classification algorithm for uncertain data. In *2009 IEEE 25th International Conference on Data Engineering*, pages 1633–1640. IEEE, 2009.

[30] S. M. Randall, A. M. Ferrante, J. H. Boyd, and J. B. Semmens. The effect of data cleaning on record linkage quality. *BMC Medical Informatics and Decision Making*, 13(1):1–10, 2013.

[31] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, volume 62, pages 98–105, 1998.

[32] M. S. Schaeffer. *Accounts Payable: A Guide to Running an Efficient Department*. Wiley, 2004.

[33] M. S. Schaeffer. *Controller and CFO's Guide to Accounts Payable*. Wiley, 2007.

[34] V. S. Verykios, G. V. Moustakides, and M. G. Elfeky. A bayesian decision model for cost optimal record matching. *The VLDB Journal*, 12(1):28–40, 2003.

[35] Q. Wang, D. Vatsalan, and P. Christen. Efficient interactive training selection for large-scale entity resolution. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 562–573. Springer, 2015.

[36] W. E. Winkler. Methods for record linkage and bayesian networks. Technical report, Technical report, Statistical Research Division, US Census Bureau, 2002.

[37] W. E. Winkler. Matching and record linkage. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(5):313–325, 2014.

[38] W. E. Winkler and Y. Thibaudeau. *An application of the Fellegi-Sunter model of record linkage to the 1990 US decennial census*. Citeseer, 1991.

[39] B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Icml*, volume 1, pages 609–616, 2001.