Universiteit Leiden
The Netherlands

Opleiding Informatica

Clustering financial transactions
on inference-accelerator hardware

Vincent van Heertum

Supervisors:
Dr. Suzan Verberne & Erik Weenk MSc

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

10-07-2021

Abstract

This thesis compares different methods to accurately cluster similar and recurring financial transactions, which is similar to clustering short text documents, and examines the use of 'Tensor Cores' –dedicated matrix multiplication cores– in NVIDIA hardware to increase performance. To extract features from financial transactions, N-gram character tokenization is mainly used. We compare four similarity metrics: Cosine similarity, Kumar-Hassebrook similarity, Dice similarity and Dice squared similarity. We also compare four different clustering methods: perceptron-based classification similarity learning, hierarchical agglomerative clustering using single-linkage and complete-linkage and HDBSCAN. These methods are applied and compared on a dataset of 2880 financial transactions from a personal bank account covering multiple years and on a dataset of about 8000 transactions of the Financial Shared Service Centre of Leiden University, covering one month. In addition to this research, a prototype for a web-based clustering tool has been built.

Contents

1 Introduction

Financial transactions are often periodic or recurring in nature and thus often part of a group of similar transactions. Transactions are basically short text documents, do not contain information about relationships with other transactions and cannot be sorted in a simple way in order to group them.
Artificial intelligence and machine learning methods are widely used in the financial industry and in particular in transaction monitoring & fraud detection (see [AMI16]). Clustering may benefit these detection algorithms, because they often utilize anomaly/outlier detection. However, it may also be a useful step in a larger data analysis process by eliminating the need to process all transactions individually, possibly increasing performance and accuracy. Accurate clustering of recurring and similar financial transactions may also be used to reduce the time needed to manually manage (personal) finances or improve exploratory data analysis for large datasets. Large organisations in particular often have a large amount of financial transactions to administer. Since financial transactions are essentially short text documents, the process of clustering these is similar to clustering/analyzing computer log files (e.g. for cyber security purposes).
Inference-accelerator hardware, or an 'AI chip', is essentially a processing unit that is specialised in matrix-multiplications. Especially neural networks often use this operation when applying a trained neural network (a 'model') to classify data, this process is called inference. In recent years, several manufacturers started developing inference-accelerator hardware. In 2018 NVIDIA started to embed 'Tensor-Cores' in its professional and consumer-grade GPU's [NVI]. Google developed a 'Tensor Processing Unit' for internal usage since 2015 [JYP+17]. Intel's Myriad series and AMD's Radeon Instinct series are also dedicated processors aimed towards optimizing matrix multiplications [Int] [AMD]. Utilizing dedicated hardware like GPU's and inference accelerators when possible and useful, often improves the performance of algorithms by orders of magnitude.
This thesis examines which feature-extraction methods that can be applied to financial transactions are most important for accurate clustering. We compare the accuracy and performance of different similarity metrics and clustering algorithms. In this thesis we also propose a clustering method that could –to some extent– take advantage of inference-accelerator hardware. The following research questions will be answered:

1. How accurate can we cluster similar and recurring financial transactions using fixed-value parameters for its algorithms on multiple datasets?

2. How does the chosen similarity metric and clustering algorithm affect our accuracy?

3. What would be the estimated performance effect utilizing inference-accelerator hardware in our algorithm?

2 Background

According to a survey conducted in 2012 (see [Sab12]), clustering techniques for financial fraud detection mostly use traditional clustering algorithms: K-means, Hierarchical clustering (single-linkage and complete-linkage) and density-based clustering methods. K-means clustering using the Euclidian distance metric is most common. The paper by Larik [LH11] proposes a new clustering algorithm TEART, built upon the Euclidean Adaptive Resonance Theory. They do not present a comparable measurement of accuracy. Testing methods for anomalous transaction reporting tend to be difficult due to the private nature of the data being processed. However, Yang et al. [YLL+14] have successfully applied the DBSCAN algorithm to mark suspicious transactions at a financial institution. The experiments of Stojanovic. [SBHS+21] show that applying machine learning methods (neural network classification) successfully detect suspicious transactions with a true-positive rate of 0.71 and a true negative rate of 0.9995.

Since our experiment is fundamentally a lot like short text clustering and not about detection fraudulent transactions, it may be better to compare our results to the experiments done by Shrestha [SJD12]. They achieved an adjusted rand-index accuracy of 0.10 on their Cicling-2002 and LDC dataset using cosine-similarity and complete-linkage hierarchical agglomerative clustering. Their best clustering method was spectral clustering using cosine-similarity and achieved an adjusted rand-index of 0.29 on their LDC dataset, and scored 0.25 on their Cicling-2002 dataset. No research or experiments could have been found that applies these methods to a dataset similar to ours, consisting of financial transactions.

When it comes to GPU-accelerated clustering algorithms, several papers already exists. The approach for fast hierarchical clustering by Shalom et al. [SDT10] utilizes NVIDIA's graphics processors with the CUDA library for single- & complete-linkage clustering. The paper on distributed k-means clustering across multiple nodes equipped with GPU's by Takizawa and Kobayashi [TK06] shows significant performance improvements when calculating distances between datapoints in a parallel and distributed manner.

3 Methods

Our methodology consists of three parts. First we apply several feature-extraction methods to our input data (see subsection 3.1). Then, for each of the resulting feature-matrices we compute a similarity matrix (see subsection 3.2). A linear combination (see Eqn 1) of these similarity matrices form a single similarity matrix R, where A_i is a feature-matrix, w_i the weight corresponding to the feature and $b \in \mathbb{R}$ the bias. The feature-extraction methods that are most important for clustering similar financial transactions will have the largest weights in this equation, both positive and negative. A weight close to zero indicate that a certain feature-extraction method can be neglected. This optimization process is explained more detailed in subsection 3.3.1.

$$R = b + w_1 A_1 + w_2 A_2 + \cdots + w_n A_n \tag{1}$$

Different clustering algorithms can now be applied on the resulting similarity matrix R (see subsection 3.3). The distance matrix is computed from the similarity-matrix as follows:

$$D = 1 - R \tag{2}$$

In order to compare the accuracy of different clustering algorithms, we need to define a measurement of accuracy. After clustering, we compare the resulting adjacency matrix with a 'true' adjacency matrix created by manually clustering recurring and similar financial transactions. Our measurement of accuracy is defined in Eqn 3, where TP are the number of true positives, TN the number of true negatives, FP the number of false positives and FN the number of false negatives.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3}$$

In addition to the accuracy, we also compute 3 other metrics: the adjusted Rand index, the Calinski-Harabasz index and the Silhouette-score [LLX+10]. The Calinski-Harabasz index and Silhouette-score do not require labelled data.
The adjusted Rand index (see [HA85]) corrects the accuracy for the probability of a positive or negative outcome and results to a value between 0 and 1.
The Silhouette-score for our resulting clusters is the mean of the Silhouette-score for each cluster where the Silhouette-score for a cluster is defined as:

$$s = \frac{1}{NC} \sum_{x \in C_i} \frac{b(x) - a(x)}{\max(b(x), a(x))} \tag{4}$$

Where $a(x)$ is the mean distance between a transaction x and all other transactions in the same cluster, and $b(x)$ the mean distance between a transaction x and all other transactions in the next nearest cluster. C_i is a cluster and NC the number of clusters.
The Calinski-Harabasz index [LLX+10] is given by the formula:

$$c = \frac{\sum_i n_i d^2(c_i, c)/(NC - 1)}{\sum_i \sum_{x \in C_i} d^2(x, c_i)/(n - NC)} \tag{5}$$

Where c_i is the mean of the distance-values within a cluster i, c is the mean of the values in the entire distance matrix, n is the number of transactions, NC is the number of clusters, $d(x, y)$ is the difference between x and y, C_i is a cluster and n_i is the number of transactions in the cluster (see [sl]).

date	description	accountnr	counterparty_name	counterparty_IBAN	amount
23-07-2020	Pa...54	NL...81	Jumbo Le...IDEN NLD		-18,19
22-07-2020	Ar...12	NL...81	BOLCOM BV	NL...00	-89,95
21-07-2020	Fa...OM	NL...81	KPN - Mobiel	NL...13	-13,41
20-07-2020	Pa...G1	NL...81	Jumbo Le...IDEN NLD		-17,71
20-07-2020	MA...20	NL...81	BELASTINGDIENST	NL8...88	-195
17-07-2020	Pa...62	NL...81	PAKHUIS LEIDEN NLD		-63

Table 1: A small sample of our dataset.

3.1 Feature extraction

When extracting features from a financial transaction, we essentially represent a transaction as a vector. If we do this for all of our n transactions, we obtain a $n \times m$ feature-matrix T, where m is the number of features extracted by the method.

$$T_{n,m} = \begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,m} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n,1} & t_{n,2} & \cdots & t_{n,m} \end{pmatrix} \tag{6}$$

3.1.1 N-grams

N-gram character tokenization [MKM05] creates a multiset (or 'bag') of strings (or 'tokens') with fixed length n from a single input string. Using $n = 2$ we obtain a bag of 2-character strings called 'bigrams', N-grams with $n = 3$ are called trigrams.

`The quick fox` ⟶ `Th he e  q qu ui ic ck k  f fo ox`

Figure 1: Converting a 13-character sentence to 12 bigrams.

For short input strings, bigram tokenization ($n = 2$) has some advantageous properties over trigrams ($n = 3$); bigrams are less error-prone to typos and information about the original sequence of tokens largely remains. Bigram multisets also require a lot less memory space for storage and thus less processing; if our input string contains all letters of the latin alphabet, 26^3 possible trigrams exists while only 26^2 possible bigrams exist.
Due to the nature of our data, we first sanitize our input string by uppercasing and removing all non-latin characters except spaces. Next, we apply bigram tokenization on the `description`, `accountnr`, `counterparty_name`, `counterparty_IBAN` attributes for all financial transactions. This will result in the feature-matrix $T_{n,m}$ (see Eqn 6). Finally, tokens that exists in every bag are removed because this would negatively affect the similarity measurement in the next stage. When measuring the commonalities/similarity between two individual financial transactions, 'global' commonalities should be ignored (features that *every* transaction has). This feature-extraction method is applied on the following attributes (see Table 1): *description*, *counterparty_name*.

3.1.2 Dates

Several additional features can be extracted out of the *date* attribute of a financial transaction: day of the month, month, year, week of the year, day of the week, quarter of the year. Doing this for all transactions, we end up with 6 feature matrices $T_{n,m}$ (see Eqn 6) with $m = 1$.

3.1.3 Amount

We convert the amount of a transaction to a positive number and end up with a single feature matrix $T_{n,m}$ (see Eqn 6) with $m = 1$. The feature matrix cannot contain negative values, since some similarity metrics will not work correctly with negative values.

3.2 Similarity metric

A similarity or distance metric is a formula which describes how similar or distant two vectors are. A similarity matrix A of feature-matrix T is a $n \times n$ matrix where $a_{ij} = f(\vec{t_i}, \vec{t_j})$ and function f the chosen similarity-metric.

$$T_{n,m} = \begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,m} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n,1} & t_{n,2} & \cdots & t_{n,m} \end{pmatrix} \quad (6) \qquad A_{n,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} \quad (7)$$

The time complexity of computing a similarity or distance matrix is $O(\frac{1}{2}n^2m)$, where n is the number of transactions and m the number of features. Thus, when such a matrix is used in the clustering process, this is the most important computation to speed up.
In order to take advantage of inference-accelerator hardware, we need a similarity metric which is merely a dot product between it's two input (feature-)vectors. More specifically, the only operation between the elements of both vectors should be a multiplication. Several metrics with this property exist (see [Cha07]), for example: Kumar-Hassebrook's PCE, Cosine-similarity and Dice-Squared similarity. In our performance & accuracy comparison, we use these three and in addition the quantitative Dice coefficient, which cannot take advantage of inference-accelerator hardware. These similarity metrics can be seen in Table 2.
Where $\oplus_{\text{outer}}$ is the outer-sum of two vectors, $p_i \in \mathbb{R}^+$, $q_i \in \mathbb{R}^+$, $r_i = \vec{t_i} \cdot \vec{t_i}$ in vector $\vec{r}$ and $u_i = \sqrt{r_i}$ in vector $\vec{u}$. Note that vectors $\vec{q}$, $\vec{p}$ and thus feature-matrix T should not contain negative values.

$$\vec{u} = (u_1, u_2, \cdots, u_m)$$
$$\vec{v} = (v_1, v_2, \cdots, v_n)$$
$$\vec{u} \oplus_{\text{outer}} \vec{v} = A = \begin{pmatrix} u_1 + v_1 & u_1 + v_2 & \cdots & u_1 + v_n \\ u_2 + v_1 & u_2 + v_2 & \cdots & u_2 + v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m + v_1 & u_m + v_2 & \cdots & u_m + v_n \end{pmatrix} \quad (8)$$

The outer-sum operation ($\oplus_{\text{outer}}$) on two vectors can be seen as the outer-product except we perform an element-wise addition instead of multiplication. $\oslash$ is the Hadamard division operator and $\cdot_{min}$

Table 2: The similarity metrics used

Similarity metric over two vectors:	Similarity matrix form:
$S_{\text{Kumar}}(\vec{q},\vec{p}) = \frac{\sum_{i=1}^{m} p_i q_i}{\sum_{i=1}^{m} p_i^2 + \sum_{i=1}^{m} q_i^2 - \sum_{i=1}^{m} p_i q_i}$	$A_{\text{Kumar}} = (TT^{\mathsf{T}}) \oslash (\vec{r} \oplus_{\text{outer}} \vec{r} - TT^{\mathsf{T}})$
$S_{\text{Cosine}}(\vec{q},\vec{p}) = \frac{\sum_{i=1}^{m} p_i q_i}{\sqrt{\sum_{i=1}^{m} p_i^2}\sqrt{\sum_{i=1}^{m} q_i^2}}$	$A_{\text{Cosine}} = (TT^{\mathsf{T}}) \oslash (\vec{u} \oplus_{\text{outer}} \vec{u})$
$S_{\text{Dice-Squared}}(\vec{q},\vec{p}) = \frac{2\sum_{i=1}^{m} p_i q_i}{\sum_{i=1}^{m} p_i^2 + \sum_{i=1}^{m} q_i^2}$	$A_{\text{Dice-Squared}} = (2TT^{\mathsf{T}}) \oslash (\vec{r} \oplus_{\text{outer}} \vec{r})$
$S_{\text{Dice}}(\vec{q},\vec{p}) = \frac{2\sum_{i=1}^{m} \min(p_i, q_i)}{\sum_{i=1}^{m} p_i + q_i}$	$A_{\text{Dice}} = (2T \cdot_{min} T^{\mathsf{T}}) \oslash ((T\vec{1}) \oplus_{\text{outer}} (T\vec{1}))$

is the matrix-minimum operation, which works exactly like a matrix multiplication but uses the min() operator instead of multiplication.

The quantitative Dice coefficient is sometimes wrongly defined as the formula we define as 'Dice-Squared'. This is often because $S_{\text{Dice}}(\vec{q},\vec{p}) = S_{\text{Dice-Squared}}(\vec{q},\vec{p})$ holds for binary vectors $\vec{q},\vec{p}$ but not when $p_i \in \mathbb{R}^+$, $q_i \in \mathbb{R}^+$.

It is also important to note that for $\mathbb{R}^+$ values of $\vec{q},\vec{p}$, in comparison to the Dice similarity metric, the Dice-Squared and Kumar-Hassebrook similarity exhibit different behaviour. When using those two metrics, larger values in the feature-vector have a much larger weight in the resulting similarity. To illustrate this behaviour, take a look at the examples in Table 3. As you can see in the first three rows, as the third value in the $\vec{b}$ increases, the resulting similarity for S_{Kumar} and $S_{\text{Dice-Squared}}$ becomes much smaller compared to S_{Dice}. For this reason, we calculate the similarity between transactions on a per-feature basis instead of creating a single feature-vector from a financial transaction. In addition, the Cosine similarity could produce a high similarity even if the values in both vectors differ significantly, which can be seen in the last two rows.

$\vec{a}$	$\vec{b}$	$S_{\text{Kumar}}(\vec{a},\vec{b})$	$S_{\text{Cosine}}(\vec{a},\vec{b})$	$S_{\text{Dice-Squared}}(\vec{a},\vec{b})$	$S_{\text{Dice}}(\vec{a},\vec{b})$
$(1,1,1)$	$(1,1,0)$	0.67	0.82	0.80	0.80
$(1,1,1)$	$(1,1,2)$	0.80	0.94	0.89	0.86
$(1,1,1)$	$(1,1,7)$	0.20	0.73	0.33	0.50
$(1,10,5)$	$(1,4,2)$	0.53	0.99	0.69	0.61
$(1,1000,3)$	$(1,9,3)$	0.01	0.94	0.02	0.03

Table 3: Examples of differences between similarity-metrics

3.3 Clustering

After calculating similarity matrices for different feature-extraction methods, a final, single similarity matrix is computed. This final similarity matrix is a linear combination of these similarity matrices (see Eqn 1). Several different clustering algorithms can now be applied and compared. We will compare perceptron/neural-network clustering, single-linkage hierarchical clustering, complete-linkage hierarchical clustering and HDBSCAN [CMZS15]. These algorithms do not require the knowledge of certain parameters (i.e. the number of clusters) that are hard to guess correctly in advance, which is essential for our purpose.
When visualizing the resulting similarity matrix using the Barnes-Hut [BH86] algorithm, a distinct property of the data becomes clear. The inner-cluster similarity often has a (very) small variance, indicated by the strongly rounded cluster shapes. Sometimes, two of these rounded clusters have quite some connections between them, meaning a small hierarchy exists sometimes.

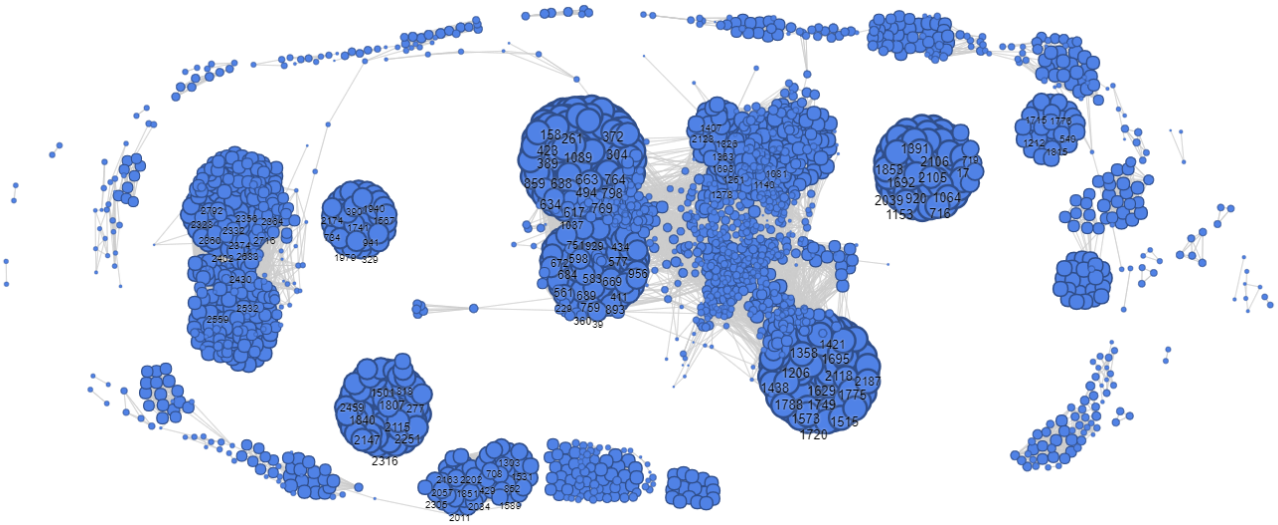


Figure 2: Final similarity-matrix of financial transactions visualized using the Barnes-Hut algorithm. DICE is used as similarity metric with a threshold of ≥ 0.7. Dataset 2 is used (personal bank account).

3.3.1 Perceptron

The resulting similarity matrix can be converted to an adjacency matrix simply by applying a threshold. In this way, the linear combination (see Eqn 1) is essentially a perceptron with a binary outcome, classifying whether or not a connection between two financial transactions exists. A larger neural network can be applied as well. Training a perceptron or neural network to classify similarity is called 'classification similarity learning' [QGCL08].
In our comparison, we experiment with a single perceptron. Since the inputs to this network are similarity matrices resulting from several different feature-extraction methods instead of the extracted features of a financial transaction, the perceptron is a lot less likely to overfit on the dataset used for training. In essence, this process of training a similarity classifier determines which feature-extraction methods are useful for clustering financial transactions. The optimize function used is the Adam algorithm, the loss function is the binary cross-entropy function.

3.3.2 Hierarchical agglomerative clustering

In hierarchical agglomerative clustering (HAC), the two closest datapoints (which is either a transaction or an existing cluster of transactions) are combined in a single cluster repeatedly, building a hierarchical tree. Different variants of hierarchical agglomerative clustering exist, each with their own definition of 'closest'. In single linkage HAC, the distance between two datapoints is the shortest distance between transactions of both datapoints. In complete linkage HAC, the distance between two datapoints is the largest distance between transactions of both datapoints. These methods have a 'distance threshold' parameter to be set in order to create flat clusters from the hierarchical approach. The distance threshold specifies when to stop merging datapoints. If the distance between two datapoints is above the distance threshold, they will not be merged.

3.3.3 HDBSCAN

HDBSCAN [CMZS15] builds upon the well-known DBSCAN [EKS+96] algorithm, but allows for varying cluster densities. It does so by first building a hierarchical tree -like hierarchical agglomerative clustering- and then cutting this tree in a 'smart' manner. When building the hierarchical tree, HDBSCAN does not use the 'plain' distance between two datapoints, but uses a mutual-reachability distance which takes density into account. This way, two pairs of datapoints which are equally far away may have a different mutual-reachability-distance, with the more densely surrounded datapoints having a lower mutual-reachability distance. Cutting this tree into flat clusters consists of two steps. First, tree branches that create a cluster with less transactions than a certain minimum are not considered a cluster split, but merely the slimming down of an larger cluster. This way, the hierarchical tree becomes much smaller. At last, the most 'stable' clusters in this small tree are cut, creating the final flat clusters. The stability of a cluster is defined as: $\sum_{p \in C_i} \lambda_p - \lambda_{\text{birth}}$, where $\lambda = \frac{1}{\text{distance}}$, λ_{birth} is the value when a cluster splits into 2 new clusters.

4 Datasets

For our analysis, we use 3 datasets. Dataset 1 contains artificially generated financial transactions with predetermined clusters. Dataset 2 contains all financial transactions from my personal bank account, 2880 in total, which have been clustered manually in order to achieve a 'ground truth' of correct clusters. Dataset 1 and 2 cover multiple years of transactions. Dataset 3 contains around 8000 transactions from Leiden University and covers a single month.

4.1 Dataset 1

For the creation of dataset 1, a generative spreadsheet file has been used (see [vHc]). Clusters can be specified, which generate transactions at a configurable interval, with a variable amount of randomness optionally added to this interval. The 'amount' of a transaction can also be configured, including a variable amount of randomness. This dataset has been used mainly for testing during development.

4.2 Dataset 2

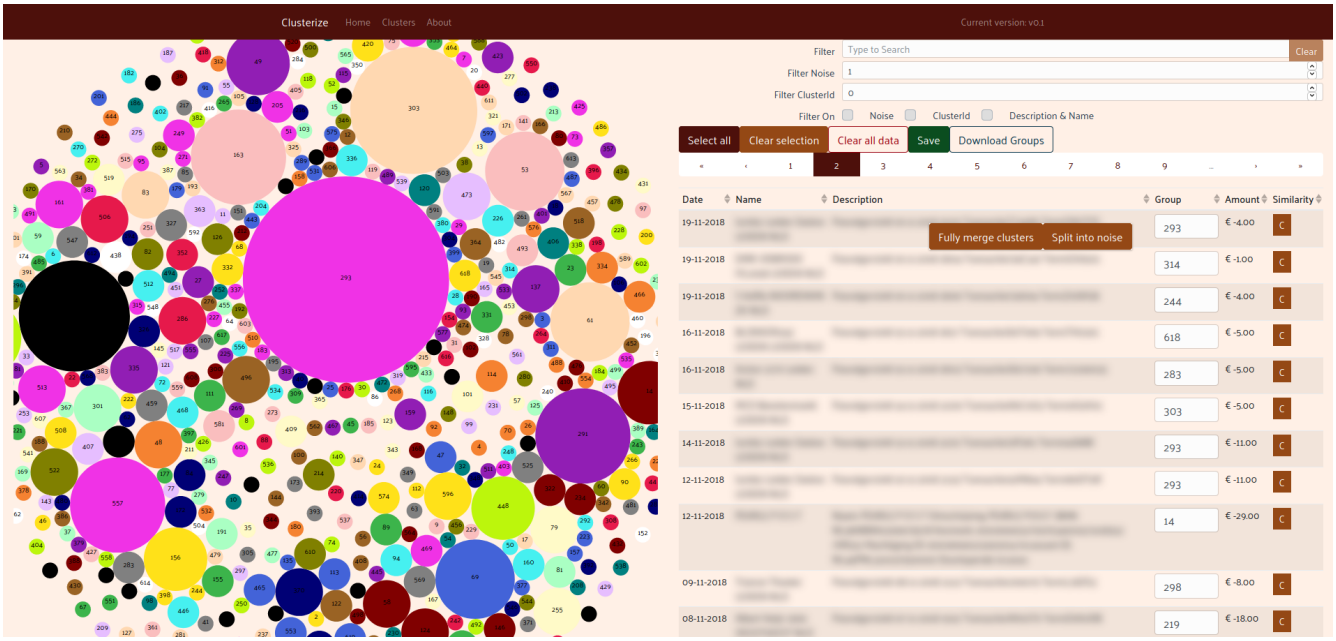


Figure 3: Clustering tool

Dataset 2 contains all 2880 financial transactions of my personal bank account, which spans about 9 years. In order to create a 'ground truth' clustering for these transactions, I have developed a tool called 'clusterize' [vHa] (see Figure 3). This tool enables easy manual clustering of data, using an intuitive user interface. The dataset was first clustered using single-linkage clustering, using the DICE similarity metric and the bigrams feature extraction method on the description of a transaction. After this, the clustering result was imported in the clustering tool and manually

checked & improved. Similar and recurring transactions where put together into clusters, for example: payments at the same supermarket chain / pub / restaurant / gym, salary payments from the same employer, and payments that belong to a unique event.

Feature extraction method	Shape	Sparsity %	Mean	Max	Min
Date - Year	[2880]	0	2015.87	2018.00	2009.00
Date - Month	[2880]	0	5.71	11.00	0.00
Date - Day of month	[2880]	0	15.80	31.00	1.00
Date - Week of year	[2880]	0	27.61	53.00	1.00
Date - Day of week	[2880]	0	2.53	6.00	1.00
Date - Quarter of year	[2880]	0	2.58	4.00	1.00
Counterparty_name - Bigrams	[2880,1698]	99	21.42	47.00	2.00
Description - Bigrams	[2880,2649]	97	75.85	245.00	0.00
Amount	[2880]	0	70.93	3000.00	0.01

Table 4: Descriptive statistics of each feature extraction method for dataset 2. Note that mean, max and min for the Bigrams method describe the string length of the corresponding attribute.

4.3 Dataset 3

Dataset 3 contains around 8000 transactions from Leiden University and covers a single month. The dataset contains small bills, reimbursements, student fees and large invoices. For this dataset, no 'ground-truth' clustered result exist. A vast amount of transactions have a large part of their (generic) description in common, with small deviations. We use this dataset to evaluate our methods applied to dataset 1 & 2.

Feature extraction method	Shape	Sparsity %	Mean	Max	Min
Date - Year	[7905]	0	2017.99	2018.00	2018.00
Date - Month	[7905]	0	0.00	0.00	0.00
Date - Day of month	[7905]	0	14.13	31.00	2.00
Date - Week of year	[7905]	0	2.57	5.00	1.00
Date - Day of week	[7905]	0	3.12	5.00	1.00
Date - Quarter of year	[7905]	0	1.00	1.00	1.00
Counterparty_name - Bigrams	[7905,896]	99	12.99	34.00	0.00
Description - Bigrams	[7905,1195]	97	40.70	114.00	4.00
Amount	[7905]	0	32997.94	61722348.00	0.29

Table 5: Descriptive statistics of each feature extraction method for dataset 3. Note that mean, max and min for the Bigrams method describe the string length of the corresponding attribute.

5 Results

The code for our analysis can be found on github [vHc]. The code to create the similarity-matrix and the perceptron is written in javascript, using the TensorFlow.js library. Python has been used to apply the other clustering methods and for the evaluation of all the clustering methods, using the numpy and scikit-learn libraries.

5.1 Feature-extraction methods

In order to determine which feature extraction methods are most relevant for clustering similar & recurring financial transactions, we use dataset 2. This dataset is most suitable because it contains real data, is sufficiently large and has an accurate 'ground truth'. As described in 3.3.1, we train a perceptron to determine which feature extraction methods are most important for clustering. The resulting input weights of a trained perceptron indicate the importance of the particular feature extraction method (see Formula 9). In essence, this is an optimization problem. In batches of size 32, the perceptron is supplied with the pairwise similarities of the financial transactions in the dataset, and the resulting binary outcome from the 'ground truth' (this indicates whether or not the two financial transactions belong in the same cluster). The similarity-metric we use for this analysis is the DICE-similarity metric. The results for other similarity metrics are roughly the same, with the importance of the `description` attribute being notably higher for the Kumar-Hassebrook similarity. These results can be seen in Table 6.

Feature extraction method	Kumar-Hassebrook	Cosine	DICE	DICE-Squared
Date - Year	6	12	15	6
Date - Month	1	8	0	1
Date - Day of month	1	2	0	0
Date - Week of year	2	8	1	2
Date - Day of week	1	1	0	0
Date - Quarter of year	1	6	1	1
Counterparty_name - Bigrams	45	44	43	51
Description - Bigrams	28	17	20	17
Amount	16	2	19	21

Table 6: Resulting importance (%) using different similarity metrics

$$\text{importance}_j = \frac{|w_j|}{\|\vec{w}\|_1} \tag{9}$$

Feature extraction method	Weight	Importance (%)
Date - Year	-1.71	15
Date - Month	0.03	0
Date - Day of month	-0.02	0
Date - Week of year	-0.11	1
Date - Day of week	-0.04	0
Date - Quarter of year	-0.10	1
Counterparty_name - Bigrams	4.99	43
Description - Bigrams	2.34	20
Amount	2.14	19

Table 7: Importance of each feature extraction method for clustering, using the DICE similarity metric.

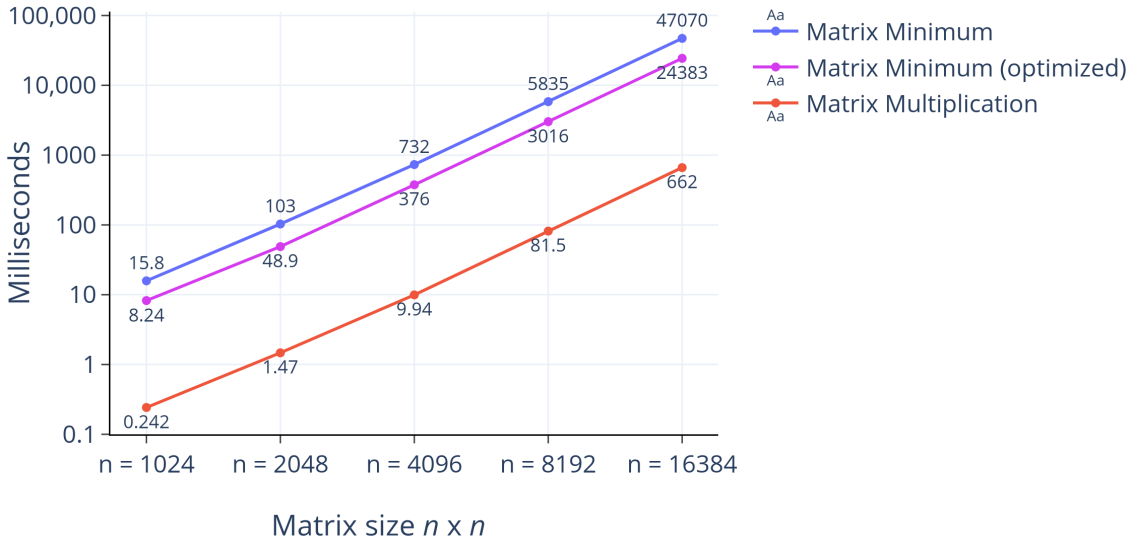
As can be seen in Table 7, the month, day of the month, week of the year, day of the week and quarter of the year attributes of a financial transaction are irrelevant for determining similar transactions. The similarity between counterparty name, description, year and amount are most important. The perceptron results for other similarity metrics are listed in Appendix B.

5.2 Similarity metrics

5.2.1 Performance

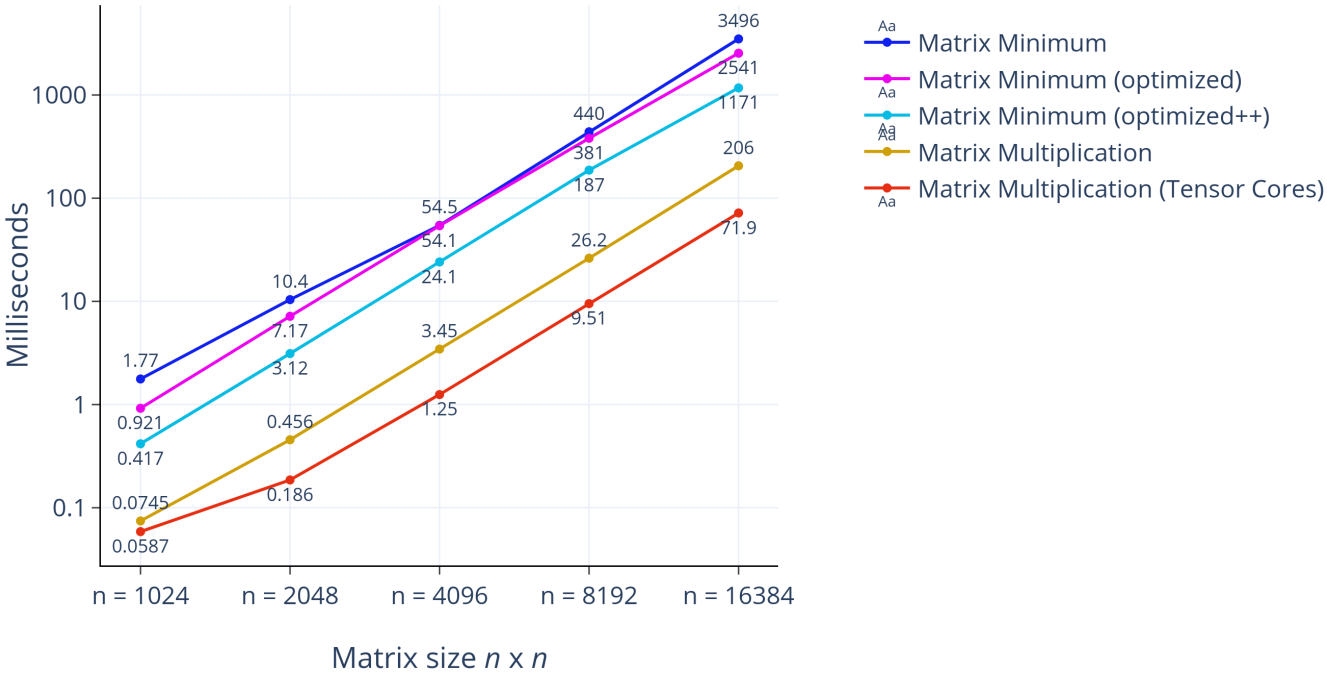
Recall that the time complexity of computing a $n \times n$ similarity or distance matrix is $O(\frac{1}{2}n^2m)$. To simplify the performance benchmark, we isolate the part of the algorithm that causes this large time complexity, the slowest part. Steps that have a time complexity of $O(\frac{1}{2}n^2)$ or less are ignored. Essentially, this is a comparison between the matrix-multiplication operation used in Cosine-similarity, Kumar-Hassebrook's PCE and Dice-Squared similarity versus the matrix-minimum operation used in the quantitative Dice coefficient. The benchmarks are ran on both a Nvidia GTX1060 and a RTX2080Ti graphics card to verify consistency, with the RTX2080Ti being equipped with Tensor Cores. The datatype for the input matrices is `int8`, and `int32` is used as accumulate/output matrix datatype as this closely resembles our use-case. We used NVIDIA's CUDA Templates for Linear Algebra Subroutines (or 'CUTLASS') library for our matrix-multiply implementation. CUTLASS decomposes the computation into an efficient hierarchy of threadblock tiles and thread tiles in order to reduce memory latency bottlenecks.

Figure 4: Performance Nvidia GTX 1060



For the implementation of the matrix-minimum operation, we modified the CUTLASS library to use the PTX ISA `vmin4` SIMD instruction [vHb], operating on two vectors of 4 8-bit integers at a time. In GPU's with compute capability ≥ 3.0 but below 5.0, the `vmin4` PTX instruction compiles to a single `VMNMX` SASS (device) instruction. For GPU's with compute capability ≥ 5.0, the `VMNMX` SASS instruction has been removed, and `vmin4` will compile to a long sequence of SASS instructions instead (see Appendix A). This long sequence of SASS instructions was not as efficient as it could be. By using the `__vmins4()` SIMD intrinsic instead, we achieved a factor 2 performance improvement on the GTX1060 and a slight performance improvement on the RTX2080Ti as well, depending on the matrix size. This is the '—Matrix Minimum (optimized)' line in Figure 4 and 5. However, the operation can be optimized even further on GPU's with compute capability ≥ 6.1 by accumulating the 4 `8-bit` integers using the `__dp4a()` intrinsic with a hardcoded `0x01010101`

Figure 5: Performance Nvidia RTX 2080Ti



value as multiplier. This makes the `__dp4a()` instruction -meant to calculate a dot product- behave as an adder. This improves performance by an additional factor 2 and is the '−Matrix Minimum (optimized++)' line in Figure 5. Taking all optimizations into account, the matrix-minimum operation is consistently 7 times slower than the matrix multiplication operation without the use of Tensor Cores. When using Tensor Cores for the matrix multiplication operation, the matrix-minimum operation is 19 times slower for sufficiently large matrices (≥ 2048).

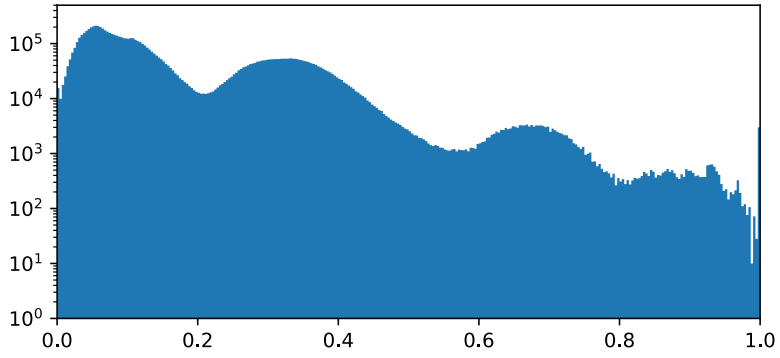
5.2.2 Accuracy

In order to determine whether the use of different similarity metrics has any effect on the accuracy of the clustering, we apply every clustering method to each resulting similarity matrix. The final results can be found in Table 9, 10, 11 and 12. The weights for each feature extraction method of a financial transaction are kept consistent for all metrics (see Table 8). These weights are different from the resulting weights in Table 7, because those weights were optimal for perceptron-based clustering. The weights in a perceptron do not allow for much variance; the slighest adjustments can result in significantly worse results. The other clustering methods are less restricted by the exact importance of each feature-extraction method. The histogram of each resulting similarity matrix is shown in Figure 6. Note that the values for Cosine similarity start with 0.1, this is because the 'amount' feature –which accounts for 10%– always results in a Cosine similarity of 1, because it is a single value, not a vector of values. We use Dataset 2 because we have a 'ground truth' available for this dataset.

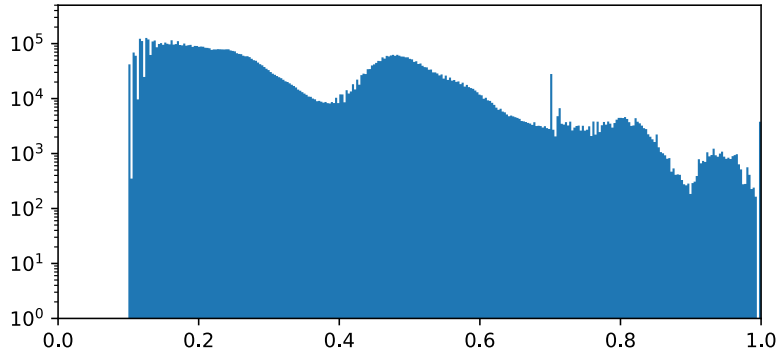
Feature extraction method	Importance (%)
Counterparty_name - Bigrams	30
Description - Bigrams	60
Amount	10

Table 8: Importance of each feature extraction method for clustering.

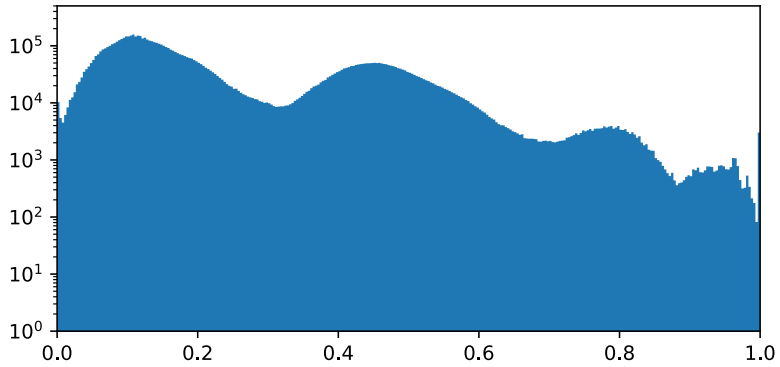
Figure 6: Histogram of values in the similarity matrices, using different similarity metrics



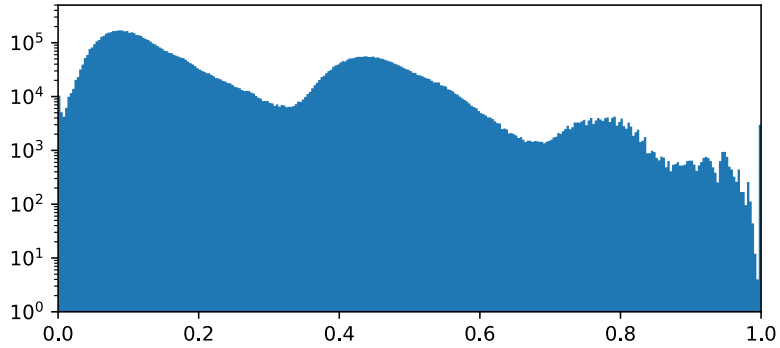
Kumar-Hassebrook's PCE



Cosine similarity



DICE-squared similarity



DICE similarity

Although the results below show no big differences in accuracy between the similarity metrics used, the accuracy of the Cosine-similarity metric is slightly worse than the other similarity metrics.

	Accuracy	Adjusted Rand	Calinski	Silhouette	Distance Threshold
Perceptron	0.9729	0.548	89.9	0.1613	
HAC-single	**0.9768**	**0.585**	95.1	0.2039	0.3
HAC-complete	0.9766	0.581	**133.6**	**0.2220**	0.5
HDBscan	0.9761	0.570	107.7	0.1723	

Table 9: Resulting clustering accuracy using the Kumar-Hassebrook similarity metric

	Accuracy	Adjusted Rand	Calinski	Silhouette	Distance Threshold
Perceptron	0.9721	0.542	114.6	0.2083	
HAC-single	0.9701	0.528	38.3	0.1914	0.2
HAC-complete	0.9742	**0.567**	**168.8**	**0.2858**	0.4
HDBscan	**0.9744**	0.547	54.5	0.1772	

Table 10: Resulting clustering accuracy using the Cosine similarity metric

	Accuracy	Adjusted Rand	Calinski	Silhouette	Distance Threshold
Perceptron	0.9708	0.529	128.1	0.1470	
HAC-single	**0.9772**	**0.594**	160.3	0.1979	0.2
HAC-complete	0.9763	0.580	**246.6**	**0.2051**	0.4
HDBscan	0.9760	0.569	172.3	0.1577	

Table 11: Resulting clustering accuracy using the Dice-Squared similarity metric

	Accuracy	Adjusted Rand	Calinski	Silhouette	Distance Threshold
Perceptron	0.9726	0.545	155.9	0.1805	
HAC-single	0.9765	0.578	191.9	0.2257	0.2
HAC-complete	**0.9768**	**0.587**	**283.4**	**0.2414**	0.4
HDBscan	0.9759	0.567	218.0	0.1907	

Table 12: Resulting clustering accuracy using the Dice similarity metric

5.3 Clustering methods

When looking at the adjusted Rand-index, the results show that in every case the perceptron is the worst clustering method, but the results between the other clustering methods do not differ a lot. Because of this, it may be more useful to determine the best clustering method on the Calinski-index and Silhouette-score. HAC complete-linkage clustering turns out to be the overall best, but it requires a 'distance-threshold' parameter to be set. When taking a look at the metrics that do not take into account a 'ground truth', we see that a significantly higher Silhouette-score

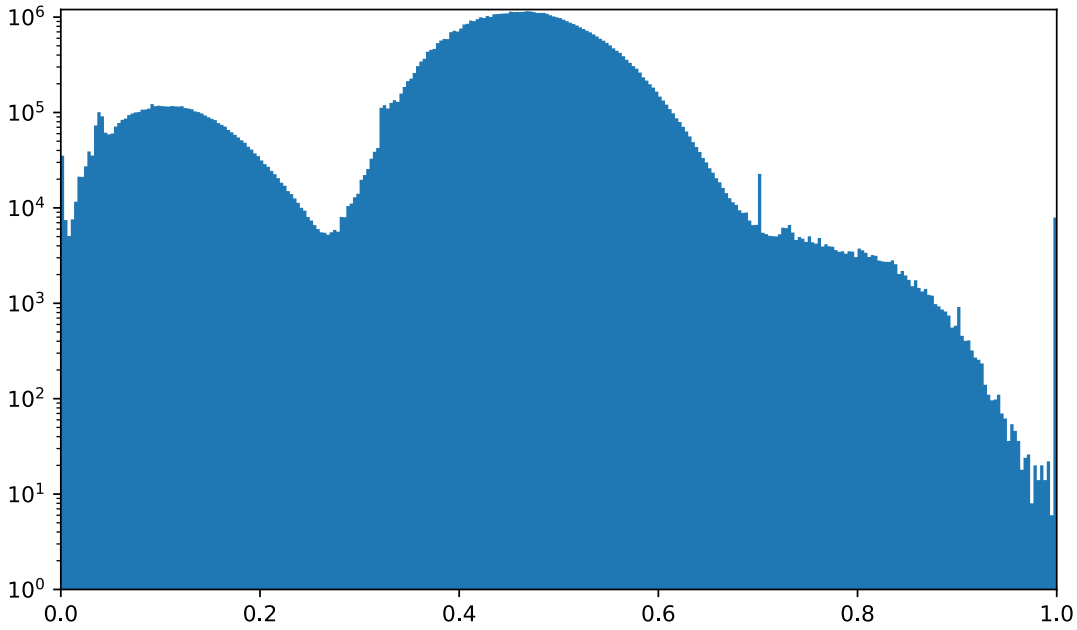


Figure 7: Histogram of values in similarity matrix of dataset 3 (FSSC data), using the Dice similarity metric

and Calinski-index corresponds with a higher accuracy. This helps judging the correctness when clustering a dataset without 'ground truth' available.
Now in order to apply our clustering methods on dataset 3, our large dataset containing the FSSC data from Leiden University, we use the Dice similarity metric. Since clustering results do not vary greatly between different similarity metrics, and the Dice similarity performs well across every clustering method used, we only use this single metric. The results of the different clustering methods can be seen in Table 13. While the Calinski-Harabasz index looks okay, we can see a Silhouette-score close to 0, quite a difference from the results of our dataset 2 (personal bank account). This indicates that the resulting clusters in dataset 3 are less defined than in dataset 2. This is likely not due to the similarity metric, given the relatively small differences in Silhouette score between a given clustering method and different similarity metrics in dataset 2. The main cause would likely be the large amount of common words in the 'description' attribute of financial transactions of dataset 3.

	Calinski	Silhouette	Distance Threshold
HAC-single	13.7	-0.3034	0.2
HAC-complete	**69.8**	**-0.0162**	0.4
HDBscan	15.2	-0.0300	

Table 13: Resulting clustering accuracy of dataset 3 (FSSC data), using the Dice similarity metric

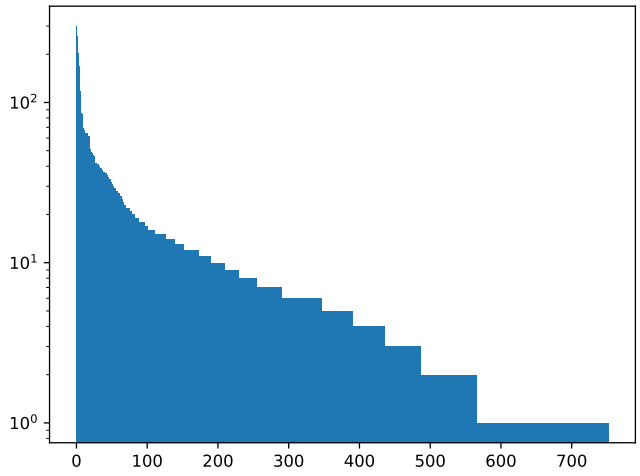


Figure 8: Cluster result dataset 3 (FSSC) using complete linkage HAC, where the cluster-id is on the x-axis and the number of transactions in a cluster on the y-axis.

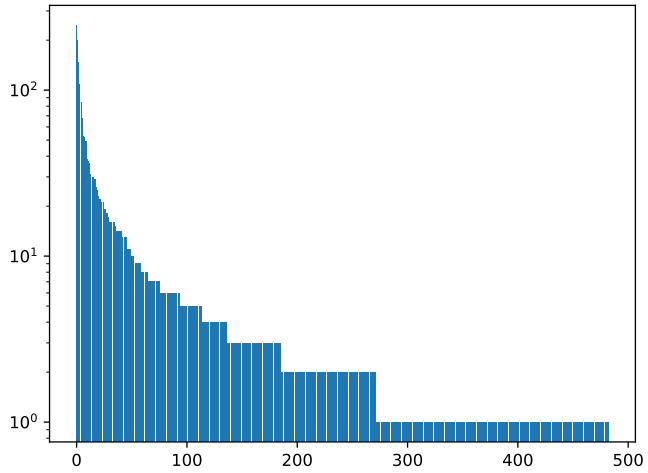


Figure 9: Cluster result dataset 2 (personal bank account) using complete linkage HAC, where the cluster-id is on the x-axis and the number of transactions in a cluster on the y-axis.

6 Discussion

When calculating a similarity matrix, the most performant similarity metric should use matrix multiplication instead of matrix-minimum and thus may benefit from NVIDIA's Tensor Cores. Since performance improvement of algorithms is an ongoing task and the architecture of GPU's and inference accelerators may change in the future, this conclusion could be affected. However, this is unlikely because a dedicated ALU-operation for minimum+accumulate has existed in the past, but has been removed from GPU's.
Hierarchical agglomerative clustering with complete-linkage turns out to be the best clustering method, but it requires a parameter to be set. The optimal value of this parameter is likely not difficult to find when no 'ground truth' clusters are available for a particular dataset, because the Calinski-Harabasz index and Silhouette score provide a good measurement of clustering quality.
The methods used in this thesis can be used in practice, but not yet without human intervention: the resulting clusters need some corrections. To easily manage and correct the resulting clustered data, a web-based prototype has been built, where the clustering algorithm can be used has a helpful time-saving technology. However, the methods could be used to fully automatically detect anomalies (outliers) in a dataset, since 100% correct clusters are not required for this: only outliers need to be correctly identified.
The results show that a significantly higher Calinski-Harabasz index and Silhouette score also relates to a higher adjusted Rand index, yet slightly. This means that significantly stronger clusters do not result in a large accuracy improvement and indicates that improvements can be made in the conversion of a financial transaction to a feature-vector. Commonly found words which are not unique to individual clusters could be ignored, which may be determined by its semantic class.

7 Conclusions and Further Research

Similar and recurring financial transactions can be clustered using fixed parameters on multiple datasets, however, some manual corrections are still required afterwards, which does not need to be difficult given a friendly user interface. We achieved an accuracy of 0.976 and an adjusted rand index of 0.580 using complete-linkage clustering. The chosen similarity metric does not significantly affect the clustering accuracy when applied on a per-feature basis to financial transactions. However, using the Cosine similarity is discouraged due to it always returning 1 when applied on single value-features. The chosen clustering algorithm affects the resulting accuracy, with complete-linkage hierarchical agglomerative clustering being the most accurate clustering method. Calculating the distance matrix is one of the slowest parts of clustering and can be sped up. Since the calculation of a distance matrix is a significant part of a clustering process, choosing a similarity metric which consists only of a dot-product (multiply-accumulate) between its two vectors will easily perform 7 times faster than a minimum-accumulate operation on NVIDIA GPU's with compute capability ≥ 6.1 using the `dp4a` instruction, and can be sped up an additional factor 2.9 using NVIDIA's Tensor Cores. Further research can be conducted on the development of better feature-extraction methods for financial transactions.

8 Acknowledgements

This work was performed using resources provided by the Academic Leiden Interdisciplinary Cluster Environment (ALICE).
I want to thank the Financial Shared Service Center of Leiden University and in particular Annette Emerenciana for providing the large dataset and Erik Weenk for being my supervisor.
I want to thank Robert_Crovella and njuff at the Nvidia developer forums for their helpful answers to my questions, the CUTLASS library developers & others on github for their help with issues I opened. Special thanks to my thesis supervisor, Dr. Suzan Verberne for providing all the assistance, tips and support needed during this project.

A Similarity metrics performance improvement

File `vmin4_dp4a_kernels.cu`, where `vmin4_intrinsic()` is more optimized than `vmin4()` on GPU's with compute capability ≥ 5.0 and `vmin4_intrinsic_dp4a()` is most optimized on GPU's with compute capability ≥ 6.1.

```
__global__ void vmin4(
        int32_t &d,
        unsigned const &A,
        unsigned const &B,
        int32_t const &c){
        #if (defined(__CUDA_ARCH__) && (__CUDA_ARCH__ >= 300))
                asm volatile("vmin4.s32.s32.s32.add %0, %1, %2, %3;"
                 : "=r"(d)
                 : "r"(A), "r"(B), "r"(c));
        #endif
}

__global__ void vmin4_intrinsic(
        int32_t &d,
        unsigned const &A,
        unsigned const &B,
        int32_t const &c){
        #if (defined(__CUDA_ARCH__) && (__CUDA_ARCH__ >= 500))
                int8_t t[4];
                unsigned &T = reinterpret_cast<unsigned &>(t);
                T = __vmins4(A, B);
                d += t[0] + t[1] + t[2] + t[3];
        #endif
}

__global__ void vmin4_intrinsic_dp4a(
        int32_t &d,
        unsigned const &A,
        unsigned const &B,
        int32_t const &c){
        #if (defined(__CUDA_ARCH__) && (__CUDA_ARCH__ >= 610))
                int32_t T = __vmins4(A, B);
                d = __dp4a(T, 0x01010101, c);
        #endif
}
```

Compiled SASS instructions:

```
   vmin4()
1    MOV R1, c[0x0][0x20] ;
2    MOV R2, c[0x0][0x148] ;
3    MOV R3, c[0x0][0x14c] ;
4  { MOV R4, c[0x0][0x150] ;
5    LDG.E R2, [R2] }
6    MOV R5, c[0x0][0x154] ;
7    LDG.E R4, [R4] ;
8    MOV R6, c[0x0][0x158] ;
9    MOV R7, c[0x0][0x15c] ;
10   LDG.E R6, [R6] ;
11   DEPBAR.LE SB5, 0x1 ;
12   PRMT R0, R2.reuse, 0x3210, R4.reuse ;
13   PRMT R8, R2, 0x7654, R4 ;
14   PRMT R9, R0, 0x7650, RZ ;
15   PRMT R11, R8.reuse, 0x7650, RZ ;
16   PRMT R3, R8.reuse, 0x7651, RZ ;
17   BFE R10, R9, 0x800 ;
18   PRMT R9, R0.reuse, 0x7651, RZ ;
19   PRMT R4, R8, 0x7652, RZ ;
20   PRMT R2, R0, 0x7652, RZ ;
21   BFE R11, R11, 0x800 ;
22   PRMT R8, R8, 0x7653, RZ ;
23   BFE R3, R3, 0x800 ;
24   PRMT R0, R0, 0x7653, RZ ;
25   BFE R9, R9, 0x800 ;
26   IMNMX R11, R10, R11, PT ;
27   BFE R4, R4, 0x800 ;
28   BFE R5, R8, 0x800 ;
29   BFE R2, R2, 0x800 ;
30   BFE R0, R0, 0x800 ;
31   IMNMX R9, R9, R3, PT ;
32   IMNMX R4, R2, R4, PT ;
33   IMNMX R5, R0, R5, PT ;
34   IADD3 R6, R9, R6, R11 ;
35   MOV R2, c[0x0][0x140] ;
36   MOV R3, c[0x0][0x144] ;
37   IADD3 R4, R5, R6, R4 ;
38   STG.E [R2], R4 ;
39   EXIT ;
40   BRA 0x1a0 ;
41   NOP;
42   NOP;
```

```
   vmin4_intrinsic()
     MOV R1, c[0x0][0x20] ;
     MOV R4, c[0x0][0x148] ;
     MOV R5, c[0x0][0x14c] ;
   { MOV R6, c[0x0][0x150] ;
     LDG.E R4, [R4] }
     MOV R7, c[0x0][0x154] ;
     LDG.E R6, [R6] ;
     MOV R2, c[0x0][0x140] ;
     MOV R3, c[0x0][0x144] ;
     LDG.E R10, [R2] ;
     DEPBAR.LE SB5, 0x1 ;
     LOP32I.AND R9, R4, 0x7f7f7f7f ;
     LOP32I.OR R0, R6, 0x80808080 ;
     IADD R9, R0, -R9 ;
     LOP3.LUT R0, R9, R6, R4.reuse, 0x96 ;
     LOP3.LUT R9, R6.reuse, R4, R9, 0x18 ;
     LOP.XOR R0, R9, R0 ;
     PRMT R9, R0, 0xba98, RZ ;
     LOP3.LUT R9, R6, R9, R4, 0xb8 ;
     BFE R4, R9, 0x808 ;
     SHR R5, R9.reuse, 0x18 ;
     BFE R0, R9.reuse, 0x800 ;
     BFE R9, R9, 0x810 ;
     IADD3 R10, R4, R5, R10 ;
     IADD3 R0, R9, R10, R0 ;
     STG.E [R2], R0 ;
     EXIT ;
     BRA 0x120 ;
     NOP;
     NOP;
```

```
vmin4_intrinsic_dp4a()
  MOV R1, c[0x0][0x20] ;
  MOV R2, c[0x0][0x148] ;
  MOV R3, c[0x0][0x14c] ;
{ MOV R4, c[0x0][0x150] ;
  LDG.E R2, [R2] }
  MOV R5, c[0x0][0x154] ;
  LDG.E R4, [R4] ;
  MOV R6, c[0x0][0x158] ;
  MOV R7, c[0x0][0x15c] ;
  LDG.E R6, [R6] ;
  MOV R3, c[0x0][0x144] ;
  DEPBAR.LE SB5, 0x1 ;
  LOP32I.AND R9, R2, 0x7f7f7f7f ;
  LOP32I.OR R0, R4, 0x80808080 ;
  IADD R9, R0, -R9 ;
  LOP3.LUT R0, R9, R4, R2.reuse, 0x96 ;
  LOP3.LUT R9, R4.reuse, R2, R9, 0x18 ;
  LOP.XOR R0, R9, R0 ;
  PRMT R9, R0, 0xba98, RZ ;
  LOP3.LUT R9, R4, R9, R2, 0xb8 ;
  MOV R2, c[0x0][0x140] ;
  IDP.4A.S8.S8 R9, R9, c[0x2][0x0], R6 ;
  STG.E [R2], R9 ;
  EXIT ;
  BRA 0x100 ;
  NOP;
  NOP;
  NOP;
  NOP;
  NOP;
```

B Perceptron weights feature-extraction methods

Feature extraction method	Kumar-Hassebrook	Cosine	DICE	DICE-Squared
Date - Year	-0.69	-1.42	-1.71	-0.60
Date - Month	0.09	-0.93	0.03	0.11
Date - Day of month	-0.09	-0.18	-0.02	-0.01
Date - Week of year	-0.19	-0.92	-0.11	-0.22
Date - Day of week	-0.06	-0.14	-0.04	-0.04
Date - Quarter of year	-0.06	-0.65	-0.10	-0.08
Counterparty_name - Bigrams	4.97	5.07	4.99	4.91
Description - Bigrams	3.09	1.92	2.34	1.66
Amount	1.77	-0.23	2.14	2.07
Bias	-5.18	-0.72	-4.94	-5.67

Table 14: Resulting perceptron weights using different similarity metrics

References

[AMD] AMD. Amd cdna architecture. https://www.amd.com/system/files/documents/amd-cdna-whitepaper.pdf.

[AMI16] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md. Rafiqul Islam. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288, 2016.

[BH86] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.

[Cha07] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *Int. J. Math. Model. Meth. Appl. Sci.*, 1, 01 2007.

[CMZS15] Ricardo J. G. B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Trans. Knowl. Discov. Data*, 10(1), July 2015.

[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[HA85] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[Int] Intel. Enhanced visual intelligence at the network edge. https://newsroom.intel.com/wp-content/uploads/sites/11/2017/08/movidius-myriad-xvpu-product-brief.pdf.

[JYP+17] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, June 2017.

[LH11] Asma S Larik and Sajjad Haider. Clustering based anomalous transaction reporting. *Procedia Computer Science*, 3:606–610, 2011.

[LLX+10] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *2010 IEEE International Conference on Data Mining*, pages 911–916, Dec 2010.

[MKM05] Yingbo Miao, Vlado Kešelj, and Evangelos Milios. Document clustering using character n-grams: a comparative evaluation with term-based and word-based clustering. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 357–358, 2005.

[NVI] NVIDIA. Nvidia tesla v100 gpu architecture. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[QGCL08] Ali Mustafa Qamar, Eric Gaussier, Jean-Pierre Chevallet, and Joo Hwee Lim. Similarity learning for nearest neighbor classification. In *2008 Eighth IEEE International Conference on Data Mining*, pages 983–988, Dec 2008.

[Sab12] Andrei Sabau. Survey of clustering based financial fraud detection research. *Informatica Economica Journal*, 16, 01 2012.

[SBHS+21] Branka Stojanović, Josip Božić, Katharina Hofer-Schmitz, Kai Nahrgang, Andreas Weber, Atta Badii, Maheshkumar Sundaram, Elliot Jordan, and Joel Runevic. Follow the trail: Machine learning for fraud detection in fintech applications. *Sensors (Basel, Switzerland)*, 21(5):1594, 2021.

[SDT10] S. A. Arul Shalom, Manoranjan Dash, and Minh Tue. An approach for fast hierarchical agglomerative clustering using graphics processors with CUDA. In Mohammed Javeed Zaki, Jeffrey Xu Yu, Balaraman Ravindran, and Vikram Pudi, editors, *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II*, volume 6119 of *Lecture Notes in Computer Science*, pages 35–42. Springer, 2010.

[SJD12] Prajol Shrestha, Christine Jacquin, and Béatrice Daille. Clustering short text and its evaluation. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, pages 169–180, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[sl] scikit learn. Calinski harabasz index sklearn source. https://github.com/scikit-learn/scikit-learn/blob/2beed5584/sklearn/metrics/cluster/_unsupervised.py#L251-L300.

[TK06] Hiroyuki Takizawa and Hiroaki Kobayashi. Hierarchical parallel processing of large scale data clustering on a PC cluster with GPU co-processing. *The Journal of Supercomputing*, 36(3):219–234, 2006.

[vHa] Vincent van Heertum. Clusterize. https://clusterize.io.

[vHb] Vincent van Heertum. Github code for the performance benchmarking. https://github.com/TheNewSound/cutlass.

[vHc] Vincent van Heertum. Github code for this thesis. https://github.com/TheNewSound/Thesis.

[YLL+14] Yan Yang, Bin Lian, Lian Li, Chen Chen, and Pu Li. Dbscan clustering algorithm applied to identify suspicious financial transactions. In *2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 60–65. IEEE, 2014.