Machine Learning For Automated Categorization of Product Articles

Filip Hedman
Abstract

Machine Learning For Automated Categorization of Product Articles

Filip Hedman

As companies gather more data every day, the possibility to extract information to make smarter business decisions increases. Machine learning and data mining has been instrumental in this trend: partly because of the increasing availability of powerful hardware, and partly because of wider availability of software tools as well as relevant data sources. This thesis uses methods of machine learning and pattern recognition to address a specific case study in the consumer business. Specifically, together with the company CASPECO we aim to find unique items in unprecise textual representations of those. A solution based on clustering is designed and evaluated. The core is to work with an appropriate distance metric between different representations. For this, two metrics are assessed: the Levenshtein and NYSIIS distance metric. The classical K-means clustering algorithm is contrasted to an adaptive clustering method, with the latter presenting a clear advantage in the given setting.

Handledare: Jonas Mattsson
Ämnesgranskare: Kristiaan Pelckmans
Examiner: Mats Daniels
IT 17 086
Tryckt av: Reprocentralen ITC
# Contents

1 Introduction .................................................. 8
   1.1 Background .............................................. 9
   1.2 Problem Definition .................................... 9
      1.2.1 Delimitations .................................... 10

2 Machine Learning ............................................ 11
   2.1 Machine Learning ....................................... 11
   2.2 Unsupervised Learning .................................. 11
      2.2.1 Cluster Analysis .................................. 12
      2.2.2 K-means ........................................... 12
      2.2.3 Adaptive Clustering ............................... 13
   2.3 Distance Metrics ....................................... 14
      2.3.1 Levenshtein String Distance ...................... 14
      2.3.2 NYSIIS Phonetic Code ............................. 15
   2.4 Evaluation Metrics ..................................... 16
      2.4.1 Verifying the Performance ......................... 16
      2.4.2 Meaning of ROC Curve ............................ 17
   2.5 Software Tools ......................................... 17
   2.6 Related Work ........................................... 18
      2.6.1 Silhouette Analysis ............................... 18
      2.6.2 Hierarchical Clustering ........................... 18
      2.6.3 Duplicate Record Detection ...................... 18

3 Results ....................................................... 20
   3.1 Preprocessing and Visualizing Data .................... 20
   3.2 Static Clustering ....................................... 22
   3.3 Adaptive Method ....................................... 23
      3.3.1 ROC Curve - Levenshtein ......................... 23
      3.3.2 ROC Curve - Levenshtein and NYSIIS .......... 24
      3.3.3 Confusion Matrix - Levenshtein .................. 26
      3.3.4 Confusion Matrix - Levenshtein with NYSIIS .... 29

4 Summary ....................................................... 32
   4.1 Discussion ............................................. 32
   4.2 Conclusion ............................................. 33
   4.3 Future Work ............................................ 34
      4.3.1 Clustering using more data points .............. 34
      4.3.2 Mapping calculated clusters to the general articles ... 34
      4.3.3 Further testing of similarity measures .......... 35

References ................................................... 36
Acknowledgements

I would like to acknowledge my reviewer Kristiaan Pelckmans for his dedication and hard work during the writing of this thesis. I would also like to acknowledge my supervisor Jonas Mattsson as well as Dan Torberg of Caspeco, which were both very helpful during the project.
1 Introduction

Modern companies accumulate vast amounts of data from their customers and users each day, and the flow of data increases each year as more companies rush to take advantage of it [1]. Endless potential lie in the analysis of this data to provide business insights, but such large data sets bring their own share of challenges. The sheer amount of data makes it impossible for a human to fully analyze it without the help of automated algorithms, and to run such algorithms often requires a lot of computational power.

Caspeco is a Uppsala based company that provides tools and services for salary handling, resource scheduling, financial analysis and budget. With thousands of customers, mainly in the hospitality business, Caspeco’s data warehouse is growing larger with each day [2].

Transactions from checkout systems are processed by Caspeco’s servers, and this data is of massive interest to restaurateurs that aim to improve their performance compared to their competitors. Some example metrics that could be gathered from the sales data include the most popular food and drinks in different parts of a country, or comparisons of margins between restaurants. To make such comparisons possible however, there is a need to identify all articles that exist at the restaurants. The problem is that each restaurant have their own way to represent the articles in their checkout system, so we need a way to map these to a set of common articles.

Machine learning is a subfield of computer science that has been researched since the 1950’s, and has gained a lot of traction in the latest years due to technology improvements and lower prices of hardware [3]. Closely related to the fields of Artificial Intelligence and Data Mining, Machine Learning aims to find patterns in large data sets without the direct involvement of a human expert [4].

The similarity measures evaluated in this thesis are Levenshtein distance and NYSIIS phonetic encoding, both regularly used in similar scenarios of duplicate record detection [5].

The results from this thesis show promising results for the adaptive method, which outperforms the K-Means method both in terms of speed as well as clustering performance. Levenshtein distance shows the most promise as a similarity measure in this scenario, but the conclusion is also that there is no silver bullet here. NYSIIS encoding works well for some shorter strings, but overall it loses too much information on long strings, which makes it hard to use in this cases as there are a lot of variance in the string lengths.
1.1 Background

Caspeco is a Uppsala based company that provides tools and services for salary handling, resource scheduling, financial analysis and budget mainly for the hospitality business, and especially restaurants [2]. With hundreds of customers, Caspeco is continuously accumulating vast amounts of data. Endless potential lie in analyzing this data to provide business insights, but there are many challenges that need to be addressed.

The main problem to be solved lies in all the transaction data that comes from the checkout systems of all the different restaurants that use Caspeco’s services. Each restaurant can add their own articles to their checkout system, and decides what to call it and what type of article it is. This poses a problem, since separate restaurants can represent an article with different names, even though the actual articles are identical. To be able to perform analytics on the sales and do comparisons between restaurants, we want to map all different spellings to one common article. See table 1 for some examples of ambiguous articles from the data set.

<table>
<thead>
<tr>
<th>Ambiguous article</th>
<th>Actual article</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carlsberg Hof 1L</td>
<td>Carlsberg Hof</td>
</tr>
<tr>
<td>Carlsberg Hof Fat 50cl</td>
<td>Carlsberg Hof</td>
</tr>
<tr>
<td>flaska carlsberg export</td>
<td>Carlsberg Export</td>
</tr>
<tr>
<td>10 fl. Heineken</td>
<td>Heineken</td>
</tr>
<tr>
<td>Heineken Fat Liten 33cl</td>
<td>Heineken</td>
</tr>
</tbody>
</table>

Table 1: Samples of ambiguous articles from the data set.

1.2 Problem Definition

The question we will try to answer in this thesis is: Can we create and test a system for recovering unique product identifiers from incomplete words?

This will be broken down into three different parts:

1. Find or construct a good distance metric measuring the similarity between words.

2. Clustering the instances into different groups using unsupervised learning, where a cluster corresponds to a unique product in the system.

3. Evaluating the system, measuring the percentage of correctly clustered transactions.
1.2.1 Delimitations

There are many data points available in each transaction of an article, for example article name, article type (such as beverage or food), time-of-sale, and price. We will only focus on the clustering of the article names in this thesis.
2 Machine Learning

2.1 Machine Learning

Machine learning is a subfield of computer science that has been researched since the 1950’s, and has gained a lot of traction in the latest years due to technology improvements and lower prices of hardware [3]. Closely related to the fields of artificial intelligence and data analysis, Machine Learning is about designing efficient and accurate prediction algorithms. Just as with data analysis, one fundamental requirement is that there are a lot of data available to learn from, and to make full use of the data at hand, some data manipulation is often involved in the process before applying the actual machine learning algorithms [6].

Machine learning is often split up in three different categories [7]:

- **Supervised learning**: The algorithm is presented with already known inputs and their desired outputs, and will try to create a general rule that successfully predicts values based on the training data.

- **Unsupervised learning**: No known input-output pairs are supplied to the algorithm, instead it tries to find patterns in the data by itself.


In this thesis we will focus on methods from the subfield of unsupervised learning, more specifically cluster analysis.

2.2 Unsupervised Learning

While supervised learning predicts output by learning from already known input-output pairs, the so called training set, unsupervised learning have no such known data about what is right or wrong to learn from. Instead, it uses different measures of similarity to find patterns in the data on its own. One other difference from supervised learning is that the algorithms does not give us any indication of their performance by themselves. Supervised learning uses the test set to compute the error, and since there is no test set in the same sense for unsupervised learning, we have to resort to other measures of error calculation [8].
2.2.1 Cluster Analysis

One common type of unsupervised learning is cluster analysis, also known as clustering. Clustering aims to find similarities in a data set, so that two or more items that are similar, according to some chosen similarity measure, belong to the same cluster. What similarity measures and specific clustering techniques that are relevant depends a lot on the type of data available, and what type of problem one wants to solve [9]. Figure 1 shows a visualization of how three clusters have been indicated using different colors.

![Figure 1: Example clustering with colors indicating each cluster.](image)

2.2.2 K-means

K-Means is a method used in cluster analysis to map a set of data points into \( K \) different clusters. The goal is to find cluster centers, also known as centroids, that minimize the intra-cluster variance. In other words, we want to minimize the sum of squared distances between each data point in a cluster to its centroid.

The term K-Means was coined by James MacQueen in 1967 [10], and the most common iterative version called Lloyd’s algorithm was developed by Stuart Lloyd in 1957, although several people have developed the same algorithm independently [11]. See listing 1 for a pseudo-code implementation of the K-Means algorithm as described by [12].
Step 1: Select an initial partition of the data set in K clusters \( \{C_1, \ldots, C_K\} \).

Step 2: Calculate cluster centroids by taking the mean of all instances in a cluster.

Step 3: for every instance \( w_i \) in the data set do:
  - Step 3.1: Reassign instance \( w_i \) to its closest cluster centroid.
  - Step 3.2: Recalculate each cluster centroid.

Step 4: if cluster membership is stable then:
  - Step 4.1: stop.
Step 5: else do:
  - Step 5.1: go to step 3.

Listing 1: Pseudo-code implementation of the K-Means algorithm.

K-Means gives no guarantees that it will converge to the global optimum, and the final clustering can depend a lot on the initialization of the centroids. The most straightforward method is to simply initialize K random points as the initial centroids. Since this is pretty fast, it is common to run the algorithm multiple times with different randomized initial centroids and pick the best result.

Another popular method is K-Means++, and the idea with that method is to spread out the initial cluster centroids by first choosing one of the data points at random as the centroid, and then continuing by choosing each subsequent centroid with a probability that is proportional to the squared distance to the closest existing centroid to that specific data point. K-Means++ is proven to perform better than the regular random approach [13], and is thus the preferred method of initialization in this project.

2.2.3 Adaptive Clustering

An adaptive clustering method was implemented that aims to be fast and effective even on very large data sets. Algorithms such as K-means require a lot of memory to fit the pairwise matrices used when calculating clusters. Since the focus is on similarity measures when clustering strings, the algorithm itself is trivial. The basic idea is to go through each article string, look at the currently existing centroid strings, and then comparing those strings using some similarity measure. The similarity measures compared in this thesis are Levenshtein distance and the NYSIIS phonetic algorithm. See listing 2 for an implementation of the adaptive algorithm in Python 3.
Step 1: Create an empty set of clusters.
Step 2: for every article $a_i$ in the data set do:
  Step 3: Get the cluster centroid $C_i$ that is closest to $a_i$.
  Step 4: if no clusters exist yet then:
    Step 4.1: Make $a_i$ the first cluster centroid.
    Step 4.2: Continue loop at Step 2.
Step 5: if no cluster is close enough then:
  Step 5.1: Make $a_i$ a new cluster centroid.
  Step 5.2: Continue loop at Step 2.
Step 6: else do:
  Step 6.1: Add $a_i$ to $C_i$.
  Step 6.2: Continue loop at Step 2.

Listing 2: Pseudo-code implementation of adaptive clustering algorithm.

2.3 Distance Metrics

2.3.1 Levenshtein String Distance

To be able to use clustering algorithms, we need to choose a similarity measure. Since our use case involves strings, we need a way to calculate how similar two strings are. This is not entirely trivial, and a lot of different algorithms have been developed for this cause.

Levenshtein distance is one metric for calculating the differences between a pair of strings. Informally it is defined as the amount of single-character edits needed to transform one string into another. Formally, it is defined as follows [14]:

$$lev_{x,y}(i,j) = \begin{cases} 
\max(i,j) & \text{if } \min(i,j) = 0, \\
\min \begin{cases} 
lev_{x,y}(i-1,j) + 1 \\
lev_{x,y}(i,j-1) + 1 \\
lev_{x,y}(i-1,j-1) + 1_{(x_i \neq y_j)} 
\end{cases} & \text{otherwise.}
\end{cases}$$ (1)

Where $x, y$ are two strings to be compared. $x_i$ is the character at point $i$ in the string $x$, and $y_j$ is the character at point $j$ in the string $y$, for all $i, j \in \mathbb{N}$.

$1_{(x_i \neq y_j)}$ is the indicator function defined as:

$$1_{(x_i \neq y_j)} = \begin{cases} 
0 & \text{if } x_i = y_j, \\
1 & \text{otherwise.}
\end{cases}$$ (2)
To give an example, the Levenshtein distance between the words *aim* and *chair* is 3, since we need to make the following three edits to transform the first string into the other:

1. *aim* → *air* (substitution of *m* for *r*)
2. *air* → *hair* (insertion of *h* at the beginning)
3. *hair* → *chair* (insertion of *c* at the beginning).

The Levenshtein distance measure have several advantages over other similar measures. It is clear and concise, which makes it fairly easy to implement, and it also fulfills the requirements for triangle inequality. This means that the distance between two strings is no larger than the sum of their distances to a third string [15].

### 2.3.2 NYSIIS Phonetic Code

The New York State Identification and Intelligence System, or NYSIIS for short, is an algorithm to convert a word into a phonetic code. This method was developed in 1970 by Robert L. Taft for the State of New York to help cluster together names that are pronounced the same way, to be used for example when there are spelling errors on surnames [16]. It features an accuracy increase of 2.7% over the traditional Soundex algorithm, which is another widely used phonetic algorithm developed in the early 1900’s [17].

Most phonetic encodings, including NYSIIS, work by encoding phonemes in different ways. A phoneme is simply a unit of sound that distinguish one word from another. Phonetic encodings usually have some kind of lookup table that maps phonemes to a character, and multiple other rules that define that specific encoding.

To get an idea of how NYSIIS might help us cluster together strings with similar spelling and structure, here are some examples of strings with their NYSIIS encoded counterparts:

<table>
<thead>
<tr>
<th>Original String</th>
<th>NYSIIS Encoded</th>
</tr>
</thead>
<tbody>
<tr>
<td>carlsberg</td>
<td>CARLSBARG</td>
</tr>
<tr>
<td>karlsbherg</td>
<td>CARLSBARG</td>
</tr>
<tr>
<td>heineken</td>
<td>HANACAN</td>
</tr>
<tr>
<td>heinikin</td>
<td>HANACAN</td>
</tr>
</tbody>
</table>

Table 2: Original data samples and their NYSIIS counterparts.

As can be seen in table 2, NYSIIS successfully sees through the spelling errors made in the two different cases by focusing on the pronunciation of the words. The produced
encodings of the misspelled words are equal to the encoding of the correctly spelled words, and can thus be clustered together.

Phonetic algorithms have successfully been used in other applications which suffer from similar type of misspellings, for example in search engines, where results with similar phonetic codes are suggested after a performed search [18]. Since the classic NYSIIS encoding produces a phonetic code of a maximum of six characters, it will not perform well on longer input strings. To get around this we are running a modified version that produces arbitrarily long strings, and we will also run the algorithm on each individual word of a string, if there exists more than one.

2.4 Evaluation Metrics

2.4.1 Verifying the Performance

To verify that the developed system produces viable results, we will make use of the large amount of already classified items in the Caspeco database. The company is currently mapping the different article names manually with an expert which carefully looks at the name and determines which product in the Caspeco database it should be mapped to. As of writing this thesis, the database of already mapped article exceeds 110 thousand different strings, consisting mostly of different typographic errors made from the restaurant owners when setting up their checkout systems.

We can test our automatic clustering algorithm on this database, since we know what the correct clusters should look like and therefore can calculate a percentage representing the system’s performance. We are going to focus on sensitivity and specificity as measurements of the algorithm’s performance. Sensitivity is the proportion of positive items in the test set that are correctly identified as such, while specificity is the proportion of negative items in the test set that are correctly classified. From these measures we can then create a ROC curve.

Since each unique cluster is essentially a class, we need a way to calculate the amount of correct clusterings over multiple classes. This can be done by creating unique pairs of every article string in the data. Each of these pairs are then evaluated by checking if the two articles belong to the same expert cluster, and if so, if they are also in the same cluster calculated by the algorithm.

Given the set \( A = \{a_1, a_2, \ldots, a_n\} \), for all \( n \in \mathbb{N} \), of all available article strings, let \( \{a_i, a_j\} \) be the unordered pair where \( \{a_i, a_j\} \in A, a_i \neq a_j \) and \( i, j \in \mathbb{N} \).

Let \( EXPERT = \{e_1, e_2, \ldots, e_n\} \), for all \( n \in \mathbb{N} \), be the set of expert clusters, where \( e_i, e_j \subset A \) and \( e_i \cap e_j = \emptyset \), for all \( i, j \in \mathbb{N} \).
Let \( CALC = \{c_1, c_2, \ldots, c_n\} \), for all \( n \in \mathbb{N} \), be the set of calculated clusters, where \( c_i, c_j \subset A \) and \( c_i \cap c_j = \emptyset \), for all \( i, j \in \mathbb{N} \).

Given these restraints, we have the following rules:

- **True Positive** \( \iff \{a_i, a_j\} \in e_i \cap c_i \).
- **False Positive** \( \iff \{a_i, a_j\} \notin e_i \land \{a_i, a_j\} \in c_i \).
- **True Negative** \( \iff \{a_i, a_j\} \notin e_i \cup c_i \).
- **False Negative** \( \iff \{a_i, a_j\} \in e_i \land \{a_i, a_j\} \notin c_i \).

for all \( i, j \in \mathbb{N} \).

Sensitivity and specificity are then defined as follows:

- **Sensitivity** = \( \frac{\text{True Positives amount}}{\text{True Positives amount} + \text{False Negatives amount}} \).
- **Specificity** = \( \frac{\text{True Negatives amount}}{\text{True Negatives amount} + \text{False Positive amount}} \).

The calculations were done 100 times, each time with a new random ordering of the article strings. This way the performance of the algorithm can be evaluated better, since there will be no bias from a specific sequence of articles.

### 2.4.2 Meaning of ROC Curve

The Receiver Operating Characteristic curve, or ROC curve for short, is a way to plot the relation between the sensitivity and specificity of our classification algorithm as we change our threshold value \( \epsilon \). We can use it to determine which \( \epsilon \) performs best for the current settings of the algorithm by examining the curve. Another usage is that we can calculate the area under the curve, which is equal to the probability that two articles that should be clustered together are correctly clustered together using our algorithm.

### 2.5 Software Tools

To process data and calculate the clusters in this project, several tools are going to be used. The main programming language of choice will be Python 3 together with the framework scikit-learn, since they are widely used in scientific and commercial communities working with machine learning [19, 20].

MSSQL is the database of choice to store all the raw article data, and simple MSSQL
scripts are used to extract the articles from the Microsoft Azure cloud platform [21, 22].

For data manipulation we mainly use Pandas and Numpy, which both are useful for extracting certain rows and calculating statistics for our data, and also for saving and loading calculated clusters to and from files [23, 24].

Data visualization is managed through the Python library Matplotlib [25].

2.6 Related Work

2.6.1 Silhouette Analysis

Some clustering algorithms such as K-Means requires the amount of clusters to be determined before the calculations begin. This cluster amount is not always trivial to pick out, since we might not know beforehand how many clusters a data set consists of. Rousseeuw proposes a way to overcome this problem, by plotting each cluster as a so-called silhouette on a plot.

This method is called silhouette analysis, and the silhouette is a representation of the tightness and separation of each cluster. The final plot combines all silhouettes, allowing a graphical overview of the relative quality of each cluster. Using silhouette analysis on multiple values for K is one way of choosing an appropriate amount of clusters for the algorithm at hand [26].

2.6.2 Hierarchical Clustering

Hierarchical clustering, as the name suggests, aims to build a hierarchy of clusters. There are two main strategies when building the clusters, agglomerative and divisive. Agglomerative is a bottom-up approach, where each data point starts with its own cluster, which is later merged together with other clusters as the algorithm moves up in the hierarchy. Divisive on the other hand is a top-down approach that starts with only one cluster, and will split up and create more clusters as the algorithm moves on [27].

2.6.3 Duplicate Record Detection

When working with real data from a lot of different sources, one common problem is that multiple entities exists to represent the same real life object. When working with text data, it can be the result of a regular typographical error or spelling mistake, or just different standards of representing information. Finding those duplicates is not always trivial, and a lot of work has been done in the field of duplicate record detection
to eliminate these problems. Elmagarmid et al. [5] discusses several algorithms and similarity measures to eliminate the problem, and some of them are very relevant for the type of problems we face in this thesis as well.

The Levenshtein algorithm used in thesis, as well as the phonetic NYSIIS algorithm, are both included in the survey as capable ways of detecting similar records of text. They do however name multiple other algorithms that may be interesting to use, but was left out due to time constraints. Some of those include, but are not limited to; Q-grams, Soundex, and Metaphone.

While this thesis is only concerned about the textual data of an article in the database, the paper also covers duplicate record detection when there are multiple data fields available. This would certainly be interesting if one were to evolve this project further, since there are often article types, such as food or beverage, included in the database.
3 Results

3.1 Preprocessing and Visualizing Data

The raw data is stored on the Azure cloud platform, and we can get access to it through a MSSQL client such as Microsoft SQL Server Management Studio. From the client we can export the data to a CSV file (comma-separated file), which is easy for us to handle in our Python application.

Due to the large number of restaurants in the system, the article names show a great variation when it comes to spelling and general style of describing the article at hand. Some article strings include corrupted data which crashes the similarity algorithms, so the decision was made to include only ASCII characters and remove faulty lines.

When testing and developing the system, it can be useful to have smaller data sets derived from the original data to enable for faster runtimes on less capable hardware. In these cases the UNIX shuf tool was used to get a random shuffle of the original data and pick the amount of wanted rows. The shuffling is done to generate a data set which is still similar to the original data set in terms of distribution. The data set used for the calculations in this section had 10000 rows.

When using K-Means for clustering, we start off by creating a matrix of pairwise distances between each article string. This essentially gives us a higher dimensional data set, and while it works fine for the actual clustering, it is hard to visualize. Principal Component Analysis, or PCA for short, is a way to make a linear transformation of the input data so that the output dimensions maximize variance and are orthogonal to each other [28]. Since it can be very valuable to visually analyze a plot of data, PCA is a useful technique. Figure 2 and 3 show the effect of PCA on a small set of test data. There are eleven article strings in the data set, which creates a pairwise matrix of eleven dimensions. Plotting data with eleven dimensions on a two-dimensional plot is nonsensical, which can be seen in figure 2. After applying PCA to the data however, it is possible to discern the four clusters in figure 3.
Figure 2: Eleven data points clustered by K-Means using Levenshtein distance as the similarity measure, visualized without PCA.
3.2 Static Clustering

Table 3 shows the performance of the K-Means algorithm for different choices of K.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>12081</td>
<td>109399</td>
<td>49534648</td>
<td>338872</td>
</tr>
<tr>
<td>1000</td>
<td>9949</td>
<td>48741</td>
<td>49595306</td>
<td>341004</td>
</tr>
<tr>
<td>1500</td>
<td>8140</td>
<td>29666</td>
<td>49614381</td>
<td>342813</td>
</tr>
<tr>
<td>2000</td>
<td>6946</td>
<td>21134</td>
<td>49622913</td>
<td>344007</td>
</tr>
<tr>
<td>2500</td>
<td>6440</td>
<td>15357</td>
<td>49628690</td>
<td>344513</td>
</tr>
<tr>
<td>3000</td>
<td>20293</td>
<td>195444</td>
<td>49448603</td>
<td>330660</td>
</tr>
</tbody>
</table>

Table 3: Performance of K-Means algorithm for different choices of K.

The results for K-Means are not especially impressive. When running the algorithm with 2500 clusters, we get a true positive rate of 30 percent and a false positive rate of 70 percent, which is the best result from these tests. It is very hard to know the optimal number of clusters beforehand, which is another reason why K-Means is a bad choice for
this sort of problem where the cluster amount is unknown.

3.3 Adaptive Method

3.3.1 ROC Curve - Levenshtein

Figure 4: True positive rate against false positive rate of the adaptive algorithm using Levenshtein distance.

Figure 4 shows the ROC curve for the adaptive algorithm using regular Levenshtein distance. Epsilon is the threshold used when deciding if two strings should be clustered together or not, and it is tested between 1 and 10. Each step in the curve represents a choice of epsilon, and it is possible to see that an optimal choice of epsilon is in the beginning of the curve, since the sensitivity increases more rapidly than the decrease of specificity. As the threshold gets bigger, more strings will be falsely clustered together, which can be seen in the right part of the curve where the false positives increase massively, while the true positives only increase slightly. In this case we chose 3 as the epsilon
for the forthcoming tests, which has a good trade-off between true positives and false positives.

By calculating the area under the curve we get 0.75, which is equal to the probability that the algorithm will successfully cluster together two strings that are supposed to be clustered together. This is clearly better than random guessing, but to get the full picture we need to look at the confusion matrix for the algorithm.

### 3.3.2 ROC Curve - Levenshtein and NYSIIS

![ROC Curve - Epsilon range 1 to 10](image)

Figure 5: True positive rate against false positive rate for the adaptive algorithm using Levenshtein and NYSIIS together.

Figure 5 shows the ROC curve for the adaptive algorithm using regular Levenshtein distance and the NYSIIS phonetic encoding together. The curve has as similar style to that of figure 4, with a large increase in sensitivity in the beginning, and a small decrease in specificity. The chosen epsilon for the NYSIIS method is 3, which is the same as the regular Levenshtein algorithm.
In this curve we can see that the algorithm reacts more heavily when epsilon is increased, which makes sense since the idea behind NYSIIS is to cluster together strings according to their phonetic similarities. Whenever two strings share some of the phonemes, they will tend to cluster together more heavily.

By calculating the area under the curve we get 0.61, which is equal to the probability that the algorithm will successfully cluster together two strings that are supposed to be clustered together. This is a lower percentage than the method with regular Levenshtein distance, which gives us a hint that this method is not performing as well. To get more details we need to look at the confusion matrix for the algorithm.
3.3.3 Confusion Matrix - Levenshtein

Unnormalized confusion matrix

<table>
<thead>
<tr>
<th>True Label</th>
<th>Predicted Label</th>
<th>Different Cluster</th>
<th>Same Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different</td>
<td>True Negative: 49642948</td>
<td>False Positives: 1098</td>
<td>False Positive Rate: 0.00002</td>
</tr>
<tr>
<td>Same Cluster</td>
<td>False Negatives: 347891</td>
<td>True Positives: 3061</td>
<td>True Positive Rate: 0.01</td>
</tr>
<tr>
<td></td>
<td>Negative Prediction Value: 0.99</td>
<td>Positive Prediction Value: 0.74</td>
<td>Accuracy: 0.99</td>
</tr>
<tr>
<td></td>
<td>(Total Negative: 49644047)</td>
<td>(Total Positive: 350953)</td>
<td>Specificity: 1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sensitivity: 0.01</td>
</tr>
</tbody>
</table>

Figure 6: Unnormalized confusion matrix of the adaptive algorithm using Levenshtein distance with Epsilon = 3. The blue boxes highlight the correctly classified items.
Figure 7: Normalized confusion matrix of the adaptive algorithm using Levenshtein distance with Epsilon = 3. The blue boxes highlight the correctly classified items.

In figure 6 and 7 we can see the performance of the adaptive algorithm using confusion matrices. We can see that when the algorithm predicts that a pair is in the same cluster, it is correct 74% of the time. On the other hand, when it predicts that a pair does not belong in the same cluster, it is correct in 99% of the cases. Those numbers make sense,
since it is easier to split up two strings that are very different than to decide if two strings that are around the cut-off point should be clustered together or not. See table 4 for the arithmetic means and standard deviations of the 100 calculation runs.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Arithmetic Mean</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Pos.</td>
<td>3061</td>
<td>52</td>
</tr>
<tr>
<td>False Pos.</td>
<td>1098</td>
<td>76</td>
</tr>
<tr>
<td>True Neg.</td>
<td>49642948</td>
<td>76</td>
</tr>
<tr>
<td>False Neg.</td>
<td>347891</td>
<td>52</td>
</tr>
</tbody>
</table>

Table 4: Arithmetic means and standard deviations of the adaptive algorithm using Levenshtein.
3.3.4 Confusion Matrix - Levenshtein with NYSIIS

Unnormalized confusion matrix

<table>
<thead>
<tr>
<th></th>
<th>Different Cluster</th>
<th>Same Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>True Negative: 49626420</td>
<td>False Positives: 17626</td>
<td>False Positive Rate: 0.00036</td>
</tr>
<tr>
<td>False Negatives: 343965</td>
<td>True Positives: 6987</td>
<td>True Positive Rate: 0.02</td>
</tr>
</tbody>
</table>

| Predicted Label    |                   |              |
| Different Cluster  |                   |              |
| Negative Prediction Value: 0.99 | Positive Prediction Value: 0.28 | Accuracy: 0.99 |
| (Total Negative: 49644047) | (Total Positive: 350953) | Specificity: 1.00 |
|                     |                   |              |

Figure 8: Unnormalized confusion matrix of the adaptive algorithm using Levenshtein together with NYSIIS, using Epsilon = 3. The blue boxes highlight the correctly classified items.
Figure 9: Normalized confusion matrix of the adaptive algorithm using Levenshtein together with NYSIIS, using Epsilon = 3. The blue boxes highlight the correctly classified items.

In figure 8 and 9 we can see the performance of the adaptive algorithm using NYSIIS with the help of a confusion matrix. We can see that when the algorithm predicts that a pair is in the same cluster, it is correct only 28% of the time. On the other hand,
when it predicts that a pair does not belong in the same cluster, it is correct in 99% of the cases. These results are clearly worse than the regular Levenshtein method, and the largest culprit seem to be that strings are falsely clustered together when they should really be in different clusters. This might point out that we lose too much information when using NYSIIS on the long strings that occur in our article data set.

See table 5 for the arithmetic means and standard deviations of the 100 calculation runs.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Arithmetic Mean</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Pos.</td>
<td>6987</td>
<td>250</td>
</tr>
<tr>
<td>False Pos.</td>
<td>17626</td>
<td>3117</td>
</tr>
<tr>
<td>True Neg.</td>
<td>49626420</td>
<td>3117</td>
</tr>
<tr>
<td>False Neg.</td>
<td>343965</td>
<td>250</td>
</tr>
</tbody>
</table>

Table 5: Arithmetic means and standard deviations of the adaptive algorithm using Levenshtein with NYSIIS.
4 Summary

4.1 Discussion

The results show that it is indeed possible to use methods from cluster analysis to find similar strings in a data set. The similarity measure is very important, and finding a good cut-off point is crucial. It is possible to generate a ROC curve to visually analyze the impact of changing the cut-off point, also known as epsilon, and find a suitable one for the chosen algorithms.

Python and the chosen toolkits such as scikit-learn and pandas proved to be great choices for the project, with easy-to-use API’s and great documentation. Scikit-learn’s underlying usage of Numpy makes it possible to run clustering algorithms on moderately large data sets on a modern laptop with reasonable performance. A lot of the most common machine learning algorithms are already implemented in the scikit-learn library, which makes it easy to try out multiple methods on a data set without needing to implement them all from scratch.

The clustering method that performed best in terms of clustering was the adaptive method developed for this thesis, with regular Levenshtein distance as the chosen similarity measure. While the method is affected by the ordering of the items to be clustered, that is also the case with K-Means. The adaptive method is much faster than K-Means however, which makes it more feasible to run multiple times to get the best value over multiple runs. The actual clustering part is greedy and simple, which puts the similarity measure in focus. This makes it a good method to use if further comparisons of similarity measures are to be done.

Figure 4 and 5 show the ROC curves for the adaptive method when using Levenshtein distance and NYSIIS phonetic encoding as the similarity measure. What we can see in those curves are that the sensitivity is increasing a lot in the beginning relative to the specificity, and it is not until epsilon is equal to 3 that specificity is starting to take over increasingly. While we do want to cluster together as many correct strings as possible, there is nothing to gain with a too large epsilon, since we get an even larger number of incorrectly clustered strings. Epsilon was set to 3 in the later tests, since that point seemed like a good trade-off between sensitivity and specificity.

The confusion matrices in figure 6 and 8 show a disproportionate amount of true negative values. This is due to the way we do our calculations, see the section Verifying the Performance for more details. There is simply a much larger probability that two article strings should be separated and not clustered together. What it really means is that we can without a problem separate two very different strings, such as "apple" and "new york", but the problems arise when we are comparing two strings that are somewhat similar, but are caught on either side of the cut-off point.
K-Means worked well for small amounts of data, and showed nice clusterings in early testing. However, as the amount of clusters grew, it performed worse. Since we create a matrix of pairwise distances between each input string, we essentially have a data set with thousands of dimensions. K-Means is known to suffer from the curse of dimensionality, which is probably the case here. K-Means tries to minimize the inter-cluster variance, and with thousands of dimensions to look at when minimizing, the problem becomes very difficult to solve [29].

NYSIIS phonetic encoding has been used successfully in similar scenarios with duplicate record detection, but the encoding did not improve the clustering performance with this data set and chosen algorithms. One possible reason for this is that NYSIIS is originally developed for use on surnames, which most often adhere to certain characteristics. Surnames are often relatively short and contain two or three syllables. The data used in this thesis contain some very long and complicated names, and the NYSIIS encoding method probably removes too much of the relevant information.

Further data preprocessing might have been useful, such as focusing on article strings of similar lengths. Having a large variance in string length makes it hard to chose a reasonable breakpoint that decides whether or not a string should be part of a cluster or not. Having a breakpoint that is too large will put all short strings in the same cluster, which will result in a lot of false positives. This effect can be seen in figure 4 and 5 as epsilon is increased.

4.2 Conclusion

Using methods from the fields of machine learning and data mining, we have successfully measured the performance of two different clustering algorithms as well as two different distance measures for textual data.

Python and the chosen toolkits such as scikit-learn and pandas proved to be great choices for the project, with easy-to-use API’s and great documentation.

The two clustering algorithms that were compared were K-Means and a custom-written adaptive method. ROC curves were used to analyze which value that performed best as a threshold for the adaptive algorithm. This was calculated separately for both the method using regular Levenshtein distance, as well as the method that used NYSIIS phonetic encoding. In both cases we chose 3 as the optimal cut-off value, by evaluating the sensitivity against the specificity. Using data correctly mapped by an expert as a reference, we then calculated how many articles that were correctly mapped with each other, and vice versa.

We concluded from those results that the adaptive method outperformed the K-Means algorithm both in terms of speed, as well as correctness of resulting clusters. The adaptive
method with Levenshtein distance as the similarity measure had a 74 percent true positive rate, while the true positive rate for K-Means was 30 percent at best.

The flexibility of the adaptive algorithm is favorable as well, since there is no need to specify the amount of clusters beforehand. While the adaptive method is simple by design and mostly depends on its similarity measure, K-Means most likely suffered from the curse of dimensionality, since it performed well in early tests with small data sets, while performing notably worse on the larger testing data set.

Two different similarity measures for strings were also evaluated to be used with the clustering algorithms. The results showed that Levenshtein distance outperformed the combination of Levenshtein and NYSIS encoding with the adaptive algorithm. The true positive rate of pure Levenshtein was 74 percent, while the use of NYSIS phonetic encoding had 28 percent.

There is however no definite correct answer here, since it depends a lot on the structure of the strings in the data set. For small data sets K-Means worked fine, and in the same vein NYSIS phonetic encoding worked well for short strings. This underlines the need to evaluate the data set beforehand, and realize that there is no silver bullet for all types of data.

### 4.3 Future Work

**4.3.1 Clustering using more data points**

In this project we only analyzed the textual property of an article, in other words, the name or title of the article. However, many of the articles in the database include more information, such as article type, cost, and so on. It would surely be useful to know if an article is a beverage or food in cases where the name itself is too closely related to discern if there are multiple articles, or just one.

**4.3.2 Mapping calculated clusters to the general articles**

This thesis focused on the unsupervised clustering of article strings, and while clusters with similar articles are found, there is nothing that actually maps these articles to the already existing general articles, also known as *Caspeco Articles* in the text. This could be done trivially by comparing each centroid string to the database of Caspeco articles, and the closest matching string will represent the final mapping for that whole cluster. Since all strings in a cluster are interpreted as the same article, we can then assume transitivity and map the same general article to the whole cluster. There are however multiple ways of solving this, and it should be investigated further.
4.3.3 Further testing of similarity measures

Due to time constraints, only a couple of similarity measures were evaluated; Levenshtein
distance and NYSIIS phonetic encoding. There are many other algorithms and techniques
available however, such as Q-grams and Jaro-Winkler [30, 31].
References


[14] V. I. Levenstein, “Binary codes capable of correcting deletions, insertions, and


[28] K. Pearson, “On lines and planes of closest fit to systems of points in space,” *The Lon-

