Investigating Machine Learning Clustering Methods to Replicate the Human Idea of Structure to Documents

Johannes Jansson, Victor Miller

March 20, 2018

Abstract

Anyone trying to maintain a set of text documents in an information retrieval system will run into problems keeping it relevant and up to date as the amount of data increases. This thesis investigates how a collection of documents can be clustered in a way that resembles how a human would organize it. It also assesses how difficult it is to implement this into an existing information retrieval system with current programming libraries, and in what practical ways this can be useful.

The text data in this project is represented by a TF-IDF model. A K-Means clustering algorithm generates one clustering, and a Support Vector Machine is trained with minimal user data to provide another clustering. These two are then evaluated and compared using a set of metrics. This project takes a practical approach to the problem, focusing on what can be implemented using existing programming libraries and what will actually work in a production environment. Software for visualizing the corpus and calculating similar documents, are implemented as well.

The supervised method SVM greatly surpasses the unsupervised method K-Means in being able to replicate the given ground truth, but both models are in themselves useful. With a relatively simple understanding of machine learning, any company could set up a similar system. It does, however, take some deeper mathematical knowledge and fine tuning to get the most out of it and tailor it to the dataset.

Acknowledgements

We want to thank our supervisor Martin Lundin from Talkative Labs, who made this entire project possible. We also want to thank our client, who taught us that when in doubt, always do both. A special thanks to our academic supervisor Kalle Åström, who guided us through this process and always encouraged us. Finally, we want to thank our significant others and families for putting up with us this year and proofreading the text.

Contents

1 Introduction

This section aims to describe the underlying problem this thesis is based on and the target of the thesis. It also defines a context by discussing what has been done before – both in this project and in other papers related to the same problem. Finally, it provides a description of the structure of this report.

1.1 Background

Large organizations often produce and store a large amount of internal documents for their employees. The reason for this may vary but the common denominator is that the company itself wants the employees to get the information in these documents. In small organizations this is not as big of a problem since the amount of content is limited. But in larger organizations, the amount of content grows and it becomes a problem to find the documents you need. How can a system be created where the end users find the needle in the haystack?

Most systems used in larger companies for their internal communication are based on tools from a time when the amount of information was small enough to overlook. They require manual work from a human being in order to keep the data in check and up to date. Few systems have been able to transcend this, and therefore their qualities have decreased as their sizes have increased.

To overcome these issues a search tool was built. The aim was to make it simple to access and find the information needed in a fashion that the users were familiar with. To further improve the usability of the system the decision to learn from the user's behavior and to cluster documents based on their content was made. In this way, the possibility to present close relationships between documents but also tailor the experience for each user would be enabled.

1.2 Aim of the thesis

The aim of this thesis is to help our customer's employees find more relevant content. This will be done by investigating and implementing Machine Learning clustering methods into an existing information retrieval system. The emphasis in this report is on implementing and evaluating different approaches rather than performing a theoretical analysis or development of the used methods. This is done in order to create as much value for the customer as possible. The main questions the authors asked was "Can we replicate the human idea of structure to documents using software?" and "How does an unsupervised K-Means clustering perform compared to supervised Support Vector Machine in answering that question?".

Problem formulation

- Can we replicate the human idea of structure to documents using software?
- How well does unsupervised K-Means replicate human clustering?
- How well does a Support Vector Machine replicate human classification?

1.3 Previous work

Talkative Labs[1] were approached with the problems mentioned above. The client had loads of useful content and a large number of employees who needed access to it but was unable to connect the two. Several full-time employees were required simply to organize the documents but people were still struggling to find what they were looking for. The system relied heavily on navigating folder structures and browsing topics to find what you were looking for. There had to be a better way to provide the users with the content they need.

To solve their problem an API that could serve different front-end applications with data was built. On top of the API, a graphical user interface (GUI) was created, that the end users would interact with. In the first iteration the GUI consisted of only a search field.

The foundation on which this project stands on is a REST API. The API serves as the backend for the service that was initially built. The API handles all operations such as uploading/downloading assets, processing assets through creating previews and extracting text from them. All text data and metadata about the assets uploaded to the API is stored in a Mongo database with an elastic search index on top. This turned out to be a powerful solution for full-text search and content delivery. This was the information retrieval system that was already in place when this project started.

1.4 Related work

The paper "Clustering articles based on semantic similarity" written by Wang and Koopman [15] describes a similar situation to the one in this project. The authors have a large dataset that they want to cluster, but they do not have proper training data. They decided to use the unsupervised methods K-Means and Louvain community detection, and found K-Means to be very suitable for their needs. The paper describes K-means as "one of the simplest unsupervised learning algorithms that solves the well defined clustering problem It scales well to large number of samples and has been used across a large range of application areas" [15, p. 6]. This encouraged the use of the same method in this project.

Page seven in the same paper contains an interesting discussion about K-Means and mentions the importance of trying different values of K to find what amount of clusters a dataset naturally falls into. This approach was also used in this project in order to get to know the corpus better. In their paper they do not have any base truth at all for evaluation, so they construct one using the average silhouette method. They then use this to determine the optimal number of clusters in their K-Means clustering. Since the label data in this project is not as good as we thought it would be this approach was very interesting to read. What they lacked was a comparison to a supervised metric, so that was one of the goals of this project.

In the paper "A Comparative Study on Different Types of Approaches to Bengali document Categorization" [5] the authors investigate different supervised classification algorithms on Bengali documents. They find that an SVM with TF-IDF feature vectors performs better than any other approach used in the study. They note that this result is consistent with other studies that have been conducted.

[1] the company that initiated this thesis

In the paper [6] the authors compare four supervised algorithms (Support Vector Machine, Decision Tree, K Nearest Neighbours and Naive Bayes). They arrive at the conclusion that for large document sets the SVM classifier is superior to the others. SVM also surpassed its competitors in computational efficiency. They used the metrics precision, recall and F-measure to evaluate the results. It is also interesting that they ran some tests with fairly small training sets (under 250 samples per category). They find that SVM is not as good as the others for a small number of training samples but quickly passes the others as the number of samples grow. This is something that could have been interesting to evaluate for us when looking at the playlist model in our case.

Another interesting paper is "Using unsupervised clustering approach to train the Support Vector Machine for text classification" [8]. The title is self-explanatory and was immediately deemed interesting for this project. It would have been interesting to see how the K-Means data works as training input into the Support Vector Machine, and how that compares to the labels as training material. An even more interesting approach may have been to combine the knowledge of labels with the output of K-Means. Rather than creating training data out of no data, K-Means could be used to improve training data of low quality and then be fed to a Support Vector Machine. That was however outside the scope and timeframe of this project.

2 Theory

This section aims to describe the theoretical concepts that this thesis is based on. It starts with something as unusual as a description of an entrepreneurial term, but one that had great impact on to execution of this project. It then describes the theoretical foundations for the different methods and concepts used within the project.

2.1 LEAN

Lean is a "system" for how to develop a successful small business that resembles the scholarly method [11]. It starts out with some ideas about how to design a product and create value for the customers. The essence of the system is to test these ideas as early as possible, to promote learning and informed decisions. The first step is to define metrics for success and designing tests. The tests should be designed to be cheap to perform and fast to implement. The project then moves on to creating an "MVP" – a Minimum Viable Product. An MVP is the smallest, simplest, fastest and cheapest way to bring value to the customers. Then the project moves on to test the idea. The tests give feedback both on whether the idea corresponds to the customer's needs and on whether the project is progressing. If that is the case, persevere and fine tune. If that is not the case, pivot into another, more productive, direction.

The LEAN system, like many business systems, sounds obvious by itself but really shines when compared to how things usually are done. It is not uncommon to start with some research and then work for a long time to perfect the product. The fear of releasing an unfished product takes the overhand but at the cost of missed feedback from the customers. The problem is that it is very easy to stray off course, or fail to notice that the demand has changed. This could result in unpleasant surprises in the late stages of the project. With a lean approach, these surprises can be detected early on and the project can adapt to it.

The LEAN system versus more traditional development methods also resembles the relationship between working in an agile way versus the waterfall model in software engineering. In this thesis, the LEAN model has been adopted to best fulfill the customer's needs.

2.2 Text operations in text analysis

The parameters used in the vectorizer was maximum and minimum document frequency, a maximum number of features, a list of stopwords and the n-gram range.

2.2.1 Stopwords

Some words do not really contribute anything to the meaning of the document or are so common that they do not help differentiating documents. These words are not included as terms in the machine learning data model. They are called stopwords. The NLTK [2] contains a list of such stopwords for different lan-

guages. Examples are me, your, up, further and very[2]. The decision to add some words that are common in this specific corpus was also made.

2.2.2 Tokenization

Tokenization is the process of moving from a list of characters to a list of words, or "tokens", also referred to as terms [7, p. 22]. The list of characters is split wherever a space or a non-word character is encountered. There are of course some special cases, such as "aren't" and "co-education" that has to be handled in a special manner [7, p. 24].

2.2.3 n-grams

Some words belong together. This goes for both hyphenated words, that might be split by the tokenization process mentioned above, and words that are actually separated by a space. The most classic example is probably "San Francisco", but other common patterns are names (John Doe) and some terms of art (machine learning). Words that frequently appear together lose some meaning if they are split up. Therefore, words that occur together frequently are stored as a single term in the data model. A term with two words is a bi-gram, and a term with n words is an n-gram.

2.2.4 Stemming

Stemming (or lemmatization) is the process of removing or replacing parts of words, usually affixes, so that two words with the same meaning end up in the same term. Stemming is performed by removing affixes so that car, cars, car's and cars' all end up within the same term car. Lemmatization uses a lexicon and understanding of grammar, so that terms like be, am, are and is all end up within the term be [7, p. 32].

2.2.5 Ignoring common/uncommon words

The final text operation used in this project is automatic removal of words that are extraordinarily common or uncommon in a document. This is done to clean up the TF-IDF matrix and to speed up the learning process. The TF-IDF vectorizer automatically discards any words above or below a given threshold when it creates the TF-IDF matrix. Since they would not affect the outcome much anyway they might as well be removed in order to speed up the process.

2.3 TF-IDF

The data model used for this machine learning project was the TF-IDF model, where each row represents a document and each column represents a term (or feature) of the corpus. The term can be either a word or an n-gram. Each value in this matrix represents how common a specific term is within a specific document. The TF-IDF matrix is calculated in three steps: the term frequency step, the inverse document frequency step, and the combination step.

[2]The complete list of 127 stopwords in NLTK can be found here: `https://gist.github.com/sebleier/554280`

The first step is quite simple. In order to assign a weight for a term in a document we simply count the number of occurrences of that term in the specific document. This is defined as the term frequency $\text{tf}_{t,d}$ [7, p. 117].

The second step is more interesting. The issue with term frequency is that it handles all words in the same manner. A word with high occurrence in a document will weigh more than a word with a low occurrence, even if that word is very common in the entire corpus. A more desirable outcome would be that words that are uncommon in the corpus but common in a document are assigned a greater weight.

That is exactly what inverse document frequency does. The inverse document frequency for a term idf_t is defined as

$$\text{idf}_t = \log \frac{N}{\text{df}_t}, \tag{1}$$

where N is the total number of documents in the collection and df_t is the document frequency, the number of documents containing the term t.

In the end we store the TF-IDF value, defined as

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t. \tag{2}$$

The TF-IDF vectorizer performs this operation for a corpus of documents. In order to clean up the TF-IDF matrix it also takes some parameters defined in section 3.5.2. In this project a list of stopwords was used, as well as a maximum and minimum document frequency, a maximum number of features, and an n-gram limit.

2.4 K-Means

K-Means is a clustering method based on minimizing the distance between the documents of a cluster and its "center point", its centroid [7, p. 360]. The most important property of a cluster in K-Means is the centroid. The centroid μ of a cluster ω is calculated as

$$\mu(\omega) = \frac{1}{|\omega|} \sum_{\mathbf{x} \in \omega} \mathbf{x}. \tag{3}$$

The goal of the K-Means clustering is to minimize the Euclidian distance between the centroid and all items belonging to a cluster. This distance is minimized via the objective function "Residual Sum of Squares":

$$RSS_k = \sum_{\mathbf{x} \in \omega_k} |\mathbf{x} - \mu(\omega_k)|^2, \tag{4}$$

$$RSS = \sum_{k=1}^{K} RSS_k. \tag{5}$$

First, each item is randomly assigned to a cluster, and the centroids are calculated. The algorithm then reassigns documents to belong to the cluster represented by their closest centroid, and the centroids are recalculated. These two steps are repeated until the clustering is done.

The clustering is done when one of four things happen. One, the RSS value becomes smaller than a predefined value (the clustering is good enough). Two,

the RSS value decreases more slowly than a predefined value (what we're doing is not enough). Three, the clustering stops changing between iterations (there is nothing left to do). Four, the number of iterations becomes larger than a predefined value (we do not have time to wait anymore).

There is no guarantee that the algorithm finds the global optimum, so it is usually executed several times with different initial conditions to improve the overall results.

2.5 Support Vector Machine

The idea behind using a Support Vector Machine (SVM) in this project is to train a system to recognize documents that would end up in the same folder if they were to be sorted by a human. The main principle of an SVM is to determine one or more separators in the feature room which best separates the classes (or folders, to use an analogy) [1, p. 194].

The simplest example of an SVM is a binary classifier which separates two different classes with a hyperplane. If the training data is easy to separate this can easily be done with a linear approach. The optimization problem can be described as finding the plane with the largest margin to the support vectors, as seen in figure 1.

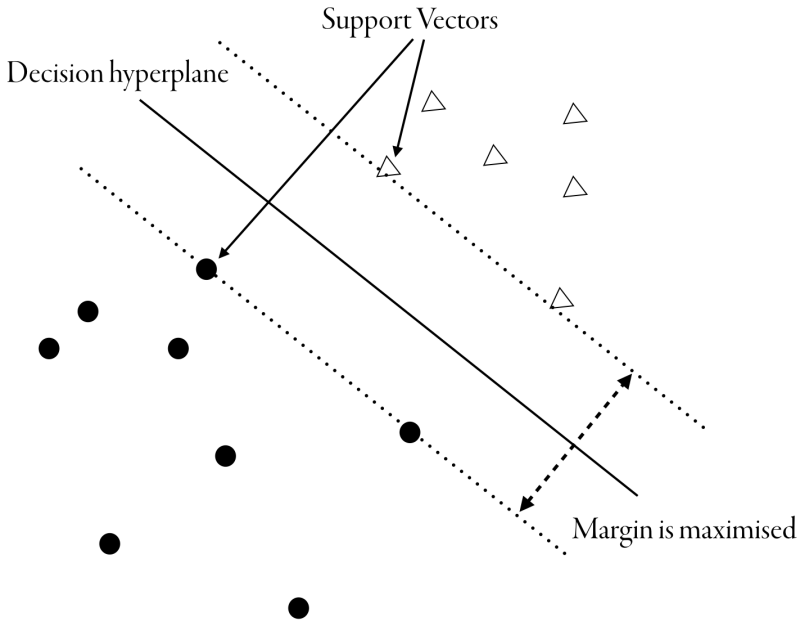

Figure 1: The main principle of an SVM illustrated.
The target of the optimization problem is
maximizing the margin between the support vectors.

If one wants to separate more than two classes the problem becomes more complex. The most common technique according to [7, p. 330] is to build a

one-versus-rest (ovr) classifier that classifies one class at a time against all the others.

All of the above applies to input data that is linearly separable, but in the general case for text classification the that is not the case [7, p. 327]. One way to solve this is the "kernel trick". It's performed by mapping the input data to a higher dimensional space and then using the linear classifier in that space.

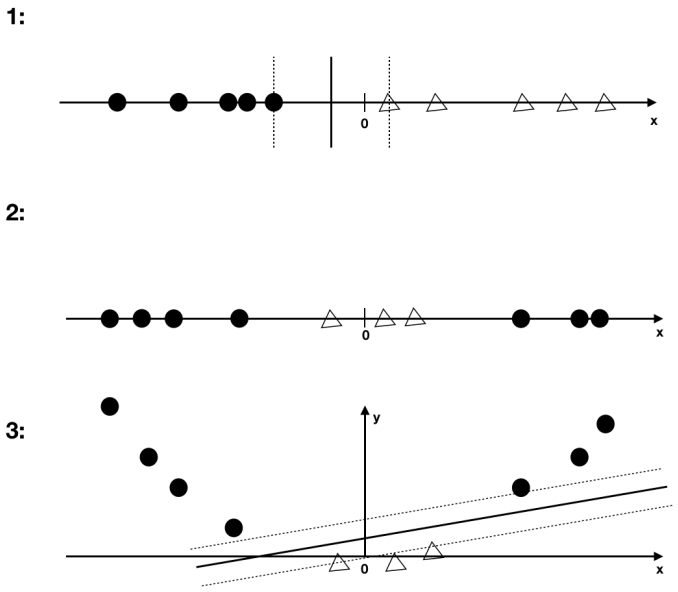

Figure 2: The first example is easy to linearly separate. The second one is more problematic. Using the kernel trick to map it into a higher dimensional space (case number 3) makes it possible to separate them with a linear approach [7, p. 333].

2.5.1 Kernel functions

scikit-learn supports four different kernel functions [13] out of the box. It also has a fifth option to provide a custom kernel function. This makes it possible to test different kernel functions to find the one that best separates the classes. The radial basis function is the default kernel function, and the linear kernel function the simplest one. The polynomial and the sigmoid kernel functions were also included in the model. The kernel functions are described in table 1.

Table 1: Mathematical definitions for the SVM kernel functions.

RBF	$\kappa(\mathbf{x}, \mathbf{x}') = e^{-\gamma\|\mathbf{x}-\mathbf{x}'\|^2}$
Linear	$\kappa(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$
Polynomial	$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma\langle \mathbf{x}, \mathbf{x}' \rangle + r)^d$
Sigmoid	$\kappa(\mathbf{x}, \mathbf{x}') = (tanh(\gamma\langle \mathbf{x}, \mathbf{x}' \rangle + r))$

2.6 Norms

2.6.1 Euclidian norm

The Euclidian norm is used in the K-Means clustering to determine the distance between documents and the centroid. It calculates the distance between the tips of two vectors. The Euclidian distance between two vectors is defined as

$$\|\mathbf{x} - \mathbf{y}\| = \sqrt{\sum_{i=1}^{M} (x_i - y_i)^2}, \tag{6}$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors [7, p. 131]. The euclidian norm also goes by the name "L2-norm".

2.6.2 Cosine norm

The cosine norm is used for visualizing data and for calculating document neighbors via document similarity. It calculates the angle between two vectors. In this project it was applied to feature vectors in the TF-IDF model to determine their similarity. The cosine similarity of two vectors is defined as

$$\text{cosine similarity} = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}, \tag{7}$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors [7, p. 121]. The metric returns the angle between the two arrays independent from the length of the array. 1 for zero degree difference, 0 for 90 degree difference and -1 for 180 degree difference. In the TF-IDF model, all feature vector components are positive, so the result of the cosine norm is bounded by 0 and 1. Two documents in the model are similar if the similarity is close to 1, and dissimilar if the similarity is close to 0.

2.7 Multidimensional Scaling

Moving from a set of coordinates to a table of distances is an easy mathematical problem - just select a norm, calculate the pairwise distances and store it somewhere. The inverse problem, using distances to create a set of coordinates, is more complicated.

Multidimensional scaling (MDS) is a method for visualizing distance matrices. "The goal of an MDS analysis is to find a spatial configuration of objects when all that is known is some measure of their general (dis)similarity." [16, p.9].

There are two branches of MDS: metric and non-metric. The metric, simpler, version assumes all distances are actual euclidian distances. The nonmetric, more general, version makes no such assumption and is therefore more complex. The most common approach to nonmetric MDS is to define some measurment of "stress", that measures how much a given set of coordinates differ from the given distance matrix. The target for the MDS method is to minimize this stress.

In this project, the manifold package in scikit-learn was used to achieve a visualization of the clusterings. It uses the SMACOF algorithm, short for "Scaling by Majorizing a Complicated Function" [3, ch. 8]. SMACOF starts out with a random coordinate assignment. It then calculates a specific stress matrix for the assignment. In order to minimize the stress, the Guttman transform is performed on the coordinates X [3, p. 190]. For the reader interested in the specifics of the stress matrix, and the Guttman transform, reading chapter eight of the title "Modern Multidimensional Scaling" [3] is recommended. After the transform the stress is calculated again. If it satisfies a predetermined condition, or if it has reached the maximum number of iterations, then it terminates. If not, the transform is repeated. The process is described in figure 3.

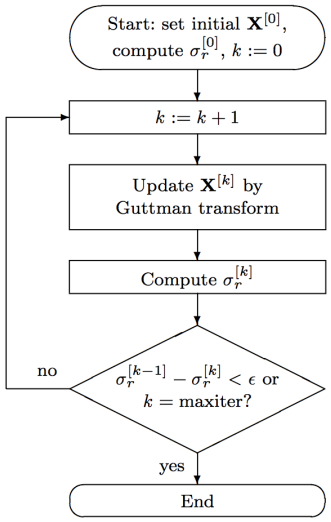

Figure 3: An overview of the SMACOF algorithm.
Source: page 192 of [3].

The purpose of MDS in this project was to provide a way to visualize the dataset and the clusterings. While TF-IDF is a great format for computers, the MDS is the peeking hole for us humans to understand what is going on.

2.8 Metrics

In order to evaluate the results of this project, a selection of information retrieval metrics was selected. Each of them require both the outcome of a clustering and a set of labels (a ground truth) to compare them to. These will be explained in this section of the report.

2.8.1 Purity

Purity is the first evaluation metric for clusterings. When calculating purity each cluster is assigned the label that is most frequent in that cluster. The accuracy of the clustering is then calculated by counting the number of "correctly" assigned documents divided by the number of documents. Mathematically purity is defined as

$$\text{purity}(\Omega, \mathbb{C}) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j| \,, \tag{8}$$

where $\Omega = \{\omega_1, \omega_2, ..., \omega_K\}$ is the set of clusters and $\mathbb{C} = \{c_1, c_2, ..., c_K\}$ is the set of labels used for evaluation [7, p. 356]. A number close to one represents an accurate clustering and a number close to zero represents a bad clustering. It is also important to note that a high purity is more easy to achieve when the number of clusters is high.

Purity is a metric that is very intuitive and easy to grasp. It also has the advantage that it can be calculated per cluster, not just per clustering. This provides accuracy for the analysis of the clusterings.

2.8.2 Entropy

The second metric used to evaluate clusterings is entropy. Entropy is a way to measure the amount of order or disorder in a system. A low entropy means that the clusters are well ordered and predictable, while a higher entropy means that the clustering is more chaotic. The entropy H of a cluster ω is defined as

$$H(\omega) = -\sum_{c \in C} P(\omega_c) \log_2 P(\omega_c) = -\sum_{c \in C} \frac{|\omega_c|}{n_\omega} \log_2 \frac{|\omega_c|}{n_\omega} \tag{9}$$

in [7, p. 99-100], with added maximum likelihood estimates of the probabilities. $|\omega_c|$ is the number of documents classified as c in cluster ω, and n_ω is the number of documents in cluster ω. The total entropy of a clustering was then defined as the weighted average of each cluster, calculated as

$$H(\Omega) = \sum_{\omega \in \Omega} H(\omega) \frac{N_\omega}{N}, \tag{10}$$

where Ω is the set of clusters $\{\omega_1, \omega_2, \cdots, \omega_k\}$, N is the total number of documents and N_ω is the number of documents in cluster ω. Unlike all other metrics, zero is a perfect score for this metric, and a higher value means that the clustering is less successful.

The entropy metric has the same two advantages as purity: it is intuitive [3] and it can be calculated per cluster, not just per clustering.

2.8.3 True or false, positive or negative

In order to understand our next few metrics, a few terms need to be introduced. When comparing a computed clustering to a set of labels (or "keys"), four different cases arise. When a document occurs in a cluster, and it should, it is a

[3] at least for us physicists...

true positive. If a document occurs in a cluster, and it should not, it is a false positive. Similarly if a document does not occur in a cluster and it should nott, it is a true negative. If a document does not occur in a cluster, but it should, it is a false negative. Table 2 summarizes this paragraph.

Table 2: True and false positives and negatives

	Relevant	**Non-relevant**
Retrieved	true positives (tp)	false positives (fp)
Not retrieved	false negatives (fn)	true negatives (tn)

2.8.4 Precision and recall

The next two metrics are precision and recall. Precision is defined as the percentage of the retrieved documents that are relevant, or true. Recall, on the other hand, is the percentage of the relevant, or true, documents that are retrieved. Mathematically they are defined as

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{11}$$

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{12}$$

Intuitively precision ensures that the received results have a high relevancy, while recall makes sure the user is not missing out on anything important. Precision and recall are calculated for the SVM model in its cross validation phase, but is not calculated for the K-Means clustering.

2.8.5 F-score

The next metric is F-score. F-score combines the concepts of precision and recall using the weighted harmonic mean. This is defined as

$$\mathrm{F}_\beta = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}, \tag{13}$$

but with $\beta^2 = \frac{1-\alpha}{\alpha}$ this can be written more compactly as

$$\mathrm{F}_\beta = \frac{(\beta^2+1)\mathrm{PR}}{\beta^2\mathrm{P} + \mathrm{R}} \tag{14}$$

$\beta \in [0, \infty]$. In F-score the parameter β can be chosen arbitrarily. A higher value penalizes false negatives harder, a lower value penalizes false positives harder. $\beta = 1$ emphasizes precision and recall equally. $\mathrm{F}_\beta = 1$ is a perfect score for this metric, and a lower value means that the clustering is less successful.

F-score is calculated for the SVM model in its cross validation phase, but is not calculated for the K-Means clustering.

2.8.6 Homogeneity completeness and V–measure

The next set of metrics are homogeneity score (hs), completeness score (cs) and V–measure score (vms). These three belong together, just like precision, recall and F-score do. They were selected primarily because of their intuitive meaning, they really tell something about the properties of the clusterings. Homogeneity means that "each cluster contains only members of a single class" [12, p. 411]. Completeness means that "all members of a given class are assigned to the same cluster" [12, p. 411]. When creating a clustering it is quite easy to maximize one or the other of these two metrics, and one might prioritize one over the other in some cases. But a great clustering should seek to maximize both, so V–measure score is defined as the harmonic mean of the homogeneity score and the completeness score. A high V–measure score is achieved by making a clustering which is high in both homogeneity and completeness, so the V–measure score may be considered a more "broad" or "general" metric for a clustering. All of the three metrics take values between one and zero, where one is the maximum score (an entirely homogeneous and/or fully complete clustering) and zero is the minimum score.

One drawback to this group of metrics is that they are not chance normalized, so one has to be careful when the number of clusters is large or the number of items is small. The V–measure score is equivalent to "Normalized Mutual Information", which is another very common metric of a clustering.

Mathematically, these metrics are defined as

$$h = 1 - \frac{H(C|K)}{H(C)}, \tag{15}$$

$$c = 1 - \frac{H(K|C)}{H(K)}, \tag{16}$$

$$v = 2 \cdot \frac{h \cdot c}{h + c}, \tag{17}$$

with the conditional entropy of classes C given cluster assignment K $H(C|K)$ and the entropy of classes C $H(C)$ defined as:

$$H(C|K) = -\sum_{c=1}^{|C|} \sum_{k=1}^{|K|} \frac{n_{c,k}}{n} \cdot \log\left(\frac{n_{c,k}}{n_k}\right), \tag{18}$$

$$H(C) = -\sum_{c=1}^{|C|} \frac{n_c}{n} \cdot \log\left(\frac{n_c}{n}\right). \tag{19}$$

2.8.7 Rand index

This metric is useful for understanding another metric in this project, even though it is not used in itself. Based on the terminology in section 2.8.3, the Rand index is defined as

$$\mathrm{RI} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}}. \tag{20}$$

Intuitively the Rand index is a measurement on how many percent of the documents that were clustered correctly. A Rand index of one is a perfect score, and lower values indicate a bad clustering.

2.8.8 Adjusted Rand score

The adjusted Rand score (ars) is based on the same principles as the Rand index, but with two major differences. The first being that it ignores permutations. The actual names of the clusters do not matter, and it does not need a mapping between which of the generated clusters that should correspond to which of the ground truth clusters. The second being that it uses chance normalization. This means that the method "projects" a random clustering to the value zero by subtracting the expected value $E[RI]$, and then scales the values so that 1 is a perfect score and minus one is the worst achievable score.

The Rand index RI can also be written as

$$RI = \frac{a+b}{C_2^{n_{samples}}}, \tag{21}$$

where a is the number of pairs that are in the same set in both clusterings, b is the number of pairs that are in different sets in both clusterings and $C_2^{n_{samples}}$ is the number of possible pairs in the dataset. a corresponds to the concept of "true positives" and the b corresponds to "true negatives" in section 2.8.3, but does not actually use the "label names" of the clusters. Equation 21 is analogous to equation 20. The adjusted Rand index ARI of a clustering is then defined as

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI]}, \tag{22}$$

in order to perform the chance normalization. Since it considers both precision and recall it is considered a "broad" metric.

2.8.9 Fowlkes Mallows Score

Fowlkes Mallows score (fms) is another "broad" metric that tries to consider all aspects of a clustering. It is based on pairwise precision and recall. Mathematically it can be defined as

$$\text{FMI} = \frac{\text{TP}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})}} \tag{23}$$

in [14, p. 6], borrowing notation from the previous section 2.8.3. A value close to one is an indication of a good clustering and a value close to zero of a bad one. The metric is chance normalized so that random label assignments have a Fowlkes Mallows score close to 0. It is also important to note that fms will always assume high values for very low numbers of clusters.

3 Method

This section aims to describe the execution of the project. It begins by describing the approach and explaining some obstacles that affected the choice of methods. It then goes on to describe the first naive steps of the project and some remarks about the kinds of data that was available. The most important part is code implementation, which describes how the machine learning system is built and structured. Finally, it describes the architecture the project was executed on.

3.1 Approach

3.1.1 LEAN

A very important part of the approach has been the LEAN model. The primary target, or "metric for success", was not a theoretical question. It was: *how far can two engineering students go in creating a useful machine learning system*? Since that was not clear at first the decision to work in iterations was made, based on the customer's feedback. Build, measure, learn, repeat. Not read, think, do, done. For us this meant two things.

First, customer preference over our preference. The work was divided into small iterations that were presented to the customer. In between each iteration there was feedback and new requests that affected the direction and momentum of the project. This did turn out to be quite different from the method we were used to from the university, where most things are planned and have a structure. Lectures and readings cover material that is later required to solve the problem. Introduction, theory, method, results, discussion. In the corporate world result is king, discussions come before method, and theory is kept to a minimum.

Balancing customer demands and keeping track of the problem formulation has been challenging but also rewarding. With a more classic approach we would have ended up with a very different solution, but with a lot less knowledge.

Second, implementation over theory. This project is all about what could be done and maintained within the timeline and/or budget, and what results could be achieved. The customer always wants results.

These circumstances may be worth keeping in mind whilst reading this report and understanding the decisions made along the way. These were the internal factors but there were also some external factors that affected our decisions. They will be presented in the following section.

3.2 Challenges

The previous section outlined the benefits of the chosen approach. This section will move on to some obstacles. Given below are four challenges that resulted in some road changes.

3.2.1 Developing in a live environment

The challenges of doing a project at a company, for a customer, has been very rewarding for us and the final outcome of the project. Nevertheless, it also made the journey more difficult. Things have taken more time than we had envisioned. Likewise, regarding the programming; writing a script that runs on

your computer when you need it to is one thing, but integrating your code into a codebase that does not allow any downtime is a different thing. This has been challenging and fun, but has also slowed us down quite a bit. A lot of time was spent writing API:s and integrations that would not have been required if the project's only purpose was this report.

3.2.2 Corporate politics

Our greatest obstacle to overcome has been corporate politics. During the first months of this project everything was running fine and we counted on an inflow of users according to a prognosis. We therefore decided to focus on supervised machine learning systems: systems that collect and make sense of user data. Suddenly this changed, due to corporate politics, as it was decided that we would wait an additional month before adding users. We kept working on the supervised systems. The month quickly turned into two months and we were getting worried. Eventually, we realized that we could not rely on user data and had to change the direction of this project altogether and look for alternate data sources. The data source we ended up with is explained in section 3.4.

3.2.3 Search optimization

In the early stages of the project, a lot of time was spent on researching how to use machine learning to improve the search results of the service. This made sense as searching is the number one way for users to interact with the information platform. Unfortunately this turned out to be a dead end.

3.2.4 word2vec

As a first step word2vec was used on the text of the entire dataset [1]. Raw text data was downloaded from the database and fed into the python implementation of Google's word2vec and training and clustering of the terms occurred. This provided insights on how the terms in the documents were related but did not directly help to understand the relationships between documents. The method was dropped in favor of other methods, more focused on documents.

3.3 Naive first steps

With all these things in mind, how does one go about investigating machine learning methods that replicate the human idea of structure? Since the problem formulation did not imply a specific method the first couple of weeks mainly consisted of experiments to figure out the road ahead and to get acquainted with the different available methods for machine learning.

3.3.1 TF-IDF demo

The first step was constructing a TF-IDF matrix and a clustering with K-Means. This quickly became the primary method of clustering in this project. It also seved as an introduction to a lot of machine learning concepts and to scikit-learn [9]. The Python library scikit-learn is a popular collection of tools for machine learning.

A list of file hashes and all the text content of the files were loaded into the system. A number of operations were performed on the text. All non-text characters were replaced with spaces. Stopwords were removed using a corpus from Natural Language Toolkit [2]. Finally the text was tokenized and stemmed. This data was then loaded into TfidfVectorizer, a feature extraction function from scikit-learn, that generated the TF-IDF matrix from the contents of all documents. The vectorizer also automatically ignored the most and least common words.

With the TF-IDF matrix in place, all pairwise distances were computed using the cosine norm. This data was used for visualizing the data. The clustering instead relied on scikit-learns K-Means tool that uses the TF-IDF matrix and number of clusters as inputs.

3.3.2 Cluster visualizer

Finally, the need for a way to visualize the documents and their feature vectors emerged. In order to get a visualization the scikit-learn tool MDS was used, short for "Multi-dimensional Scaling", and the classic python tool Matplotlib [4]. MDS uses the cosine norm distance matrix and generated a 2D representation of all documents. These were plotted in matplotlib as a scatter plot with colors based on cluster.

These were the first steps to probe the problem and get used to some machine learning methods and libraries. In order to make something truly useful, a better structure and method for evaluation was required.

3.4 The Data

As is always the case with machine learning solutions, the data is the most essential part. We will therefore elaborate what kind of data we had access to and how it was processed.

3.4.1 The corpus

The corpus consists of approximately 18400 pdf documents. They are internal guidelines, product specifications, safety labels, slide-decks and reports written for the employees of the company. The scope of this project is only raw text extracted from the pdfs, not images or layouts.

3.4.2 The user data

The user data comes in two different forms. One, in the form of *playlists* and the other in the form of user anlytics, which will be explaind further on. Users, and admins, can create playlists containing one or several documents from the corpus. A playlist provides two pieces of information. Firstly, each document in the playlist is somehow related to the words in the title of the playlist. Secondly, each document in that playlist is somehow related to all of the other documents in the playlist.

The other form of user data comes from Segments[4], a web user analytics tool. It was configured to track events such as "The user performs a search" and

[4] https://segment.com

"The user navigates through the preview". This connected documents to each other, and documents to search queries. Since the information retrieval system had very few users none of this data could be used in this thesis.

3.4.3 URL data

The old system had a unique URL for each document. This URL was retrieved and stored in the data model and became the training data for this thesis.

The full URL:s were unique, so they would not work as training data. The solution to this was to only look at a certain "level" of the URL. Level = 0 only looked at the first part of the URL, for example `/Performance`. Level = 1 looked at the first two parts of the URL, for example `/Performance/Lists`. In this way, different sizes of "clusters" or "classes" could be generated by varying the level parameter. This was then used as training data for the supervised method, and as evaluation data for the metrics.

3.5 Code implementation

This section will explain the machine learning process based on the structure of the code. A short overview is presented below in list and diagram form (figure 4), each one is then described by a subsection. Each part of the implementation was named after a famous character from Sir Arthur Conan Doyle's legendary story Sherlock Holmes.

- The API handles input and output and starts calculations.
- Hudson handles data extraction and preparation.
- Mary creates the TF-IDF model.
- Sherlock performs a K-Means clustering.
- Mycroft performs an SVM classification.
- Moriarty calculates the metrics.
- Grid Visualizer visualizes the clusters.
- Godfrey performs the MDS on the TF-IDF distances.
- Magnifier.js visualizes the result from Godfrey.

3.5.1 Data preparation - Hudson

The first step of the machine learning process is Mrs. Hudson. Hudson handles the initial data streams described above. It establishes a connection to the live Robomongo database to collect data. Its first step is to get all document ids and text data. The text data is stripped of line breaks and non-ASCII characters and then stored in an array. The ids are inserted as keys in a table that will later be used to store labels and playlists.

The next step is to get paths for all documents with a path stored on them. The path (or URL), as covered above, is an indication on which folder a content creator has once put the document in. The user gets to select what level folder

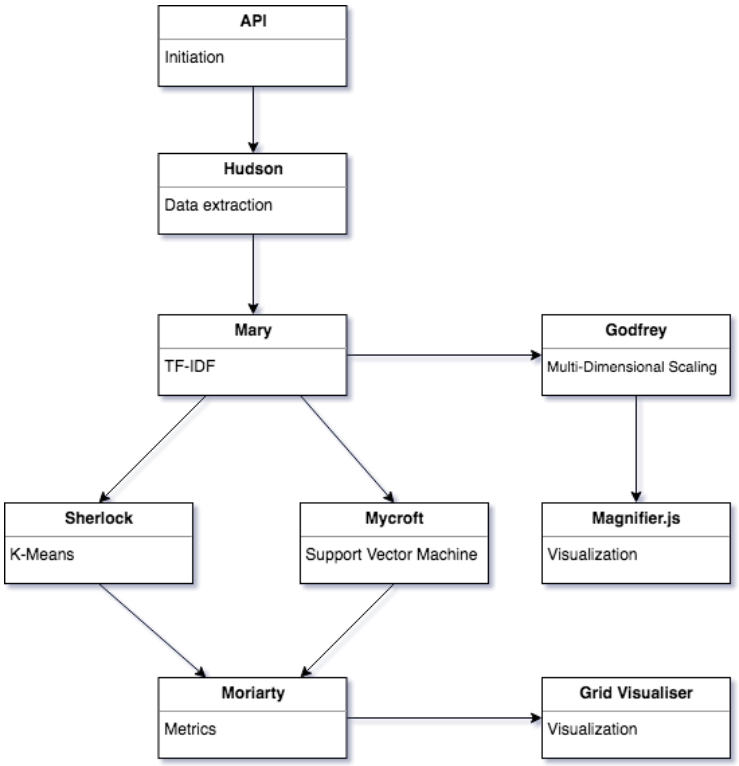

Figure 4: An overview of the code structure.

that should be extracted. For example, the path `/Resources/Presentations/Productivity/Turkey` could result in the label "Resources" with level = 0, or "Turkey" with level = 3. All document ids associated with a path are updated in the table.

After that, another request to the API is performed, asking for all playlist ids and the document ids the playlists contain. The same table is updated with this information.

Except for this massive table, two different lookup tables are created. One stores the document playlist assignment sorted by document id, the other stores the document URL path tags sorted by document id. Finally, all text data and the data tables are JSON-encoded and uploaded to an Amazon S3 bucket.

3.5.2 TF-IDF – Mary

The second step of the machine learning process is Mary. Mary performs some text operations to make the text material more suitable for training and then creates the TF-IDF matrix and the distance matrix.

Just like in all steps, the first thing that happens is that data is loaded from S3. Mary uses text.json, containing all text data, and data.json, containing all document ids, URLs and playlist ids.

The next step is tokenizing and stemming. The text from each document is sent into two different stemmers in the nltk python package [2]. One tokenizes by word and one tokenizes by sentence. The tokens are then filtered so that

tokens containing no letters are removed. These filtered tokens are then sent into the nltk Snowball Stemmer, set for English language. The stems and the filtered tokens are then stored.

Once the tokenizing and stemming is complete it is time to create the TF-IDF matrix. This is performed using the scikit-Learn TfidfVectorizer class. The parameters that were sent into the vectorizer were maximum and minimum document frequency, maximum number of features, the list of stopwords and the n-gram range. All of these parameters are described in section 2.3.

The vectorizer performs its fit_transform method on the document contents. This results in a TF-IDF matrix and a list of feature names.

With the TF-IDF matrix in place, it has become time to calculate the distance matrix. The distance matrix contains the pairwise distance between all documents, calculated using the cosine norm from scikit-learn metrics. The distance matrix stores 1− the value returned from cosine_similarity, so that 1 represents the maximum distance and 0 represents the minimum distance. This matrix is then converted into a sparse csr matrix (Compressed Sparse Row Matrix) using a scipy function.

The TF-IDF matrix, the distance matrix and an array with the document id order of the TF-IDF matrix are uploaded to S3.

3.5.3 K-Means - Sherlock

The third step of the machine learning process is Sherlock. Sherlock performs a K-Means clustering of all the documents based on the TF-IDF matrix.

Just like in all steps, the first thing that happens is that data is loaded from S3. Sherlock uses the TF-IDF matrix, the associated list that explains which row that corresponds to which document, and some additional metadata about the documents.

Before the clustering happens, the number of clusters to divide the corpus into is decided. It could be passed in as a parameter, but it is usually calculated based on the labels the calculation is going to be benchmarked against. A computation that will be compared to a facit of 10 keys will be clustered into 10 clusters.

The k-Means clustering is performed by calling the scikit-learn K-Means function. The only input parameters are the number of clusters to generate and the TF-IDF-matrix. The cluster assignment for each document is returned in list form.

When all calculations are done the clustering results are uploaded to S3.

3.5.4 SVM - Mycroft

The fourth step of the machine learning process is Mycroft. Mycroft performs a classification based on two different training methods. The first method was used to evaluate how well it performed. The second method was used to train the model to be used in the platform. The performance of the two different methods are much alike, therefore only the method for evaluation will be described, as the other follows the same pattern.

The input to the SVM model is based on the earlier scripts in the pipeline. The feature vectors are taken from the TF-IDF matrix computed in Mary. The labels are given as the path to the folder that each file used to reside

in, as described in section 3.4.3. The associated folder structure depth can be set before computation to test different numbers of distinct labels but also to find underlying correlations based on how a content creator has classified the documents before. This list of labels is computed by the Hudson script. The feature vectors (TF-IDF-matrix) and labels are downloaded from S3.

To be able to evaluate the model the training set was split into two. The largest set was used to train the model and the smaller set was later used to cross-validate the classification. The results of the evaluation can be found in the result section below (4.3). The default split ratio is that 3/4 of the data is used for training and 1/4 of the data is used for the evaluation. In order to evaluate which kernel function that performs best the algorithm was executed with different kernel functions and input parameters.

The last thing that happens in Mycroft is the evaluation of the model. For each computation, three different metrics are calculated with scikit-learn's built-in metrics calculator and stored in the database. The metrics are precision, recall and f-score (defined in sections 2.8.4 and 2.8.5). Another set of metrics are calculated for both K-Means and SVM. This calculation is performed in Moriarty (section 3.5.8). The output data from Mycroft is the trained model and a list of validation vectors and labels. This is serialized and uploaded to S3.

3.5.5 Finding similar documents - Deduction

After the distance matrix has been calculated (as described in section 3.5.2) the 100 closest neighbors to each asset was stored in the mongo database. The idea behind this was to quickly be able to display related documents in the GUI without having to do any calculation "on demand". At one stage in the project some progress was made towards creating a discover page where the users could explore the dataset based on the distance between the documents. This was however depreciated in favor of other functionality. The customer was satisfied with being able to see the closest neighbors of a document.

3.5.6 Visualizing clusterings - Godfrey and Magnifier

The purpose of the visualization tool is twofold. First, it has proven to be very helpful for us in evaluating the machine learning models. A visual representation of the data is more intuitive than a metric based. Even though metrics actually give a more fair measurement of the quality of a clustering the visual representations has been quite helpful.

Secondly, the visualizing tool is useful for understanding the corpus even outside of the development of the machine learning tools. The content creators at the company can use the visualization tool to find duplicate documents or different versions of the same document. It can be used to find "lacking" documents (there should be five versions of this document, there is only four versions) or find where new documents are needed (this document exists for Europe and Asia, but not for America). In the long run it could even become a convenient way to navigate through the document database.

The data for the visualization tool was created in a separate python script called godfrey.py. This was vaguely based on the "naive" visualization tool used in the early stages of the project. The distance matrix for the documents was

loaded and then processed with the Multi-Dimensional Scaling tool built into scikit-learn [9]. The scikit-learn method MDS.fit_transform is based on the SMACOF algorithm, described in section 2.7. This creates a representation of the documents, with proper (or close to proper) distances, in a two-dimensional plane or a three-dimensional room. These coordinates were then processed with Pandas, a Python data analysis library, in order to associate each point with a document id and a cluster. The result was exported as a CSV file. A filter that selects n random documents from the corpus to plot in order to create simpler, more interpretable figure was used as well.

The resulting CSV file was then parsed into D3[5], a javascript library for data visualization using the SVG data format. The first test was a D3 example written by Jonas Petersson [10].This was later tweaked to look good with the data from this project. With the use of information panels it was possible to see information about clicked documents and scroll through the pages. Finally the tool was integrated into the plaform for content creators.

3.5.7 Visualizing clusterings - Grid-visualizer

The primary method of evaluating the clusterings will be the mathematical metrics, but some intuition based methods were also used. Both for ourselves and for the customer. Overwhelmed with the level of abstraction and sheer amount of data presented in Magnifier, the development of yet another method for visually evaluating the clusterings started. Grid-visualizer randomly selects some items from each cluster, and creates a "mosaic" with their preview images (the first page of the PDF) as a webpage. This gives a quick intuitive overview of how successful the clustering has been.

3.5.8 Evaluation - Moriarty

The next step of the machine learning process is Moriarty. Moriarty is a toolbox of metrics evaluating the quality of clusterings. Initially data from the clustering of Sherlock, the labeling of Mycroft and the base truth from Hudson were loaded from S3. This data was also permutated to create some additional helpful variables used in the metrics.

Moriarty contains the metrics purity, entropy, homogeneity, completeness, V-measure, Adjusted Rand score and Fowlkes Mallows score. Each metric functions independently and returns its score to the main function. Purity and Entropy were written from scratch, based on the theory described in section 2.8.1 and 2.8.2. They are extra interesting because they provide a per-cluster metric, while the other ones return one value for the entire clustering. The remaining metrics were calculated using scikit-learn methods.

Two additional things were calculated, besides the actual metrics, to assist in analyzing the result of the clusterings. First, the parameter "counts" was filled with information about the number of documents assigned to each cluster. Second, the parameter "toplist" shows how many of each of the base truth labels are assigned to each cluster. These both would prove to be useful tools in understanding the other metrics.

When all calculations were done the metrics data objects are stored in the Mongo database.

[5] `https://d3js.org`

3.6 Architecture

3.6.1 Storage of the data

All the data in this project was either stored in a Mongo database[6], in a Redis[7] database or on Amazon S3[8]. The Mongo databases was used for the entire web service before the machine learning parts were added. All the document text data, all path data, and all playlist data was therefore collected from the Mongo database. The queue system for the machine learning computations was stored on Redis. The S3 bucket was used for all other data, mainly ouput files from the different steps of the machine learning process but also the documents themselves and preview images.

3.6.2 Docker

All code is executed in Docker[9] containers for simplified distribution and deployment. Docker is a lightweight system to run applications in a contained environment. It allows the user to build containers based on known operating systems (such as Ubuntu) and then install the tools needed for that specific application. The containers can then be executed on the same node (server) and share the available resources. In this way application with completely different needs can be executed on the same infrastructure. It is also relatively easy to scale up the capability of the application as the workload increases.

3.6.3 Amazon Web Servies - AWS

The Docker containers were deployed with different services from Amazon Web Services (AWS)[10]. The servers used can be found in table 3. The Docker containers were controlled via the Docker Cloud[11] service for container management.

Table 3: The different servers used to run the different components of the system.

Instance type	vCPU	Memory (GiB)
M4.large	2	8
C4.Xlarge	4	7.5
R3.8Xlarge	32	244

3.6.4 The API

The API handles requests for invoking new cluster calculations and retrieving the neighbors of a certain document (amongst many things). The API comprises of containers for handling requests and worker containers for performing heavier tasks. In order to manage job priorities and queueing, a Redis backed job queue

[6] https://www.mongodb.com
[7] https://redislabs.com
[8] https://aws.amazon.com/s3
[9] https://www.docker.com
[10] https://aws.amazon.com
[11] https://cloud.docker.com

was used to distribute the workload between the worker containers. This ensured low response times for the API while still allowing for larger operations. The API was built on NodeJS[12], with a Mongo database and Elastic Search[13] on top for creating a search index.

[12] https://nodejs.org
[13] https://www.elastic.co/products/elasticsearch

4 Results

This section presents the results of the project. It begins by presenting some information about the ground truth labels that were used. It then goes on to present results in sections related to their different methods. K-Means, SVM and the comparison section contains mostly data and plots while Similar Documents and Visualizing Clusters contains more reflections and images.

4.1 Ground truth labels

The ground truth labels for level 0 and 1 can be found in table 4 and 5.

Table 4: Number of documents belonging to each label according to the ground truth. Level = 0.

Label	Count	Percent
CompanyConcept	9885	61.6 %
Performance	4576	28.5 %
RealLifeExamples	1578	9.8 %

Table 5: Number of documents belonging to each label according to the ground truth. Level = 1. Note how almost 98 % of all documents belong to the four most popular labels.

Label	Count	Percent
CompanyConcept/SiteCollectionDocuments	9792	61.1 %
Performance/Lists	3302	20.6 %
RealLifeExamples/SiteCollectionDocuments	1461	9.1 %
Performance/SiteCollectionDocuments	1274	7.9 %
RealLifeExamples/MLDocuments	105	0.7 %
CompanyConcept/Lists	50	0.3 %
CompanyConcept/Articles	26	0.2 %
CompanyConcept/SiteCollectionImages	16	0.1 %
RealLifeExamples/SiteCollectionImages	11	0.1 %
CompanyConcept/Documents	1	0.0 %
RealLifeExamples/MarketingComm	1	0.0 %

4.2 K-Means (Sherlock)

The data used to create all plots in this section can be found in tables 9 and 10. Figures 5 and 6 were created from Sherlock runs that were told to create different numbers of clusters, seen on the x axis. The metrics purity and entropy can be calculated per clustering in our system, that is why they are presented separately from the others. The metrics were calculated with the "level" (explained in 3.4.2) set to zero, producing three different labels, and to one, producing eleven different labels. [14]

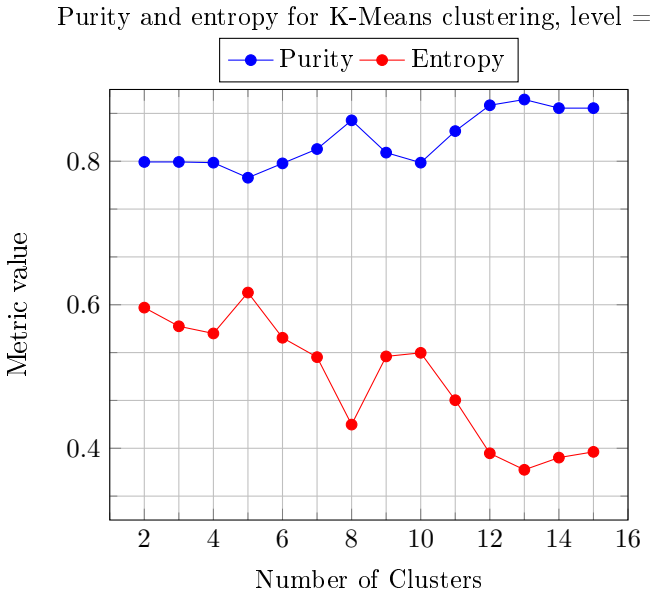

Figure 5: Purity and entropy for K-Means clustering. We expect a local maximum for purity and minimum for entropy at nc = 3, since that is how many labels we have at level zero. Note that a low entropy means a better clustering.

[14] Level two creates 93 different clusters and was therefore excluded.

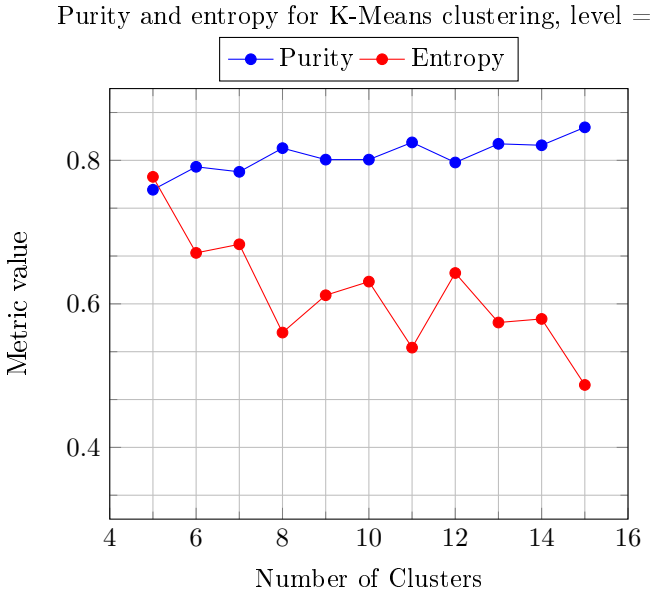

Figure 6: Purity and entropy for K-Means clustering. We expect a local maximum for purity and minimum for entropy at nc = 11, since that is how many labels we have at level one. Note that a low entropy means a better clustering.

Homogeneity, completeness and V-measure scores for K-Means clustering, level = 0.

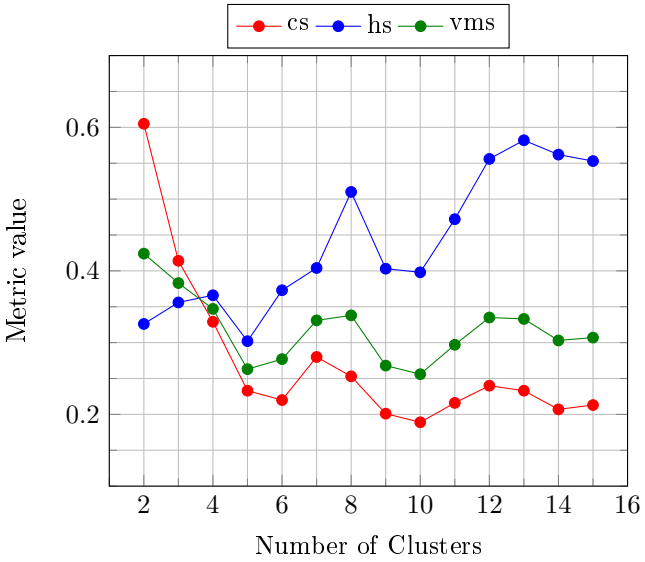

Figure 7: Homogeneity, completeness and V-measure scores for K-Means clustering with different number of clusters. Metrics calculated with level = 0, meaning there are three labels.

The next two figures (7 and 8) show the three correlated metrics homogeneity, completeness and V-measure score (hs, cs and vms). Just as before the metrics were calculated with the "level" set to zero and to one.

The final two figures for K-Means 9 and 10 show all the "broad" metrics for cluster performance. Just as before the metrics were calculated with the "level" set to zero and to one.

Homogeneity, completeness and V-measure scores for K-Means clustering, level = 1.

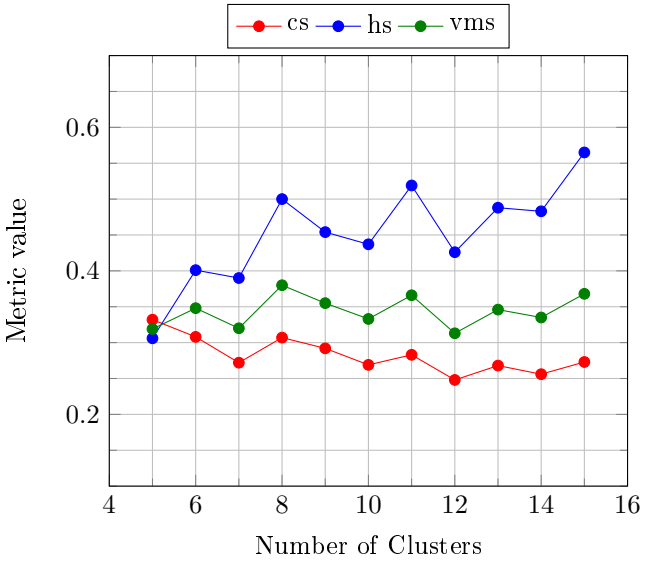

Figure 8: Homogeneity, completeness and V-measure scores for K-Means clustering with different number of clusters. Metrics calculated with level = 1, meaning there are eleven labels. We therefore expect a local maximum at nc = 11.

Table 6: Purity, entropy and size for individual K-Means clusters with number of clusters set to three 3 and level set to 0. Cluster 2 is perfet and cluster 1 is almost perfect.

cluster	**purity**	**entropy**	**size**
0	0.723	0.782	11664
1	0.999	0.008	2929
2	1.000	0.000	1446

For some limits and values for number of clusters (nc) the individual cluster purity, entropy and size are interesting to discuss. This data is displayed in tables 6, 7 and 8.

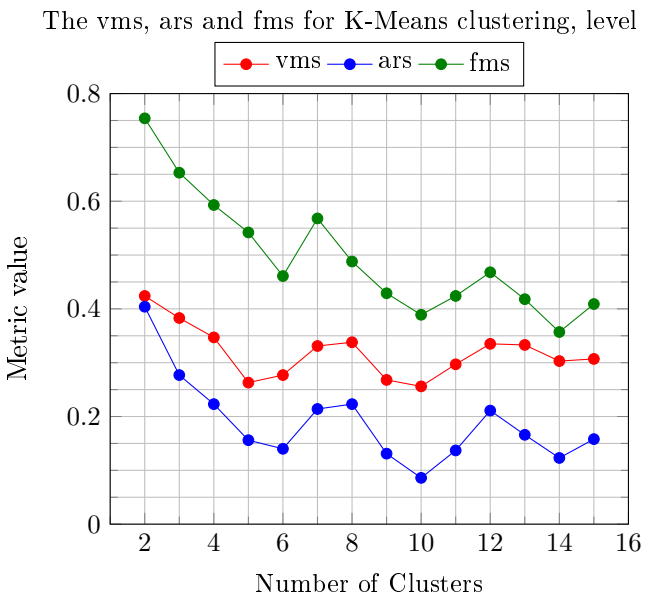

Figure 9: V-measure score, Adjusted Rand score and Fowlkes Mallows score for K-Means clustering. Local maximums are exected at nc = 3, since that is how many labels there are at level zero. Note that fms is unreliable for low values of nc.

Table 7: Purity, entropy and size for K-Means clusters with number of clusters set to 8 and level set to 0.

cluster	**purity**	**entropy**	**size**
0	0.719	0.763	3085
1	1.000	0.000	1527
2	1.000	0.000	1362
3	0.809	0.593	1475
4	0.839	0.531	6158
5	0.999	0.009	866
6	0.864	0.408	1070
7	1.000	0.000	496

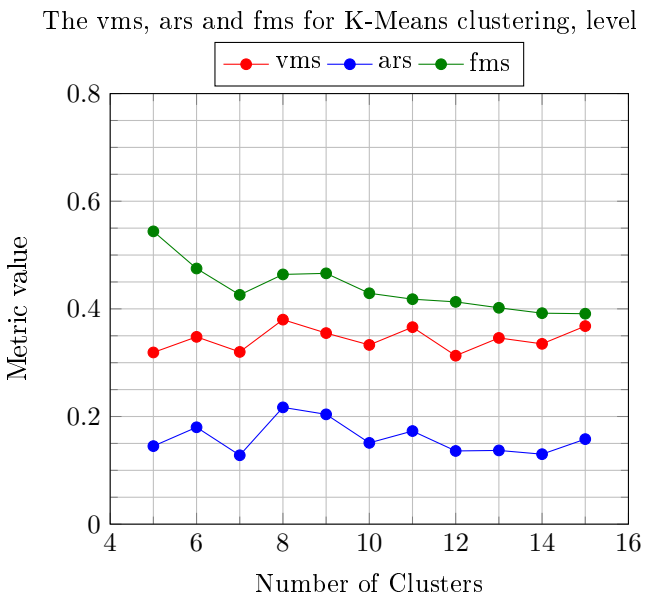

Figure 10: V-measure score, Adjusted Rand score and Fowlkes Mallows score for K-Means clustering. Local maximums are expected at nc = 11, since that is how many labels there are at level one. Note that fms is unreliable for low values of nc.

Table 8: Purity, entropy and size for K-Means clusters with number of clusters set to 11 and level set to 1.

cluster	**purity**	**entropy**	**size**
0	0.996	0.026	496
1	1.000	0.000	411
2	0.863	0.421	1057
3	0.961	0.186	516
4	0.959	0.201	615
5	0.990	0.061	866
6	0.999	0.007	1116
7	1.000	0.000	1317
8	0.527	1.115	1394
9	0.790	0.716	5367
10	0.715	0.872	2884

Table 9: All metrics for K-Means clustering with number of clusters set to different values. Metrics were calculated with level set to 0.

nc	purity	entropy	hs	cs	vms	ars	fms
2	0.799	0.596	0.326	0.605	0.424	0.404	0.754
3	0.799	0.570	0.356	0.414	0.383	0.277	0.653
4	0.798	0.560	0.366	0.329	0.347	0.223	0.593
5	0.777	0.617	0.302	0.233	0.263	0.156	0.542
6	0.797	0.554	0.373	0.220	0.277	0.140	0.461
7	0.817	0.527	0.404	0.280	0.331	0.214	0.568
8	0.857	0.433	0.510	0.253	0.338	0.223	0.488
9	0.812	0.528	0.403	0.201	0.268	0.131	0.429
10	0.798	0.533	0.398	0.189	0.256	0.086	0.389
11	0.842	0.467	0.472	0.216	0.297	0.137	0.424
12	0.878	0.393	0.556	0.240	0.335	0.211	0.468
13	0.886	0.370	0.582	0.233	0.333	0.166	0.418
14	0.874	0.387	0.562	0.207	0.303	0.123	0.357
15	0.874	0.395	0.553	0.213	0.307	0.158	0.409

Table 10: All metrics for K-Means clustering with number of clusters set to different values. Metrics were calculated with level set to 1.

n	purity	entropy	hs	cs	vms	ars	fms
5	0.759	0.777	0.306	0.332	0.319	0.145	0.544
6	0.791	0.671	0.401	0.308	0.348	0.180	0.475
7	0.784	0.683	0.390	0.272	0.320	0.128	0.426
8	0.817	0.560	0.500	0.307	0.380	0.217	0.464
9	0.801	0.612	0.454	0.292	0.355	0.204	0.466
10	0.801	0.631	0.437	0.269	0.333	0.151	0.429
11	0.825	0.539	0.519	0.283	0.366	0.173	0.418
12	0.797	0.643	0.426	0.248	0.313	0.136	0.413
13	0.823	0.574	0.488	0.268	0.346	0.137	0.402
14	0.821	0.579	0.483	0.256	0.335	0.130	0.392
15	0.846	0.487	0.565	0.273	0.368	0.158	0.391

4.3 SVM (Mycroft)

In this section all metrics for the SVM classification are presented. The data has been calculated using the test data (25 % of the data) from the cross-validate split, and not the training data.

Table 11 shows the number of clusters created for different SVM methods.

Table 11: Number of distinct labels categorized by the SVM by level and kernel function. For level 0 the documents had 3 distinct labels during training and validation. As we can see in this table the linear kernel function managed to separate and predict all the 3 different classes. For level 1 the number of distinct labels was 11.

Level	**Linear**	**RBF**	**SIGMOID**	**Poly**
Level 0	3	2	1	1
Level 1	6	2	1	1

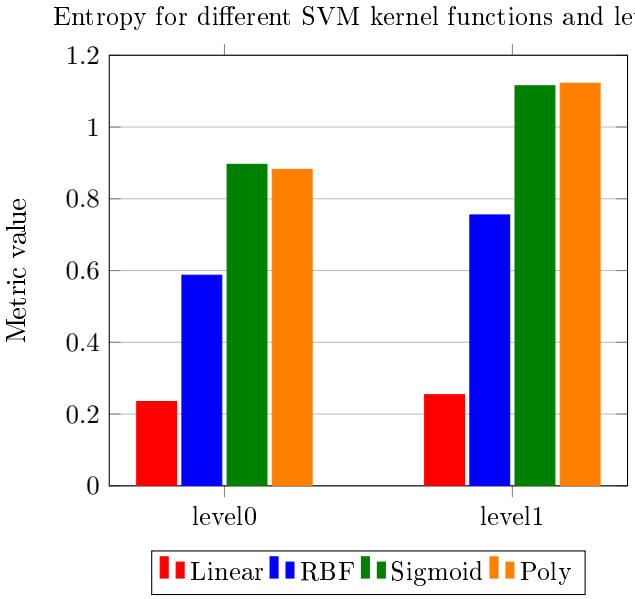

Figure 11: The entropy calculated for the different kernel functions at two different levels. Level 0 corresponds to 3 distinct labels during training, level 1 corresponds to 11 distinct labels during training. Interesting to note here is that the entropy for the linear kernel function is much lower than the others.

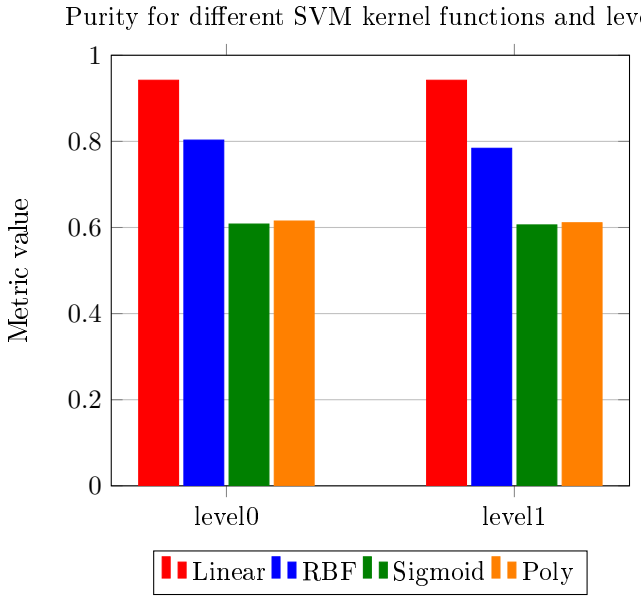

Figure 12: The purity calculated for the different kernel functions at two different levels. Level 0 corresponds to 3 distinct labels during training, level 1 corresponds to 11 distinct labels during training. Interesting to note here is that the entropy for the linear kernel function is much lower than the others.

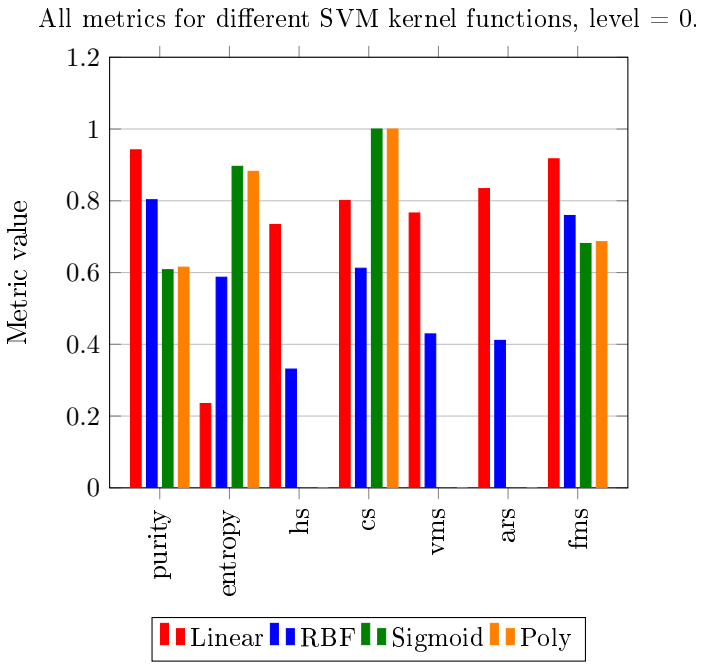

Figure 13: The other metrics for level 0. This is table 12 in plot format. Note that for metrics but entropy a high value is a good score.

Figures 11 and 12 show the purities and entropies of the SVM classification. Summaries of all metrics, for level 0 and level 1, can be found in figures 13 and 14. The underlying data can bee seen in tables 12 and 13.

Table 12: All metrics for SVM clustering with different kernels. Level is set to 0.

Measure	**Linear**	**RBF**	**SIGMOID**	**Poly**
purity	0.942	0.803	0.608	0.615
entropy	0.235	0.587	0.896	0.882
hs	0.734	0.331	0.000	0.000
cs	0.801	0.612	1.000	1.000
vms	0.766	0.429	0.000	0.000
ars	0.834	0.411	0.000	0.000
fms	0.917	0.759	0.681	0.686

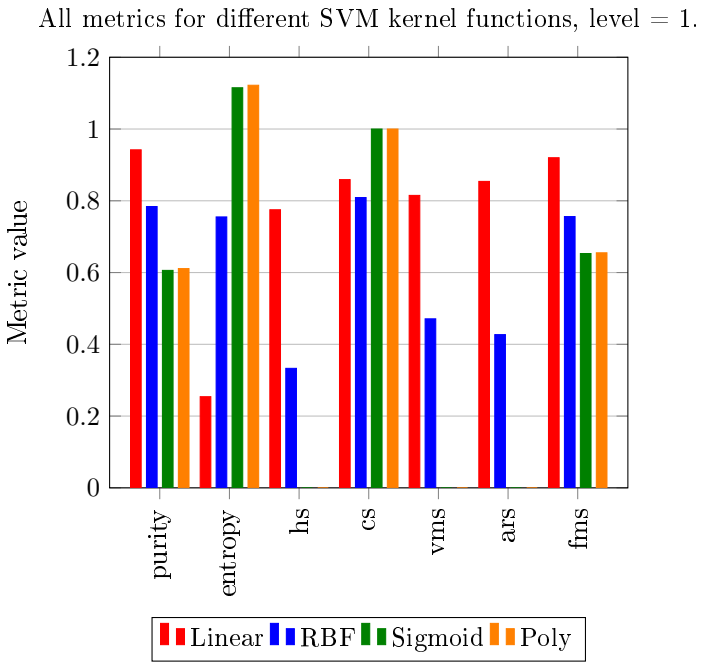

Figure 14: The other metrics for level 1. This is table 13 in plot format. Note that for metrics but entropy a high value is a good score.

Table 13: All metrics for SVM clustering with different kernels. Level is set to 1.

Measure	**Linear**	**RBF**	**SIGMOID**	**Poly**
purity	0.942	0.784	0.606	0.611
entropy	0.254	0.755	1.115	1.122
hs	0.775	0.333	0.000	0.000
cs	0.859	0.809	1.000	1.000
vms	0.815	0.471	0.000	0.000
ars	0.854	0.427	0.000	0.000
fms	0.920	0.756	0.653	0.655

4.4 Comparing K-Means and SVM

In order to compare the two methods some plots were created. They display best selection of K-Means computations compared with SVM computations, in terms of all of the previously discussed metrics. Figures 15 and 16 show the more general metrics in order to asses which clustering is better, and 17 and 18 the more aspect specific metrics in order to analyze how they differ.

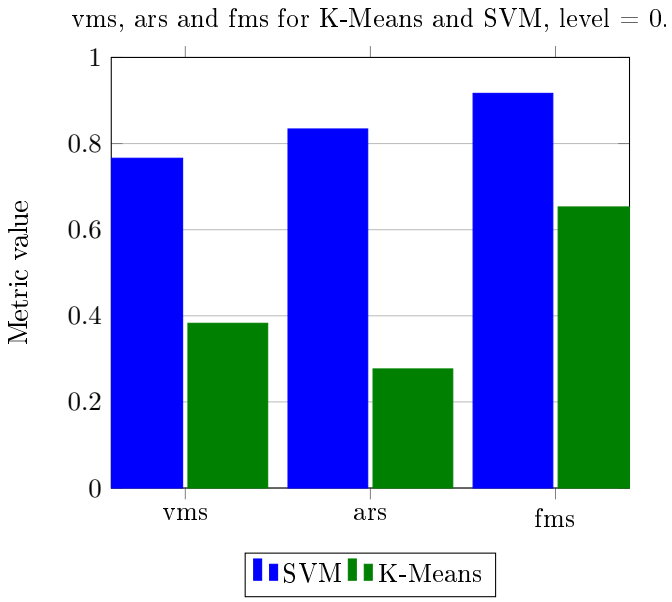

Figure 15: V-measure, Adjusted Rand index and Fowlkes Mallows score for K-Means and SVM, calculated with level = 0 (3 labels).

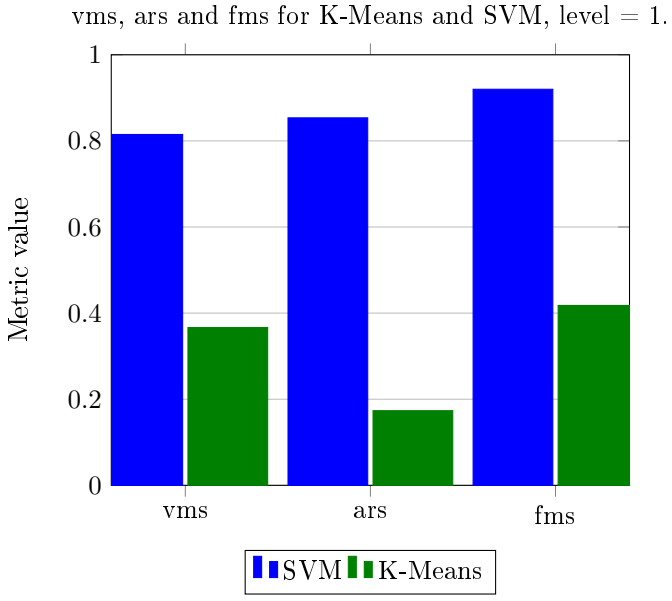

Figure 16: V-measure, Adjusted Rand index and Fowlkes Mallows score for K-Means and SVM, calculated with level = 1 (11 labels).

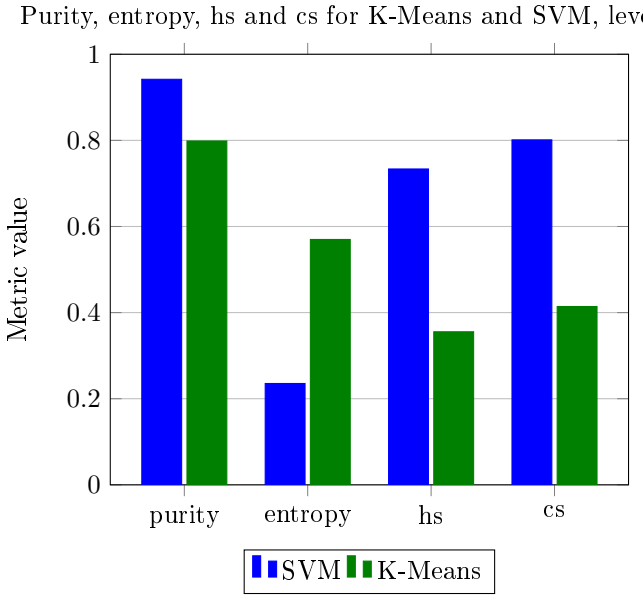

Figure 17: Purity, Entropy, Homogeneity and Completeness scores for K-Means and SVM, calculated with level = 0 (3 labels).

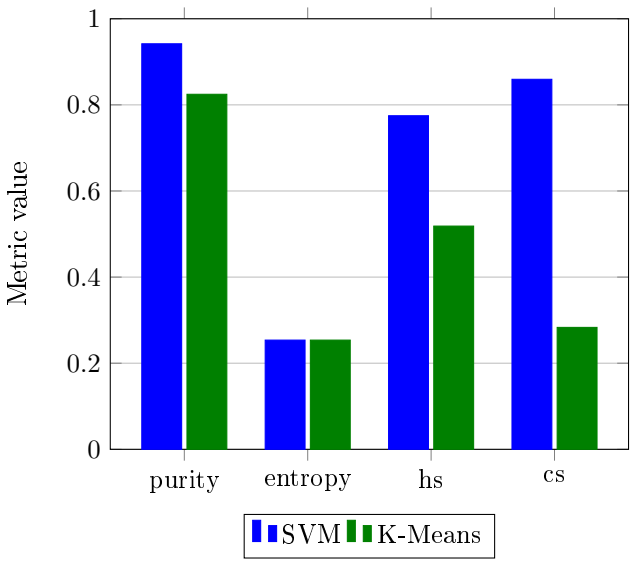

Figure 18: Purity, Entropy, Homogeneity and Completeness scores for K-Means and SVM, calculated with level = 1 (11 labels).

4.5 Similar documents (Deduction)

There are no quantifiable results on the "similar documents" part of the project since other parts were prioritised. However, the screenshot in figure 19 shows how the results are displayed on the platform.

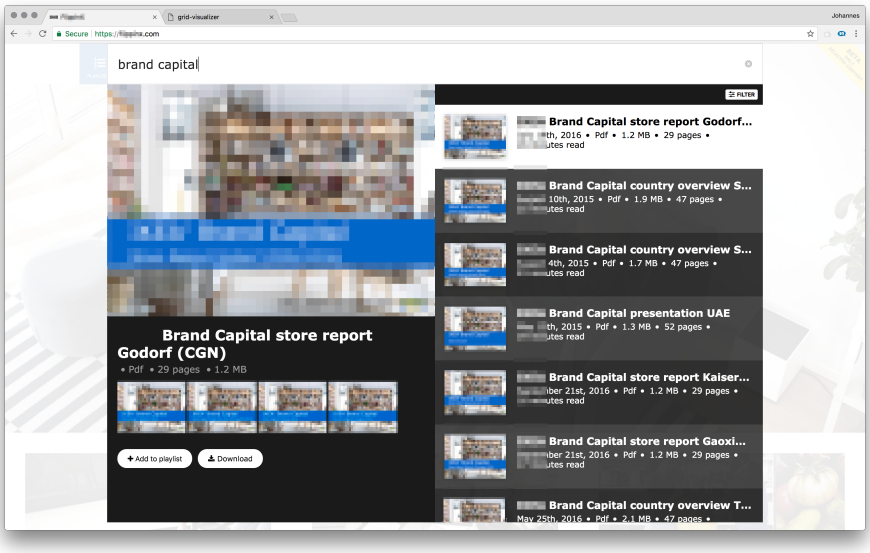

Figure 19: Screenshot from the platform, showing the related documents feature in application.

4.6 Visualizing Clusters

4.6.1 Magnifier

Since Magnifier is "just" a tool for navigating and examining the corpus and clustering, there is no mathematical presentation of it to present. A few screenshots were created to visualize the tool, in order to be able to discuss its usefulness.

4.6.2 Grid cluster preview

The grid cluster preview is an alternate way to view and assess the clustering. It randomly selects a number of documents from each cluster and then displays their first page as a thumbnail image. Note that some documents do not have previews available. The visualizations can be found in the appendix, figures 23 to 39. The previews have been pixelated in this report.

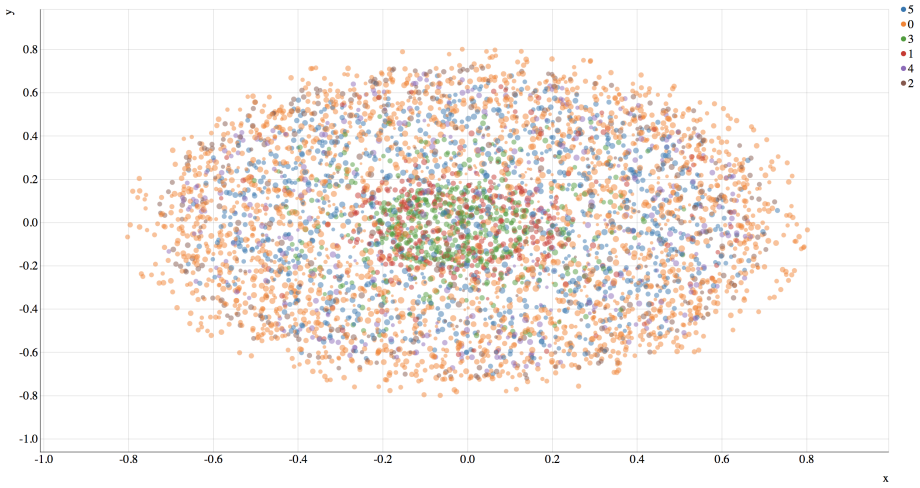

Figure 20: Screenshot from Magnifier, the MDS visualisation tool. Z-axis is indicated by dot size, cluster assignment is indicated by dot color. For large amounts of documents the picture becomes very hard to interpret.

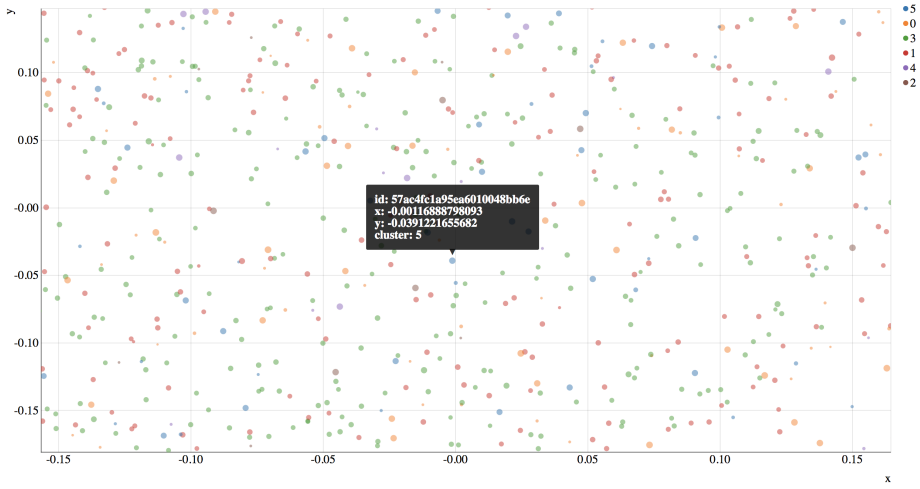

Figure 21: Screenshot from Magnifier, the MDS visualisation tool. Z-axis is indicated by dot size, cluster assignment is indicated by dot color. A zoomed in view is more useful for comparing documents. Clicking a dot once shows more info, clicking a dot twice opens the document preview.

5 Discussion

This section contains the discussion part of the project. It begins by discussing the data presented in the result section for the two clustering methods and the comparison. It moves on to discuss the similar documents feature, the visualizations, and the architecture. Finally, it contains some discussion related to the method itself, and some hindsight reflections about the project.

5.1 K-Means

One thing worth mentioning before looking at the actual data is that K-Means is initiated with a random assignment. This is different for each run of the clustering algorithm and may therefor result in different outcomes even though the input data is consistent. The series of clusterings were selected for being "good"; not extraordinarily so, but ones with good overall metrics. Additional clusterings were calculated and most of the trends noted above occur in all of them. They will not, however, be presented here.

5.1.1 Purity and entropy

Purity and entropy for level = 0 (figure 5) In the purity and entropy plots we were expecting, or hoping, to see a local minimum of entropy and a local maximum of purity where number of clusters equals 3 (from now on abbreviated "nc") and 11 respectively, since that is the number of distinct labels corresponding to level 0 and 1. We also expected purity to increase and entropy to decrease with nc.

Figure 5 does not match our expectations. We see an increase of purity and a decrease of entropy for larger values of nc, but we see no significant change around nc = 3. When looking closer at the clusters it is clear that it does not succeed very well at its task. The three clusters it creates (named 0, 1 and 2) can be described as follows (data can be found in table 6).

Cluster 0 has a quite high entropy (0.782) and a quite low purity (0.723). It contains all documents from the label `RealLifeExamples`, but also documents from the other labels. Containing 11664 documents it is the largest in size, and appears to contain most documents that simply did not fit elsewhere.

Cluster 1 has a very very low entropy (0.008) and a very very high purity (0.999). It contains 2926 documents with the label `Performance` (which is about 60 % of the documents with that label) and 3 documents with the label `CompanyConcept`.

Cluster 2 is perfect in terms of entropy and purity, since it only contains documents from a single label (`CompanyConcept`). That is, however, the label containing the largest amount of documents and far from all of them are found in this cluster.

In summary, the clustering with nc = 3 has managed to create two quite good clusters (in terms of purity and entropy, that is) and then placed the remaining ones in one cluster. Let us have a look at what is happening when we increase nc.

The local max and min we expected at nc = 3 instead appears at nc = 8 (data can be found in table 7).

This clustering contains three perfect clusters. Clusters 1 and 7 only contain the label `Performance`, and cluster 2 only contains the label `CompanyConcept`. Cluster 5 is almost perfect, and also contains `Performance` documents.

The remaining clusters have entropies in the range 0.408 - 0.763 and purities in the range 0.719 - 0.864. They all contain documents from all three labels, two of them contain mostly documents from `CompanyConcept` and one mostly from `Performance`.

Purity and entropy for level = 1 (figure 6) The next plot, figure 6, looked more like expected. It finds the same local minima of entropy and maxima of purity for nc = 8 as discussed in the previous paragraph, but it also finds the expected max/min at nc = 11 that we expect from level 1 (which results in 11 distinct labels). The analysis of nc = 8 might be more interesting now that we have a more specified set of labels (the data used to create these plots can be found in table 8.

Clusters 1 (411 items, label is `/Performance/Lists`) and 7 (1317 items, label is `/CompanyConcept/SiteCollectionDocuments`) are perfect. Clusters 10, 9, 8 and 2 are the worst with the current metrics and level. However, that might have an explanation.

In all of them, the majority of documents are labeled with the labels `/Performance/SiteCollectionDocuments`, `/RealLifeExamples/SiteCollectionDocuments` and `/CompanyConcept/SiteCollectionDocuments`. These labels are different since we are using the entire path, but they all belong to folders with the name `SiteCollectionDocuments`. If we had a more complex system for the labels and/or the metric we could account for this and see that the clustering may be even better than we thought. On the other hand, the three aforementioned labels together make up almost 80 % of the corpus, so the more probable explanation may be that it is just randomly distributed. There is not enough data to be sure.

The remaining clusters have entropies in the range 0.007 - 0.201 and purities in the range 0.959 - 0.999, which mean that they are pretty good clusterings.

5.1.2 Completeness, homogeneity and V-measure

Completeness, homogeneity and V-measure for level = 0 (figure 7) The next group of metrics belongs together since the metrics measure different aspects of a clustering. It is easy to maximize one on the cost of the other, and vice versa. That is why we want to compare them to each other, and also consider their harmonic mean V-measure.

The fact that there is nothing significant happening around nc = 3 does not surprise us this time since we already discovered this flaw of the clustering when investigating purity and entropy. Note that the clustering is exactly the same as in figure 5, it is just the metrics that have been replaced.

The min/max value of entropy and purity for nc = 8 appears in this graph as well. It is very distinct in the homogeneity metric, but less so in completeness and V-measure. For completeness, the peak is actually at nc = 7, but that aspect of the clustering was not revealed by looking at purity and entropy. It means that while the clusters are less fragmented at nc = 8, the labels are less fragmented at nc = 7. Depending on the purpose of the clustering one might be more desirable than the other.

There also appears to be some kind of peak for both homogeneity and completeness around nc = 12 or 13, but it is difficult to say what that means since it is so small. In investigating the individual cluster labels it was difficult to make any strong conclusions. Since there is a maximum around eleven for the metrics at level = 1 as well, it may actually be a property of the corpus.

Completeness, homogeneity and V-measure for level = 1 (figure 8) Looking at limit = 1 we encounter few surprises, just as when considering purity and entropy. We notice two very clear peaks for homogeneity, the previously discussed one at nc = 8 and an expected one at nc = 11. This time completeness peaks at 8, which did not happen for limit = 0, but also at 11 as expected.

We also notice a very high homogeneity for nc = 15, but since this is the endpoint of our data it is difficult to say what that means. The homogeneity metric appears quite volatile for level = 1.

5.1.3 Other metrics

Other metrics for level = 0 (figure 9) We will now look at the more "general" metrics. While purity, entropy, completeness and homogeneity measure specific aspects of the clustering, V-measure, Adjusted Rand index and Fowlkes Mallows score (vms, ars and fms) attempt to give a well-rounded assessment of a clustering. This makes them more difficult to use to understand the nature of the clustering, but more suitable to compare the overall result.

From the earlier analysis, we already expect to see maximas for all metrics around nc = 7 and 8, and this is true for this group of metrics as well. Fms has a very distinct peak at 7 and vms and ars a slight peak at 8.

We also notice a quite distinct peak of fms and ars for nc = 12. It is the same one that has been observed before, so this makes it more probable that the corpus splits "extra neatly" into 11-12 categories, no matter the labels, but the data is far from conclusive.

Fms is not very useful when the number of clusters is low (as mentioned in section 2.8.9) so we will have to disregard its promising start. For all values of nc fms is greater than vms, that in its turn is greater than ars. This is true both in level 0 and level 1.

Other metrics for level = 1 (figure 10) For level 1 we notice a quite distinct minima for all three metrics at nc = 7. Aside from that, there is not really much to discuss. The maximum at nc = 11 is visible in vms and ars, but not in fms. The aforementioned maximum at nc = 15 is also present.

Overall the "other metrics" are difficult to discuss and compare. The purpose of them is rather to see which nc:s are "good" and which are not; which trends we noticed in the other metrics that were just minor, and which ones really makes a mark all the way out to the more general metrics.

5.1.4 Concluding remarks

Since we have not actually modified any other parameters of the K-Means model than the number of clusters, this is all basically an investigation of the corpus rather than the model itself. By sweeping and looking at the metrics and clusterings we have learned more about the corpus itself, and how it responds when

we try to categorize it. This is one of the reasons we have set aside so much space for this section. With that in mind, let us find out how this applies and compares to SVM.

5.2 SVM

Based on the theory presented in this report we expected to find that the more complex kernel functions would perform better than the simple linear one. The results from the computations show that this is not the case. We will start by discussing the results for each kernel function and then give a conclusion to our findings by comparing them.

5.2.1 Linear

In table 12 and 13 we notice that the entropy of the linear function is relatively low. As the authors of [7, p. 327] write, text data is high dimensional and rarely linearly separable. This is why we did not expect the metrics for the linear kernel function to be so good. What we expected to see here is that the linear classifier would have trouble to separate the different groups of documents. If we look at the purity and completeness score (cs) we can see that the classes are fairly clean. Looking further into what the classes look like we can see that for level 0 we have two major classes `/CompanyConcept` and `/Performance` which stand for 66% and 27% of the documents. The last class only contains 9% of the documents. Looking at level 1 we have one very dominant class `/CompanyConcept/SiteCollectionDocuments` that contains 64% of the documents. After that we have `/Performance/Lists` with 21% of the documents. The other classes are fairly small compared to these two.

5.2.2 RBF

This is the default kernel function. Compared to the performance of the other ones it has the second best results, beaten by the linear kernel function. Looking on the purity score for level 0 and 1 from table 12 and 13 we can see that it is not far from the high scores that the linear kernel function produced. Level 0 is just a notch better than level 1. What is more interesting to see is what happens with the entropy and homogeneity score (hs). For level 0 the jump from the linear version is not that big but for level 1 (with 11 labels) we have quite bad scores. If we take a look at table 11 we see that for level 0 it manages to predict two out of three classes but for level 1 only two out of eleven classes. Since the training labels are the full path (like `/RealLifeExamples/SiteCollectionDocuments`, `CompanyConcept/SiteCollectionDocuments` and `/Performance/SiteCollectionDocuments`) for the level 1 classification, there seems to be an underlying structure that we do not capture. For level one, one of the two classes contains all the `*/SiteCollectionDocuments` documents. But since the label is the full path the prediction receives a very low homogeneity score.

5.2.3 SIGMOID and Polynomial

We did not get any useful results with these kernel functions. Because of this, we did not dig any deeper into these kernel functions.

5.2.4 Comparing linear and rbf

The two different kernel functions worth comparing are the linear and the rbf. They both performed relatively well but they also gave results that we did not expect. One reason to why the linear classifier performs so well could be that the documents are given as groups of documents, or if you want, they are already in classes. If we look at the documents many of them are written based on a template, making them very similar. The next group of documents is based on another template making it easy to distinguish the differences between the two groups. In a situation where the documents are not based on templates it is tempting to think that the linear classifier would not have performed as well. This is, however, another investigation and not part of this thesis.

Given the hypothesis that documents are written based on templates it is easier to see that the linear kernel function performs well for level 0, but it performs really well even for level 1. As said before we have sub-levels in the labels for level 1 which seem to have a lot in common (the `*/SiteCollectionDocuments` labels). The linear kernel function manages to separate these very well but the rbf kernel function does not. The rbf kernel function puts all of the `*/SiteCollectionDocuments` documents in the same class. Whether this is bad or not is up for discussion. One could argue that the best thing is to classify all `*/SiteCollectionDocuments` into the same class, but then we are losing the information about their top domain (`CompanyConcept` etc.). It is up to the client to decide which way is best and what the logical approach is for the users to find and consume this information. Regardless of this, we note that the linear kernel function in this case manages to distinguish between the different classes a lot better than any other kernel function. To further improve the performance of the classifier it is tempting to try out different kernel functions and parameters but

> It is frequently the case that greater performance gains can be achieved from exploiting domain-specific text features than from changing from one machine learning method to another.

as Manning, Raghavan and Schütze write in [7, p. 335]. It is probably a better idea to explore what our specific domain looks like and use the different sub-levels of the labels.

Something to notice is that according to the ground truth a very large proportion of the documents belong to a small set of labels (as seen in tables 4 and 5). Another thing to note is as the authors of [5] find the SVM does not perform as well when the number of training samples is low. This is the case for the lower part of the labels in tables 4 and 4. To improve the results in this case we are back at learning more about our domain.

5.3 Comparison of methods

This comparison is bound to be quite unfair, since we are comparing a supervised method with an unsupervised method. Also, the label data we are benchmarking against is the kind we used to train the SVM model. We used cross-validate in order to not dope the system, but the comparison is still in favor of the SVM. The K-Means method may make a clustering that makes more sense in the TF-IDF, but if it is not the one the humans setting the labels thought about it is

not going to show up in any other metrics than purity and entropy. Our goal was simply to get an idea of how much of a difference supervision makes.

5.3.1 General metrics

For level 0 we see that SVM performs between two and three times better than K-Means for the metrics vms and ars. For fms SVM is only about 1.5 times better. It is difficult to discuss much about this. We expected SVM to be much more accurate than K-Means, and it is.

For level 1 SVM performs very similarly to level 0, just a little bit better (5 % on average). K-Means, on the other hand, performs considerably worse when faced with a larger number of clusters (25 % worse on average). This makes sense since the differences between documents in different clusters become more subtle. SVM counteracts that with receiving more complex training data, something K-Means can not.

5.3.2 Specific metrics

Now let us consider the more specific metrics. These are not suitable for deciding which method is better overall, but helps us to understand the differences between the clusterings.

For SVM hs and cs behave similarly to the more general metrics, they increase slightly from level 0 to level 1. Purity is unchanged, which is odd but certainly not impossible. Entropy interestingly enough increases, but not by much (remember that lower entropy means better clustering).

For K-Means both purity and hs increases, even though all general metrics decrease. So even though the clustering quality decreases overall when K-Means is forced to separate the corpus into more clusters, the homogeneity and purity increases. This is typical for these two metrics; the greater the number of clusters, the greater the accuracy. The entropy increases slightly, which is unexpected. Entropy usually decreases with the number of clusters. But the change is very small so it is difficult to draw any conclusions from it. Finally, cs decreases drastically (by approximately 30 %) which is expected when the number of clusters increases.

5.3.3 Summary

In summary, SVM beats K-Means in every metric and every clustering. It should come as no surprise that a supervised method greatly surpasses an unsupervised when it comes to being able to replicate the human idea of order. Now we have it quantified.

5.4 Similar documents (Deduction)

The similar documents feature was one of the first ones that the customer requested. It was also one of the easiest to implement. It does not even technically involve machine learning, all it is is a calculation of distances between rows in the TF-IDF matrix.

We looked for ways to evaluate it and improve it, but soon realised that we did not quite see the need for it. It did what it was required to do and it did it well. Also, since we did not have users yet, there was no source for training

data or data to benchmark against. The result was that the plans to improve the similar documents feature was postponed and everybody was still happy.

5.5 Visualizing clusters

5.5.1 Magnifier

Magnifier was the first way to interact with the result of the clusterings and classifications, before metrics or find similar documents was implemented. As such it was of great help. The ability to click document coordinates to show previews and scroll through the documents proved to be a very useful feature, especially when evaluating if the distance between two document was accurate. Navigating through the corpus in Magnifier taught us a lot about the dataset.

One version of Magnifier was actually implemented as an exploration view of the corpus for content creators. They could use it to navigate from one document to its neighbors, to find documents that were probably connected, and ones that were probably duplicates.

The number of documents quickly became a limitation for the graph. Keeping track of the relationships between a hundred document was no problem, but when plotting the entire corpus it quickly became difficult to survey. While the MDS did its job it is not easy to make sense of dozens of dimensions projected in 2D (or 3D, when using dot size) so the shapes and locations of the dots did not tell us much in the end.

Towards the end of the project, we used Magnifier less and less, in favor of the metrics that told us more about the specifics of the computations. But magnifier has helped us a lot, especially early in the project while evaluating the TF-IDF model and the similar documents features. It would probably be possible to develop it into an even more powerful tool for data analysis, especially if we could find a way to limit the number of documents to show in each view.

5.5.2 Grid cluster preview

The grid cluster preview was just a simple tool we created to be able to present our progress to the customer and our supervisors. It did, however, end up encouraging us and proved to be an interesting tool to quickly assess and analyse a clustering. It is not as scientific as looking at metrics and plots of coordinates, and it is very difficult to use to assess small differences in quality. But we found out that the end customer was a lot more impressed with this method of visualization than trying to explain the mathematical formulation of Fowlkes Mallows Score and show plots.

Interestingly enough our impression from looking at the results from the grid cluster previews is that the K-Means clustering looks better than the SVM clustering. Clusters 0, 1, 2, 5, 6, 7 look very very clean (you can easily visually tell that the documents belong together). For svm several clusters look ”ok”, but not as nice as K-Means. It is quite difficult to draw conclusions from this. Since SVM is supervised its performance depends very strongly on the accuracy of the labels. For this particular clustering, we knew that SVM had an purity of 0.94. Just to double check we generated a grid cluster preview of the actual labels rather than a clustering, and could establish that they indeed look very

similar. One example is attached to this section, see figure 22. SVM is doing its job, but the labels are not doing it justice.

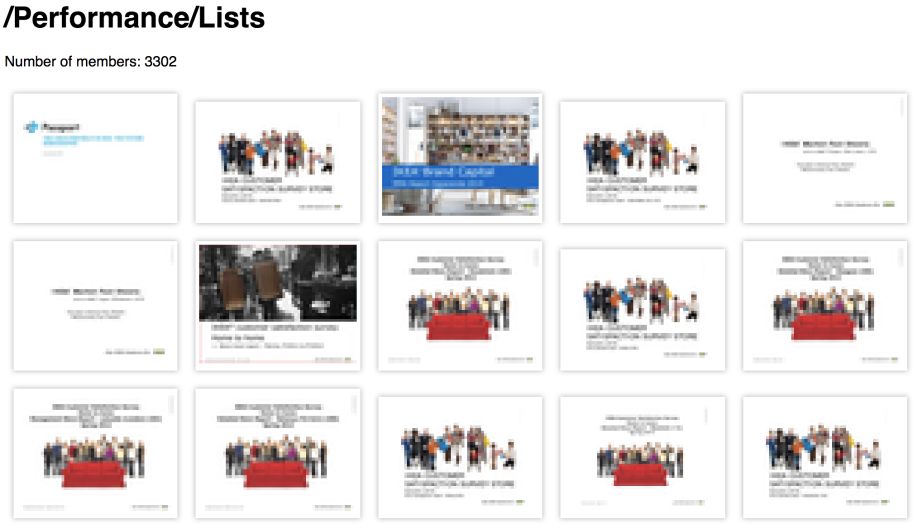

Figure 22: Visual representation of the label data.
Level = 1. Path: `/Performance/Lists`.

Something worth noting is that not all documents have previews generated, and the script is written to ignore those rather than displaying them. This is desirable for our application, but it should be noted that it might make the clustering appear better than it really is. If a specific type of document lacks previews it will not show up in the visualization, and that may cause a cluster to look purer than it really is. This will, however, be true for all clusterings, and should not impact comparisons.

Another interesting quality of this visualization method is that it shows an aspect that has been ignored in the ml system: the aesthetics and images of the documents. At first, some of the clusterings seemed "too easy", as they all looked the same. But the computer does not know that, it only looks at the word frequencies but is still able to cluster documents in an impressive way.

5.6 Metrics

We were quite satisfied with the metrics we decided to use in this project. Entropy and purity proved very helpful in that they could conveniently be calculated per individual cluster, and not only per clustering. They were also very intuitive and easy for us to understand and get a feeling for early on.

Making distinctions between different clusterings using homogeneity and completeness turned out to be more difficult than we hoped to. They still aided us in assessing the specific qualities of the clusterings we were looking at. Their definitions may not be as clear as for purity and entropy, but after getting used to them they started to make sense and helped us understand our clusterings.

The remaining metrics v-measure, adjusted rand score and Fowlkes Mallows

score turned out to be very similar, and quite hard to draw conclusions based on. We still decided to include all of them because three metrics do reveal more than one, and make the conclusions more robust. We could, however, probably have excluded one or two of them and ended up with similar conclusions.

5.7 Architecture

One of the big challenges of this thesis has been that we have been fully responsible for setting up and running the entire stack. From spinning up instances on Amazon, creating docker containers, setting up databases and last but not least deploying the different components. All of this while people were using the service. We have learned a lot from this process, some things that are difficult to learn anywhere else. It has, however, taken time away from other things that could otherwise have been explored more thoroughly.

5.8 The LEAN Method

We really appreciated striving for a LEAN method in this project. It has allowed us to always move in the direction that will maximise the results for the customer and users. In order to really follow the LEAN system we probably should have been more data-driven. We should have had more clear ways to evaluate how satisfied the customer and the users were with the system, and let this guide us. This was, however, difficult when the project was delayed so that we did not get the users we expected. We still did our best to be data-centric on the feedback from our customer, but it definitely made the process less LEAN.

It is a quite different approach to a project than what we are used to from LTH, where we usually are served the theoretical foundations required and a plan for the project ahead. This is a great method for learning material but it does not replicate the conditions we faced in the working life. At least from our experience so far. We think it was worth the occasional mishap, delays, and pivots to strive for a LEAN method. It certainly did not make writing this report and finishing the project easier, but we firmly believe it made the end product better and helped us to learn a lot more on the way.

Overall we are happy that we went with this approach rather than a more traditional academical one.

5.9 Challenges

In this section we will discuss some of the things we spent time on that did not make it to the final implementation.

5.9.1 word2vec

Starting out with word2vec was probably a good decision, even though it did not actually end up being a part of the thesis. It gave us the basic concepts of machine learning and it was thrilling to see results (although not useful ones) so early on. The issue was that we saw no clear way of transferring the term-term relationships into document-document relationships. We also investigated using doc2vec, a method that works on entire documents instead of terms but decided to focus on the TF-IDF based methods instead.

5.9.2 User data

This data does not relate as clearly as the playlist data to our corpus. It is therefore a lot more difficult to use as learning material. However, we knew there would be a lot more data of this type available than there would be playlists so we decided to see what we could extract from it. We analyzed user behavior and defined a scoring table for different behaviors. If a user downloads an item, the relationship between the search query and that item received five points. If a user clicks "show more result", the relationships between the search query and all visible documents is decreased by one point. From these relationships labels could be created. Since we did not receive enough users during the project we did not dig deeper into this.

5.9.3 Search optimization

In hindsight the search optimization part of the project was a mistake. It could be attributed to our eagerness to get started, being so encouraged by our first naive steps. There was no programmatically simple way to add machine learning data to the search algorithms of Elastic, and writing a new search engine would not lead to an improvement. The machine learning aspects of Elastic were more focused on finding anomalies in a corpus, especially over time. This is something we realized pretty early on, but it was not until we went to the Elastic{ON} conference we got the bigger picture of how ElasticSearch relates to the Elastic machine learning system. At least we were wise enough to realize our mistake pretty early and pivot towards more productive methods. As a bonus, a lot of the work we spend on researching and implementing search result metrics was useful for the cluster comparison performed later on.

6 Conclusions

This thesis asks the question "Can we replicate the human idea of structure to documents using software?". Answering an open question like this is never easy, but we have made some progress.

The unsupervised method K-Means definitely provides some promising results in itself; the results of some metrics are not bad, and the result from the grid-visualizer is quite impressive. When sweeping the number-of-clusters parameter in K-Means it finds some optimal values for the corpus. It is quite clear that in some ways K-Means is "smarter" than the training labels; it finds different, maybe better, ways to organize the data.

The supervised method SVM greatly surpasses K-Means on all our metrics. Given the results in this report, the SVM better replicates "the human idea of structure" and it is therefore the better method in this case. Since the SVM is supervised it also has a greater potential for improvement than K-Means. Choosing a level, or multiple levels combined, from the current training data could make a great difference. The supervised nature of the SVM also makes it a more interesting candidate for future expansions with playlists and user-generated data.

The main problem we had in this project was the lack of good training and evaluation data. Even though the metrics show some promising results the usability for the end users is hard to evaluate without the actual users. If we were to do this again we would spend more time on understanding the input data at an earlier stage than we did. Then again, we did not get the input data that we expected which is also something to learn from.

We think that we have succeeded in replicating *some* human idea of structure with our software solution. With a relatively simple understanding of machine learning and a surprisingly short amount of time, we believe that any company could set up a similar system to this one. However, it does take some deeper mathematical knowledge in order to get the most out of it. An understanding of the machine learning algorithm is important, but a deeper understanding of the dataset and labels may be even more important.

6.1 Future work

If somebody were to take over this project tomorrow, we would encourage them to spend more time examining the corpus. One way to do this could be to improve (and spend time in) the visualizing tools. A smaller, simpler version of Magnifier would be interesting to navigate around neighbors and learn the patterns of the clustering.

We also believe that there is a lot more to learn from the URL:s used as ground truth labels in the project. By combining different levels of these (maybe based on the K-Means clusterings) they could be greatly improved. Interviews with the people who uploaded the documents would help as well. Using URL:s (or folder structure in general) as labels for supervised machine learning could definitely be a thesis in itself since there are so many aspects to it.

Another approach would be to use the result from K-Means as training data for SVM, as discussed in the paper "Using unsupervised clustering approach to train the Support Vector Machine for text classification" [8]. Combining both

the K-Means data and some version of the labels we had access to would be very interesting.

Getting new labels by recording and processing real user data is perhaps the most exciting next step. When feedback is built into the system it will grow better as more people start using the product.

Finally, we have barely scratched the surface of the practical applications of the machine learning system. There are so many useful features for the admins and users of the information retrieval system that we can think of, and many more if the development is performed in collaboration with said people.

References

[1] Aggarwal, C. C., and Zhai, C. *Mining Text Data*. Springer, New York, 2012.

[2] Bird, Steven, E. L., and Klein, E. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[3] Borg, Groenen, I., and F., P. J. *Modern Multidimensional Scaling*. Springer, 2005.

[4] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering 9*, 3 (2007), 90–95.

[5] Islam, M. S., Jubayer, F. E. M., and Ahmed, S. I. A comparative study on different types of approaches to bengali document categorization. *CoRR abs/1701.08694* (2017).

[6] Mandal, A. K., and Sen, R. Supervised learning methods for bangla web document categorization. *CoRR abs/1410.2045* (2014).

[7] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[8] Niusha Shafiabady, L.H. Lee, R. R. V. K. N. A. A. D. I. *Using unsupervised clustering approach to train the Support Vector Machine for text classification*. Neurocomputing 211, 2015.

[9] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[10] Petersson, J. D3 zoomable scatterplot. `http://bl.ocks.org/peterssonjonas/4a0e7cb8d23231243e0e`. Accessed: 2017-08-31.

[11] Ries, E. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.

[12] Rosenberg, Hirschberg, A., and Julia. V-measure: A conditional entropy-based external cluster evaluation measure.

[13] scikit. Scikit learn support vector machines. `http://scikit-learn.org/stable/modules/svm.html#svm-kernels`. Accessed: 2018-03-01.

[14] Wagner, S., and Wagner, D. Comparing clusterings- an overview.

[15] Wang, Koopman, S., and Rob. Clustering articles based on semantic similarity. *CoRR abs/1702.04946* (2017).

[16] Wickelmaier, F. *An introduction to MDS*. Sound Quality Research Unit, Aalborg University, Aalborg, 2003.

Appendix

Graphical cluster overview

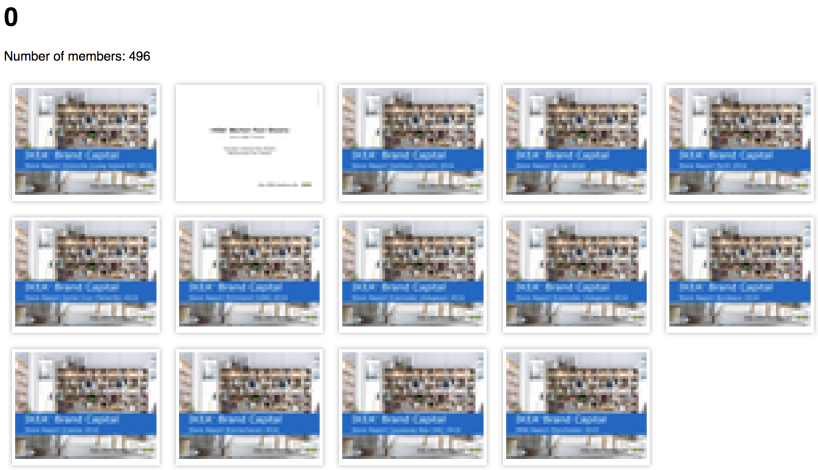

Figure 23: Visual representation of kmeans clustering. Level = 1. Cluster 0.

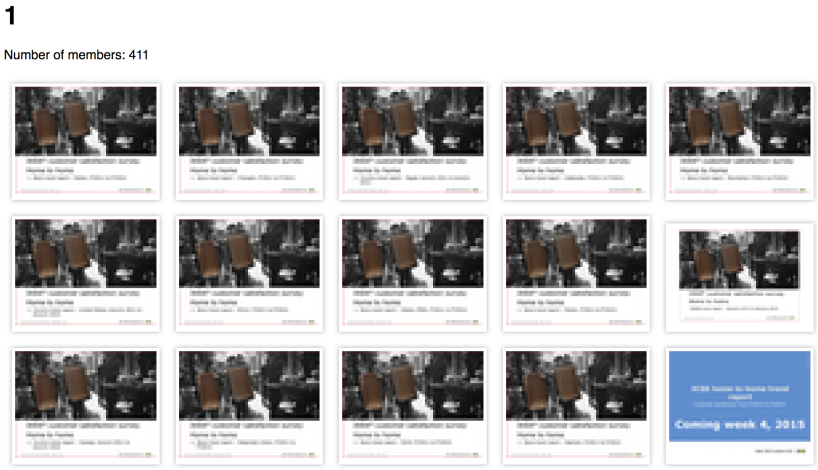

Figure 24: Visual representation of kmeans clustering. Level = 1. Cluster 1.

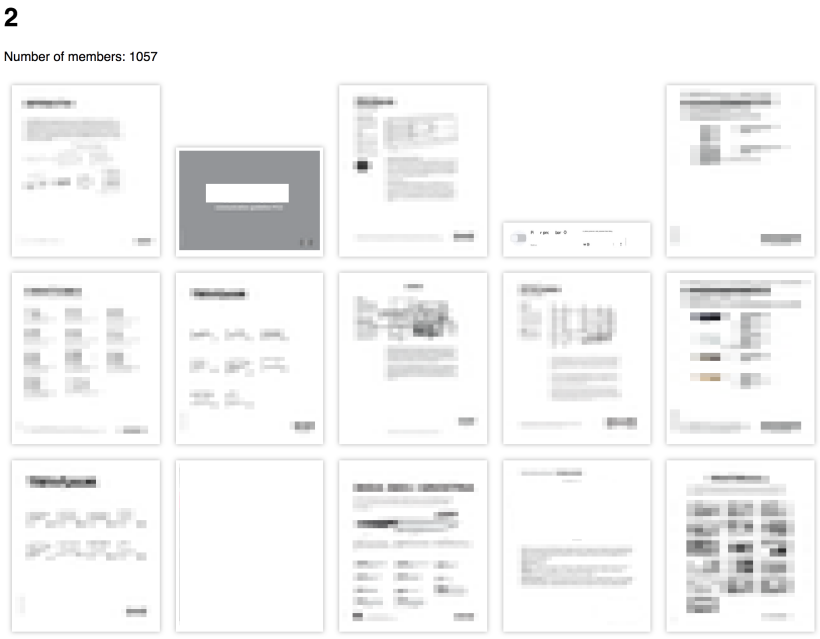

Figure 25: Visual representation of kmeans clustering. Level = 1. Cluster 2.

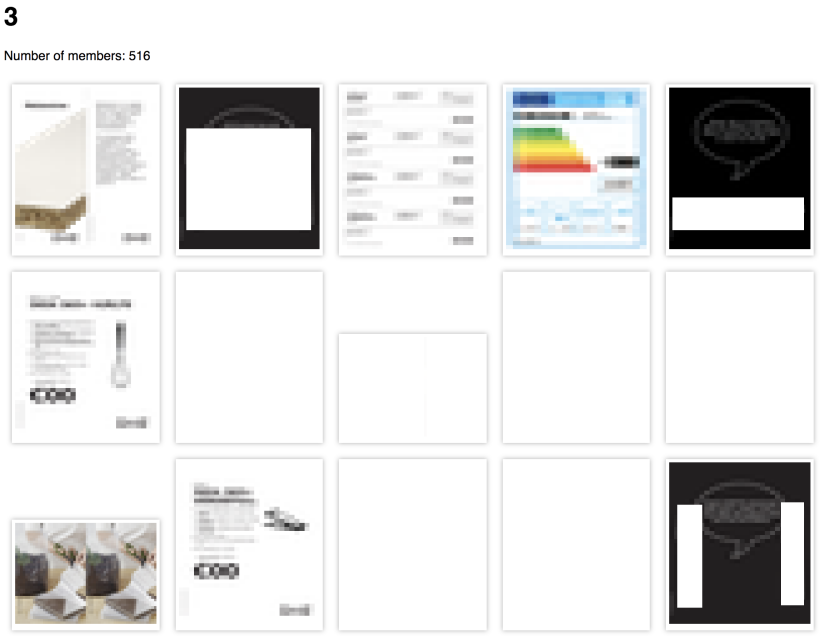

Figure 26: Visual representation of kmeans clustering. Level = 1. Cluster 3. This cluster was very good, but contained images that had to be hidden.

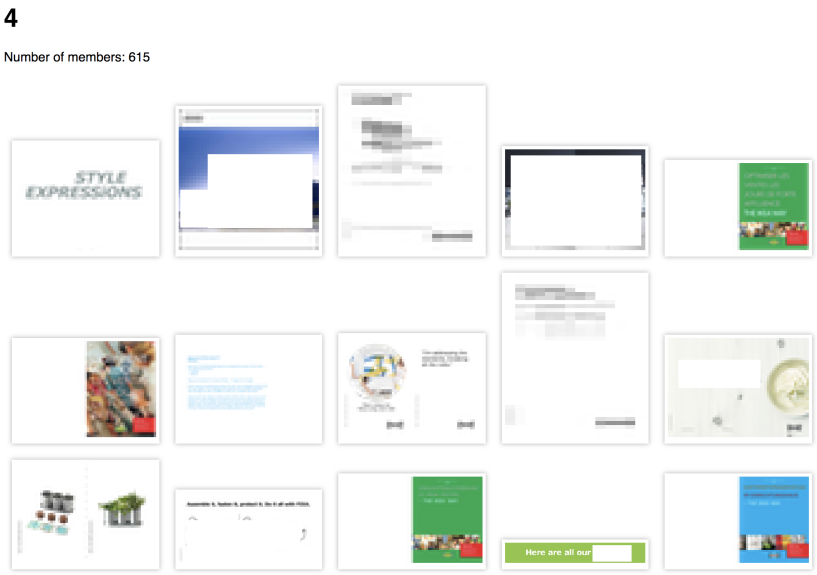

Figure 27: Visual representation of kmeans clustering. Level = 1. Cluster 4.

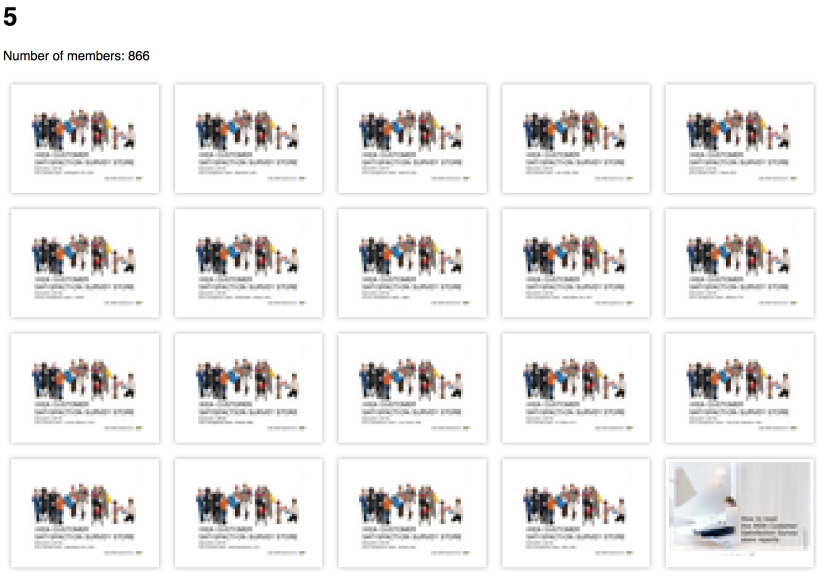

Figure 28: Visual representation of kmeans clustering. Level = 1. Cluster 5.

Figure 29: Visual representation of kmeans clustering. Level = 1. Cluster 6.

Figure 30: Visual representation of kmeans clustering. Level = 1. Cluster 7.

8

Number of members: 1394

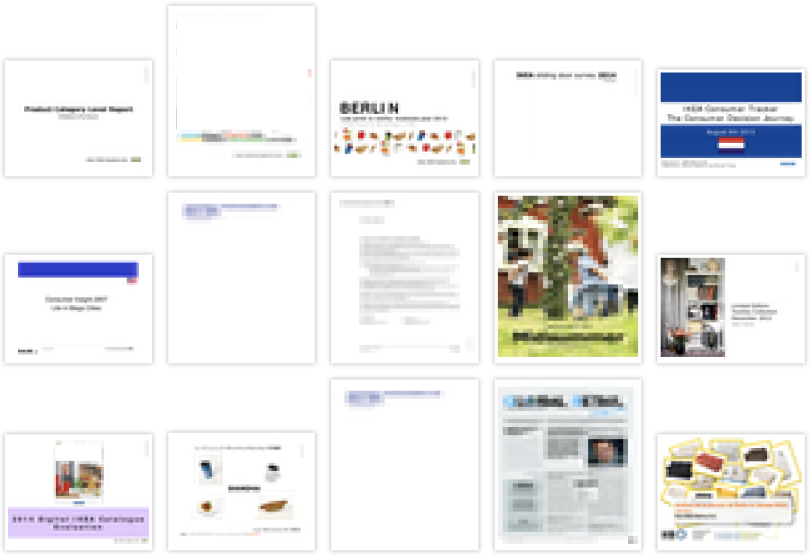

Figure 31: Visual representation of kmeans clustering. Level = 1. Cluster 8.

9

Number of members: 5367

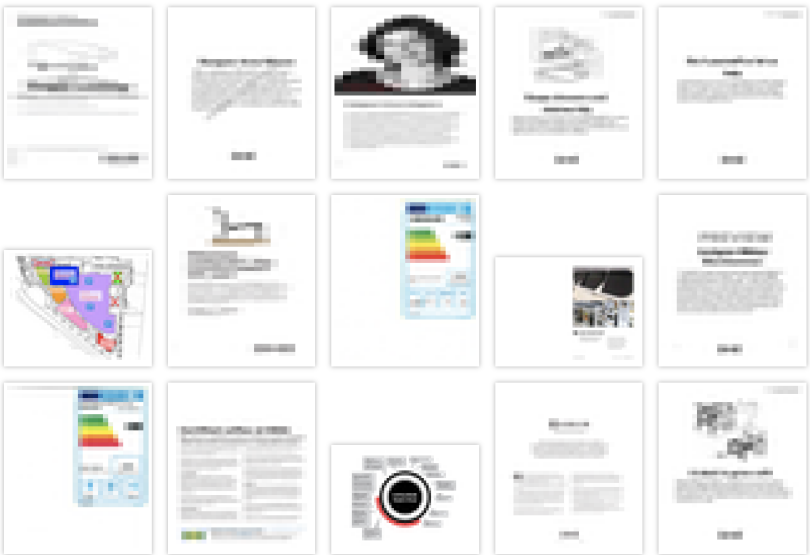

Figure 32: Visual representation of kmeans clustering. Level = 1. Cluster 9.

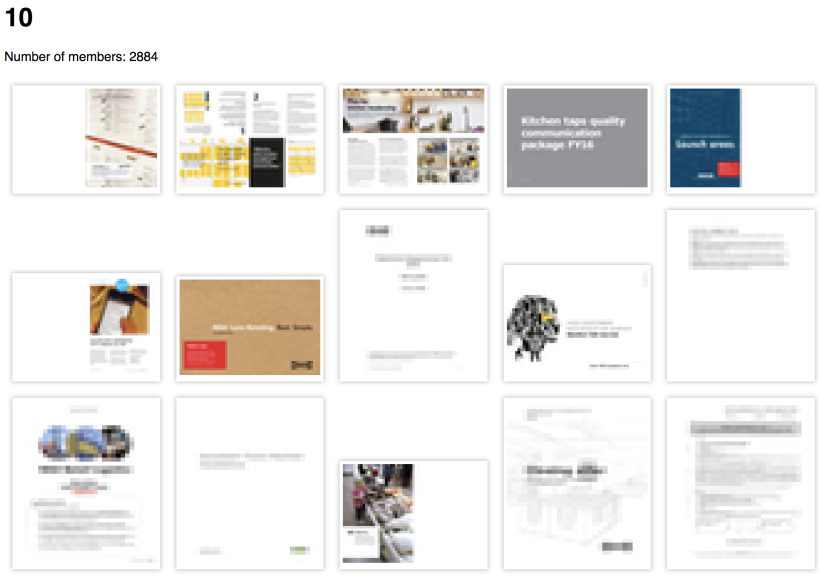

Figure 33: Visual representation of kmeans clustering. Level = 1. Cluster 10.

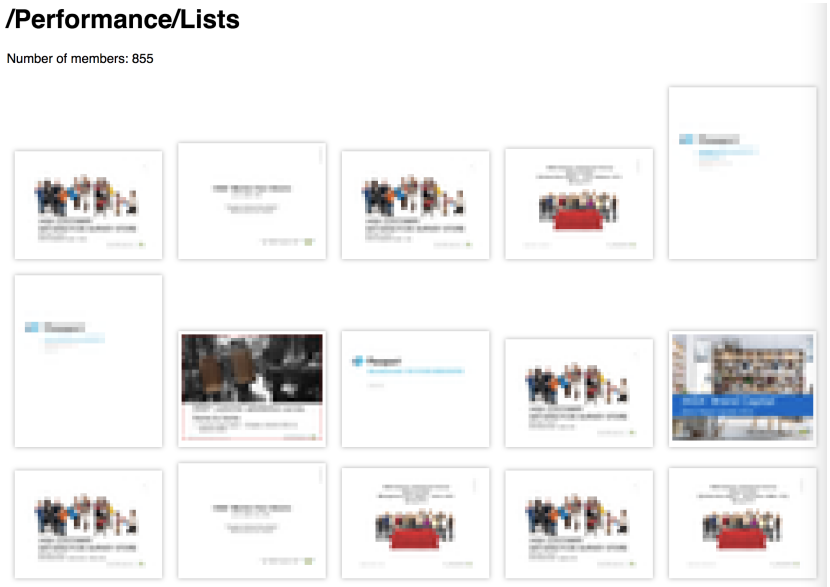

Figure 34: Visual representation of svm clustering. Level = 1. Cluster: /Performance/Lists.

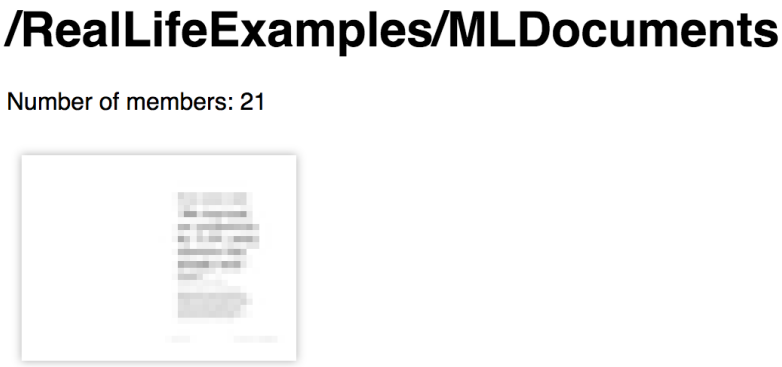

Figure 35: Visual representation of svm clustering. Level = 1. Cluster: /RealLifeExamples/MLDocuments. Previews were not available for the other 20 documents.

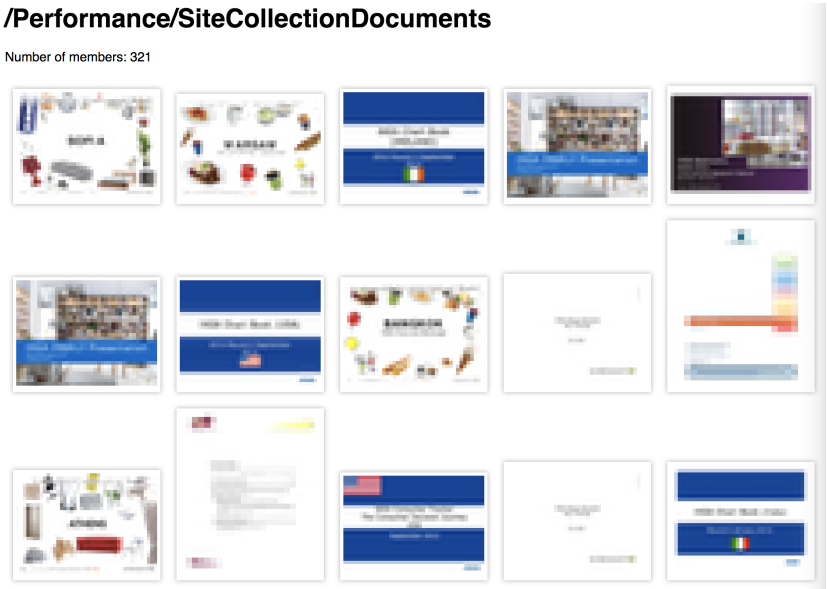

Figure 36: Visual representation of svm clustering. Level = 1. Cluster: /Performance/SiteCollectionDocuments.

Figure 37: Visual representation of svm clustering. Level = 1. Cluster: /CompanyConcept/Articles.

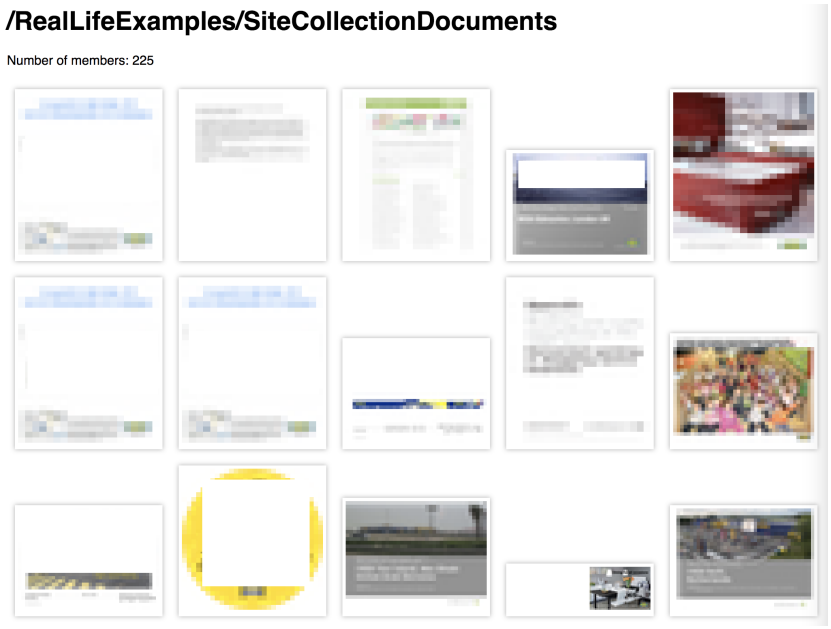

Figure 38: Visual representation of svm clustering. Level = 1. Cluster: /RealLifeExamples/SiteCollectionDocuments.

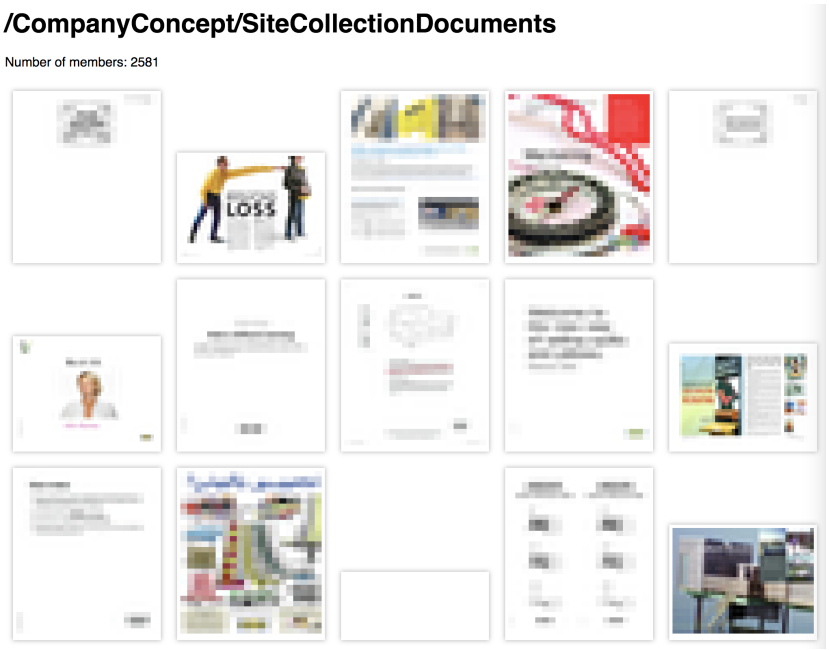

Figure 39: Visual representation of svm clustering.
Level = 1. Cluster:
/CompanyConcept/SiteCollectionDocuments.